

AD-A265 308



NAVAL POSTGRADUATE SCHOOL
Monterey, California

(2)



DTIC
ELECTE
JUN 02 1993
S B D

THESIS

**AN IMPLEMENTATION OF
TRAFFIC MONITORING FOR UNIX
NETWORK PERFORMANCE MANAGEMENT**

by

Fu Chen-Hua

March 1993

Thesis Advisor:

Chin-Hwa Lee

Approved for public release; distribution is unlimited.

93 6 01 062

93-12354



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED		1b. RESTRICTIVE MARKINGS	
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE		5. MONITORING ORGANIZATION REPORT NUMBER(S)	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)		7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School	
6a. NAME OF PERFORMING ORGANIZATION Computer Science Dept. Naval Postgraduate School	6b. OFFICE SYMBOL (if applicable) CS	7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER		
8a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (if applicable)	10. SOURCE OF FUNDING NUMBERS	
8c. ADDRESS (City, State, and ZIP Code)	PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO. WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) An Implementation of Traffic Monitoring for UNIX Network Performance Management			
12. PERSONAL AUTHOR(S) Fu, Chen-Hua			
13a. TYPE OF REPORT Master's Thesis	13b. TIME COVERED FROM 01/91 TO 03/93	14. DATE OF REPORT (Year, Month, Day) March 1993	15. PAGE COUNT 312
16. SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.			
17. COSATI CODES		18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)	
FIELD	GROUP	Network performance, Network traffic monitoring, Network traffic status, Network profile, Network configuration, File transfer simulation, PING	
19. ABSTRACT (Continue on reverse if necessary and identify by block number)			
Efficient performance and high throughput are the major goals of the network performance management. How can we achieve these goal? First, it is necessary to know the network traffic situations. This thesis research implements a network traffic query utility for users to monitor the network traffic situations. There are several network traffic situation reports available for users to understand the traffic situation over the network. The network users also can query the network/system status of their respective computer hosts. This information would help users to diagnose the network problems. Realizing the network traffic situation, manager and users can schedule the network applications, re-configure the network configuration, or reallocate the network resources to improve the network performance and throughput. The Naval Postgraduate School campus network will be used as an example to demonstrate and illustrate the usage of this network traffic query utility.			
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS		21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL Chin-Hwa Lee		22b. TELEPHONE (Include Area Code) (408) 656-2190	22c. OFFICE SYMBOL EC/Le

Approved for public release; distribution is unlimited

***An Implementation of
Traffic Monitoring for
UNIX Network Performance Management***

by
Fu Chen-Hua
Captain, R.O.C. Army
B.S., National Defense Management College, 1977

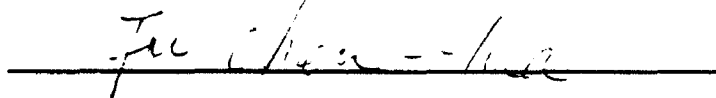
Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF COMPUTER SCIENCE

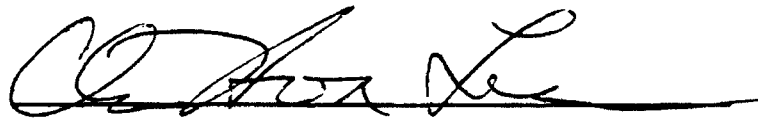
from the

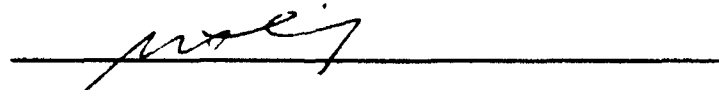
NAVAL POSTGRADUATE SCHOOL
March 1993

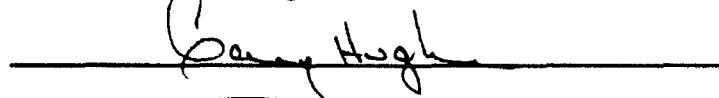
Author:


Fu Chen-Hua

Approved By:


Chin-Hwa Lee , Thesis Advisor


Man-Tak Shing, Thesis Co-advisor


Gary J Hughes, Chairman,

ABSTRACT

Efficient performance and high throughput are the major goals of the network performance management. How can we achieve these goal? First, it is necessary to know the network traffic situations. This thesis research implements a network traffic query utility for users to monitor the network traffic situations. There are several network traffic situation reports available for users to understand the traffic situation over the network. The network users also can query the network/system status of their respective computer hosts. This information would help users to diagnose the network problems. Realizing the network traffic situation, manager and users can schedule the network applications, reconfigure the network configuration, or reallocate the network resources to improve the network performance and throughput. The Naval Postgraduate School campus network will be used as an example to demonstrate and illustrate the usage of this network traffic query utility.

DTIC QUALITY INSPECTED

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or
A-1	Special

TABLE OF CONTENTS

I.)	INTRODUCTION	1
A.)	BACKGROUND.....	1
B.)	OBJECTIVES	1
C.)	OVERVIEW OF THE NETWORK TRAFFIC MONITORING IMPLEMENTATION.....	2
D.)	ORGANIZATION.....	2
II.)	NETWORK TRAFFIC MONITORING AND PERFORMANCE MANAGEMENT: SURVEY	4
A.)	OVERVIEW	4
B.)	NETWORK TRAFFIC MONITORING.....	4
C.)	NETWORK PERFORMANCE MANAGEMENT.....	5
D.)	NETWORK PERFORMANCE MANAGEMENT SURVEY.....	6
1.)	Name and Domain Information	6
2.)	Network Node Traffic Status	7
3.)	Network Statistics Information	8
E.)	SUMMARY.....	8
III.)	EDVOLPEMENT OF A NETWORK TRAFFIC MONITORING AND PERFORMANCE MEASUREMENT PROGRAM.....	10
A.)	SCOPE.....	10
B.)	DESIGN CONCEPT	10
C.)	NETWORK TRAFFIC MEASUREMENT UTILITY.....	11
1.)	Overview	11
2.)	Individual Host Network Status Query	11
a.)	Host Name and Address Query	11
b.)	Internet Domain Name Server Query.....	12
c.)	ECHO Data Packet: Ping Utility	13
d.)	Show Network Status Utility.....	13
e.)	I/O Statistics Report Utility	14
f.)	Virtual Memory Statistics Report Utility	15
g.)	UNIX command Utility.....	15
h.)	File Transfer Measurement Tools	16
3.)	Network Profile Traffic Status Query	21
a.)	Reachability Test.....	22
b.)	Traffic Statistics.....	22
c.)	Network Node Traffic Status Rating.....	28
4.)	Network Profile Maintenance	34
D.)	SUMMARY.....	36

IV.) NETWORK TRAFFIC MONITORING AND PERFORMANCE MANAGEMENT: EXAMPLE AND ANALYSIS	39
A.) INTRODUCTION	39
B.) ANALYSIS - NPS CAMPUS NETWORK TRAFFIC MONITORING EXAMPLE	39
C.) SUMMARY	67
V.) CONCLUSIONS AND RECOMMENDATIONS	69
A.) CONCLUSIONS	69
B.) SUGGESTION FOR FURTHER RESEARCH	69
1.) Improvement To User Interfaces	70
2.) Portability Improvement	70
3.) Developing a Scheduling/Dispatching Algorithm for Network Performance Management	70
APPENDIX A - A DIALOGUE OF NPS CAMPUS NETWORK TRAFFIC MONITORING	71
A.) NETWORK PROFILE SETUP	71
B.) PROFILE TRAFFIC STATUS	76
C.) INDIVIDUAL NETWORK NODE TRAFFIC STATUS MEASUREMENT	103
D.) SUMMARY	116
APPENDIX B - I/O AND VIRTUAL MEMORY STATISTICS REPORTS' FIELDS MAPPING TABLE	117
APPENDIX C - SOURCE CODE OF NETWORK TRAFFIC MONITORING MAIN PROGRAM	119
APPENDIX D - SOURCE CODE OF FILE TRANSFER SIMULATION SERVER PROGRAM USING UNIX DOMAIN STREAM PROTOCOL (TCP)	288
APPENDIX E - SOURCE CODE OF FILE TRANSFER SIMULATION SERVER PROGRAM USING UNIX DOMAIN DARTAGRAM PROTOCOL (UDP)	295
LIST OF REFERENCE	303
INITIAL DISTRIBUTION LIST	304

I. INTRODUCTION

A. BACKGROUND

Computer network is an important branch of the Computer Science. It provides an environment for the users to share the computer resources. It enhances the availability and utilization of the computers. Users can exchange mails, transfer files, perform remote execution, login from a remote terminal, etc.. Network performance is a primary factor that concerns the users. High performance is desirable for better productivity since many distributed tasks share the network. What is the performance measures of the computer network? Performance of a network can be measured by three main indicators: availability, response time, and accuracy. The measurement of response times on a network is dictated by the performance of each node and the traffic of each link.

The global TCP/IP Internet interconnects millions of network nodes at many institutions over the world. It offers the environment for significant collaboration through network services and electronic information exchange. Its functionality depends on reasonable behaviour of every network end-users, smooth operations of the hosts and routers in the Internet. It is highly useful to observe the network status and statistical data on the remote hosts from a local computer host. According to this useful information, the network end-users and computer system administrators can make decisions for proper operations. Therefore, the performance and productivity of the network will be enhanced.

B. OBJECTIVE

This thesis research attempts to develop a program that can monitor and measure the traffic load of a network either locally or remotely. The results of this research can be used as a tool for network managers or system administrators to monitor the network performance. They can use the traffic status of the network to adjust workload through scheduling or routing to improve the performance of the network. The status is also

available to the network end-users. The network end-users can watch the traffic load and decide when and how to use the network for their applications. Selecting a proper time when the traffic is low can avoid the network traffic jam. Choosing a proper network configuration or network profile would help a better distribution of the work load.

C. OVERVIEW OF THE NETWORK TRAFFIC MONITORING IMPLEMENTATION

This program is written in the C language. It uses the socket data structure provided by the Berkeley Unix system as the interface between this network application program and the network communication protocol software [Ref 5]. The TCP and UDP protocols are available for users to measure the network traffic status in the Internet network environment. This application program provides tools for the network manager or system administrator to query useful information about the specified network hosts. It also allows them to test the reachability of different network nodes and to know the immediate traffic status in a star topology. The manager sits at the center, and the links to those nodes specified in the network configuration file are observed. This is a natural preprocessing before dispatching distributed tasks over a network, and it is the baseline for network performance management. The program also provides the network configuration profile query, and it reports the traffic status of the current hot spots.

D. ORGANIZATION

This thesis is organized as follows. Chapter II provides a survey of the network performance measurements. We will discuss the characteristics of the computer network performance and network management. Two important issues discussed in Chapter II are: (1) how to obtain and (2) how to use the observed information to manage and optimize the network performance. The development of the network performance monitoring program will be introduced in Chapter III. In that chapter, we will specify the architecture of the network performance measurement and its methodology. Testing data are collected and

analyzed in Chapter IV. Chapter V gives conclusions and recommendations of possible further works on network performance management.

II. NETWORK TRAFFIC MONITORING AND PERFORMANCE MANAGEMENT: A SURVEY

A. OVERVIEW

The use of computer networks have an explosive growth since 1980s. There are many universities, companies, government and military sites communicate and exchange information over the connected Internet [Ref1]. The Internet becomes worldwide network for the computer users. The traffic status of the Internet is of great concerns to the users.

Why is the network traffic status great concerns to the network users? Performance is an important issue. High volume network traffic causes a slow response time to network applications. Knowing the network traffic of Internet, the users can select links of lighter traffic load for those applications that require efficient and reliable communications. If the network traffic status can be monitored, the network would be used more effectively and efficiently.

How can one achieve the goals of the network performance management? There are several methods to improve the performance of the network. One naive approach is to pay for the incremental costs of hardware/software capacity and capability as situation arises. Alternatively, one can monitor the network traffic. By avoiding the network traffic jam and using the network idled resources, users can effectively improve the performance.

B. NETWORK TRAFFIC MONITORING

The network traffic monitoring is a fundamental aspect of the network management. Two types of traffic monitoring are possible: error detection and baseline monitoring. Network errors should be detected and investigated. In practice communication errors must be logged over time. Furthermore, logging error rates as a function of network traffic rate can be used to show the congestion effects. Daily traffic monitoring may reveal the

operational baseline of network nodes. The baseline determination and traffic monitoring are the keys to early fault detection [Ref 2].

The preliminary step of the baseline measurements is to measure the utilization of the network. Under the utilization measurement, the network problems are most likely to arise, and the network tuning efforts are most likely to be beneficial.

It is worth to develop a source/destination traffic matrix. There, a breakdown of traffic between the local host and other Internet sites are shown. The volume, type and time of the network traffic should be logged. The sudden increases in the variance of delay or the volume of routing traffic should be of concern. The variance in utilization and delay should be tracked as well. They may indicate some problems occurred in the network.

C. NETWORK PERFORMANCE MANAGEMENT

Performance management encompasses two different activities [Ref 2]. One is passive network host monitoring to detect problems and determine operational baselines. The goals are to measure network host utilization and locate the bottlenecks, since bottlenecks should be the focus of the network manager for the network performance tuning. And the performance report is usually required by the upper level management to justify the costs/benefits of communication systems.

Another aspect of performance management is related to active performance testing and capacity planning. Although there are many complex analytic models of network communication available, they do not usually provide correct information on the network performance. Most analytic models of computer network are based on some assumptions on the network traffic load distributions and service rate. Hence, for realistic situations one have to actually measure the network performance which is often costly.

The network performance is usually influenced by the load characteristics of specific applications. Simulation is a good method for network traffic measurement. The traffic simulator generates the actual traffic and trace this traffic over the network. Two guidelines are used for the capacity planning [Ref 2]. A rough estimate of the network host or gateway

capacity is used. A 50% network utilization is assumed so that one can avoid excessive queuing delays.

D. NETWORK PERFORMANCE MANAGEMENT SURVEY

Network performance management should provide timely information of network node names and domain information, network nodes traffic status, and network statistics information. These information will help the network end-users and the network manager to use the network effectively.

1. Name and Domain Information

The network application generally needs an Internet address to open a connection or send a datagram. However users prefer to use the symbolic name rather than the IP address. Name servers on the network should provide the translation tables for mapping between IP addresses and symbolic names.

The domain information is not limited to finding out Internet addresses. Each domain name should be an item in a database. The item can keep records which define a number of different properties [Ref 3], such as: Internet address, computer type, and a list of services provided by a network host. The user can ask for the specific piece of information, or information of a given network host name or alias.

The Internet defines the operation of the name of domain servers and the protocols used to make queries of them. Basically, the name and domain query are the official way to retrieve and evaluate the network host name. These can be done by the network utilities via the name servers and domain servers.

Through the name/domain query, the network users can easily get the network node information. Otherwise, they may go through many network sites to get the node information. These will increase the total load of the network and waste the users' time. From the performance viewpoint, the availability of proper name/domain information can release the traffic load of Internet and indirectly enhance the performance of the Internet.

For the network users, the name/domain information query can provide an useful information and saving of time.

2. Network Node Traffic Status

Network traffic status can report the traffic congestion of the network. The traffic depends on the capacity of the network and the applications invoked by users. If there are many network applications exceeding the capacity of the network, the network congestion will occur. Otherwise, the network traffic should be smooth if there are only a few network applications existing.

How can we measure the network traffic over the Internet? The basic approach is to do network traffic simulation. A special packet will be sent by the traffic generator over the network. The round trip time of the packet to a specific network node of interests will be measured. This round trip time can indicate the network traffic status.

Two types of network traffic monitoring are the real time traffic query and the long term traffic trend analysis [Ref 4]. The real time traffic query provides the facilities for users to measure and understand the current traffic situations of the hosts specified by users. The long term traffic measurement will collect the traffic status of the network hosts for a duration. The long term traffic analysis allows users to find tendency and bottlenecks of the network traffic status. This information is available to the network manager as a network performance reference. The network manager can tune the network according to the information provided by the long term traffic status measurement.

There are two methods to measure the traffic load: specific network host query and network configuration profile query. The specific network host query reports a single specific traffic load of a link. Alternatively, the users may concern some hot sites in the Internet. They can put the symbolic names or IP addresses of the hot sites into the network configuration profile. The network configuration profile query is available for users to measure the network status between the management station and several hot sites.

3. Network Statistics Information

The performance of a computer host is important for the network user. It will directly impact the network performance. The network statistics information of users' hosts should be offered to the users and network managers. It will include two parts: the network status and the system status of host. The network status will display the contents of various network-related data structures and information. It includes the four major information: (1). a list of the activities for each network protocol, (2). network data structure, (3). network routing table, and (4). cumulative traffic statistics. Through these network status information, users and network managers can reveal the current network situation in their hosts.

The computer host system status will let users understand the current status of their computer host. It provides the following information for user and system manager: 1. a report about processes, virtual memory, disk, trap, and CPU activity, and 2. terminal, disk I/O activity, and CPU utilization.

These information will let the system managers diagnose the computer host and do some adjustments to get better performance. The users can also get the current system/network status of the computer host, and they may choose some actions to enhance their productivity over the network.

E. SUMMARY

There is a close relationship between the network traffic monitoring and the network performance management. The network traffic monitoring reveals the congestion status of the network, and it is useful to the users and network managers. Users may avoid the network traffic congestion period when running applications over the network. Consequentially, they can achieve a better productivity with a shorter response time. It also enhances the total throughput on the network by distributing the network applications load. The network managers can locate the network bottlenecks, tune the network, and reallocate resources in configuration changes.

When tuning a network, there are two rules of thumb [Ref 2]. First, there is the principle of locality: a system will perform better if most traffic is between nearby destinations. For instance, local networks use bridges to restrict local communications within clusters. The second rule is to avoid creating the bottleneck over the network. In other words, resources must be carefully allocated. One has to strive for the global optimality whenever possible. These rules of thumb will help us to improve the network performance.

Note that monitoring traffic is not without costs; measurement itself is an overhead of the network load. There is a trade-off between accurate traffic status monitoring and the network traffic load. Too much network traffic monitoring degrades the network performance. A good performance management only monitors the necessary and sufficient network traffic status.

III. DEVELOPMENT OF A NETWORK TRAFFIC MONITORING AND PERFORMANCE MEASUREMENT PROGRAM

A. SCOPE

The program developed in this thesis will operate under the UNIX operating system which provides the socket data facility. This program will monitor and measure the traffic of network nodes selected by users. It can be run at any node on the Internet. The program uses several popular UNIX commands such as 'ping', 'nslookup', 'netstat', 'iostat', and 'vmstat'. These commands can be used to collect useful network information for users and system/network managers.

B. DESIGN CONCEPT

The network traffic monitoring and performance measurement program will setup the facility for users, network managers and system managers, and obtain various network traffic status and traffic statistics. Basically, this program will provide an interactive environment for users to specify the scenarios of monitoring of their interests. The batch mode also is available for some query functions. Similar to the concept of SNMP (Simple Network Management Protocol) used in the TCP/IP environment, the program described in this chapter can be installed at any host that acts as a NMS (Network Management Station). In other words, this program is operated in a logical 'STAR' topology where the monitoring program is polling managed nodes. The host computer where a user is sitting is the center node. The users can directly monitor or measure the traffic status of specific network nodes.

This program provides the network profile query facility and locates the hot spots in a network in real-time mode or in long-term monitoring mode. The users may learn the traffic situations of the network nodes of interests. Besides, this program can also provide the capability for maintaining the network profile. With the monitoring results of a given

network, users can adjust resource allocation, tune the network applications, or reconfigure system components for better performance. Therefore, this program supports network management in several aspects: fault management (by using 'ping'), configuration management (by maintaining the profile), performance management (by monitoring communication quality and response time.... etc.).

C. NETWORK TRAFFIC MEASUREMENT UTILITY

1. Overview

The network traffic measurement utility can be divided into three parts:

(a). individual host network status query, (b). network profile traffic status query, and (c). network profile maintenance.

The first two functions query the network information and traffic status. Through the queries one may learn important indicators of a network: node status and traffic status. The third function maintains the network profiles and acts as a network configuration management tool. Figure 3.1 displays the architecture of the monitoring system. Each part of the network traffic measurement utility is discussed in the following sections.

2. Individual Host Network Status Query

The individual host network status query operation provides eight query functions and utilities for users to obtain the network and host system information. These query functions are described below.

a. Host Name and Address Query

This query function gives the 'name information' of a specific network node that users are interested in. The users need to input the name or address of a network node. The "name" of the network node can then be obtained. The "name" information includes official host name, alias listing, address type, address length, and Internet Protocol(IP) address.

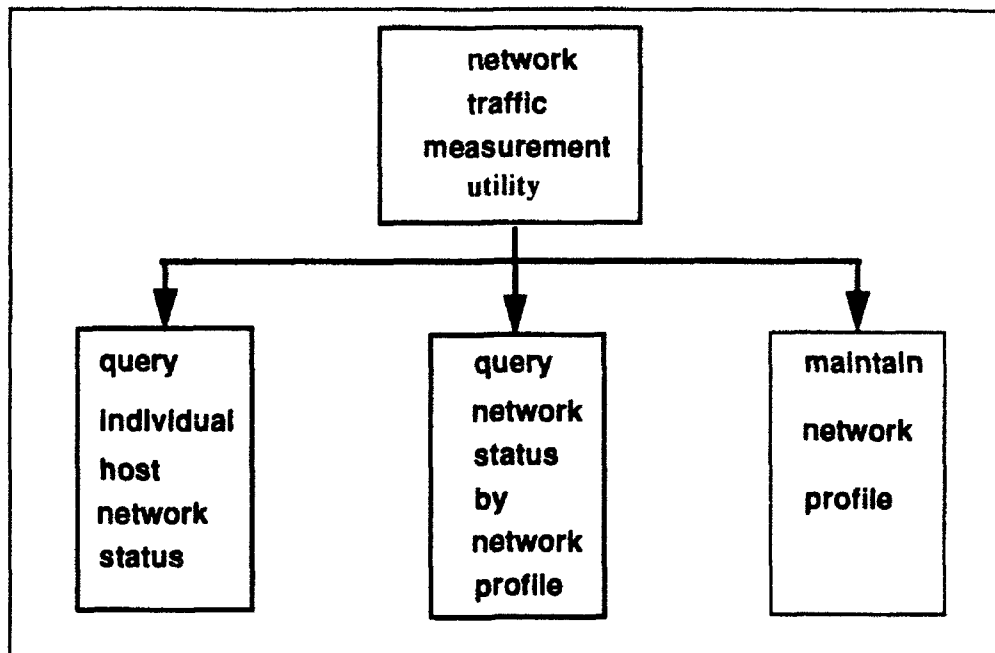


Figure 3.1 Network Traffic Measurement Utility System Architecture

b. Internet Domain Name Server Query

The 'Internet domain name server query' function uses the 'nslookup' UNIX command to query domain name servers interactively. It contacts the name servers, request information about a specific network node, and print a list of network nodes in a specific domain. In this query the users can select the server, query type, and other domain name server query settings. Some useful information can be printed for user's further investigation. Additionally, users can get the detailed information of network nodes from the domain name server. This useful information saves users' time in searching a specific network node, hence it releases the traffic load on the Internet.

The network domain and name server function extends users' ability to reach network nodes over the Internet. This would help users to get the network information and enhance the users' productivity over the network. It also reduces trying errors and the traffic load produced by users on the Internet.

c. ECHO Data Packet: Ping Utility

The 'ECHO data packet-ping utility' function provides two major network information: reachability test and round-trip time measures. The round trip time includes the time needed from the users' host to the specified network node and then echo back from that specified node. This information will display the network traffic status between these two network nodes. This function utilizes the ICMP protocol's mandatory ECHO_REQUEST datagram to elicit an ICMP ECHO_RESPONSE from the specified network node or network gateway. The ECHO_REQUEST datagram has an IP and ICMP header, followed by a 'struct timeval', and an arbitrary number of bytes in padded packet. If the specific node echoes back, it indicates the reachability of that node. Other options by the users when sending ECHO_REQUEST to target nodes, are routing types, verbosity, and datagram sizing. When echoing, the target node sends back ECHO_RESPONSE. These ECHO_REQUEST and ECHO_RESPONSE messages indicate the node-to-node network traffic status. They are helpful for users to know the traffic status from users' host node to peer nodes of concern.

There are two execution modes for users to select: interactive and batch. The users can directly input the ping command and its parameters in interactive mode. If users are familiar with the syntax of the 'ping' command, they can select the batch mode to execute the ping utility and get a faster response.

d. Show Network Status Utility

This function displays the contents of various network-related data structures in various formats depending on the options selected. There are three types of network status information: Active Sockets, Network Data Structures, and Cumulative Traffic Statistics.

Active Sockets format displays a list of active sockets for each protocol. It shows the local and remote address, the 'send' and 'receive' queue sizes (in bytes), the protocol, and the internal state of the protocol. There are several Network Data Structures

reports available for users to select: 1. the statistics of the network's private buffer pool, 2. the routing table, 3. the state of interfaces that have been auto-configured. Cumulative Traffic Statistics format displays the statistics of packets transmitted on the network interfaces. When the time interval argument is set by users, the network status utility according to packets transferred displays a table of cumulative statistics on, errors and collisions, the network addresses for the interface, and the maximum transmission unit. The cumulative statistics format displays the network traffic statistics information since the system was last rebooted. The first line summarizes the cumulative network traffic statistics information, and every 24th line thereafter, from the time the system was last rebooted. Each subsequent line shows the network traffic statistics information for the interval specified by users since the previous line displayed.

As mentioned earlier, users may use the utility either interactively or in batch mode. When running interactively, the users can select the parameters of the network status utility and get the network status report they wanted. When the users are familiar with the 'netstat' command, they can execute this utility in the batch mode.

e. I/O Statistics Report Utility

The 'I/O statistics report utility' function reports terminal and disk I/O activity, as well as CPU utilization. The collected information includes cumulative statistic since the latest reboot. If users set the 'interval' parameter for 'I/O statistics report utility', each subsequent report displays the I/O activities during the prior interval only.

This function utilizes the 'iostat' UNIX command collecting I/O statistics from the kernel. The kernel maintains several counters. Each disk seeks, data transfer completions, and the number of words transferred are maintained by these counters. For terminals collectively, the number of input and output characters are kept. Besides, at each clock tick, the state of each disk is examined, and a tally is made if the disk is active. The kernel also provides approximate transfer rates of the devices. This utility allows users to run it interactively or in batch mode. Information about the I/O activities can be used to

detect abnormal conditions that may degrade the network traffic performance. One can fix these abnormal conditions to improve the network performance.

f. Virtual Memory Statistics Report Utility

The 'virtual memory statistics report utility' function uses the 'vmstat' UNIX command to report the process, virtual memory, disk, trap and CPU activities. If users do not specify any parameter for this utility, this utility displays a one-line summary parameter of the virtual memory activity since the system has been booted. If users set the time interval parameter, the 'vmstat' will collect information about virtual memory during this interval and list the summary. If the count parameter is given, the statistics listings are repeated that many times.

There are other system information available for users. This utility can report the number of processes in each of the three following states: (1). in 'ready queue', (2). blocked for resources and (3). runnable or short sleeper (< 20 secs) but swapped. It also reports the usage of virtual and real memory, the information about page faults, paging, and activity. The number of disk operations per second and the trap/interrupt rate averages per second over last 5 seconds are displayed in this utility report. The break down of percentage usage of CPU time also is included in this virtual memory statistics report. Again, both the batch execution mode and interactive execution mode are available. To run this utility in batch mode one must know how to use the 'vmstat' command.

g. UNIX Command Utility

The 'UNIX command utility' function provides the interface for users to execute the UNIX commands within this 'network traffic monitoring and performance measurement' program without leaving the utility. The users may use the UNIX commands or UNIX shell to execute the functions.

h. File Transfer Measurement Tools

The 'file transfer measurement tools' measures the file transfer time from the user's host to a specified network node. It stores the file transfer time and other information in a specific 'collection file' or 'statistic file'. It can generate the file transfer statistics for users to analyze the network traffic between two network nodes. The file maintenance utility is available for maintaining the data files. This function can be divided into three subfunctions: file transfer measurement, measurement data statistics report, and data file maintenance utility.

The 'file transfer measurement' subfunction utilizes the file transfer to simulate the network traffic between two network nodes. This subfunction simulates an client/server architecture over the Internet. One requirement to run this option is that both the server and client must support the UNIX socket system calls. The users can select the protocols, packet size for each data transfer, file size for data transmission, and measurement times. The file transfer simulator uses the socket system calls to establish the file transfer environment and transfers file from client to server. The file transfer option supports two protocols: TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). The TCP protocol is a reliable connection_oriented data protocol which is characterized by the need for reliable, sequenced delivery of data. The UDP protocol is an less reliable connectionless data protocol which is a low-overhead, minimum functionality

protocol. Figure 3.2 and Figure 3.3 show how the socket system calls work for the connection-oriented protocol(TCP) and connectionless protocol(UDP):

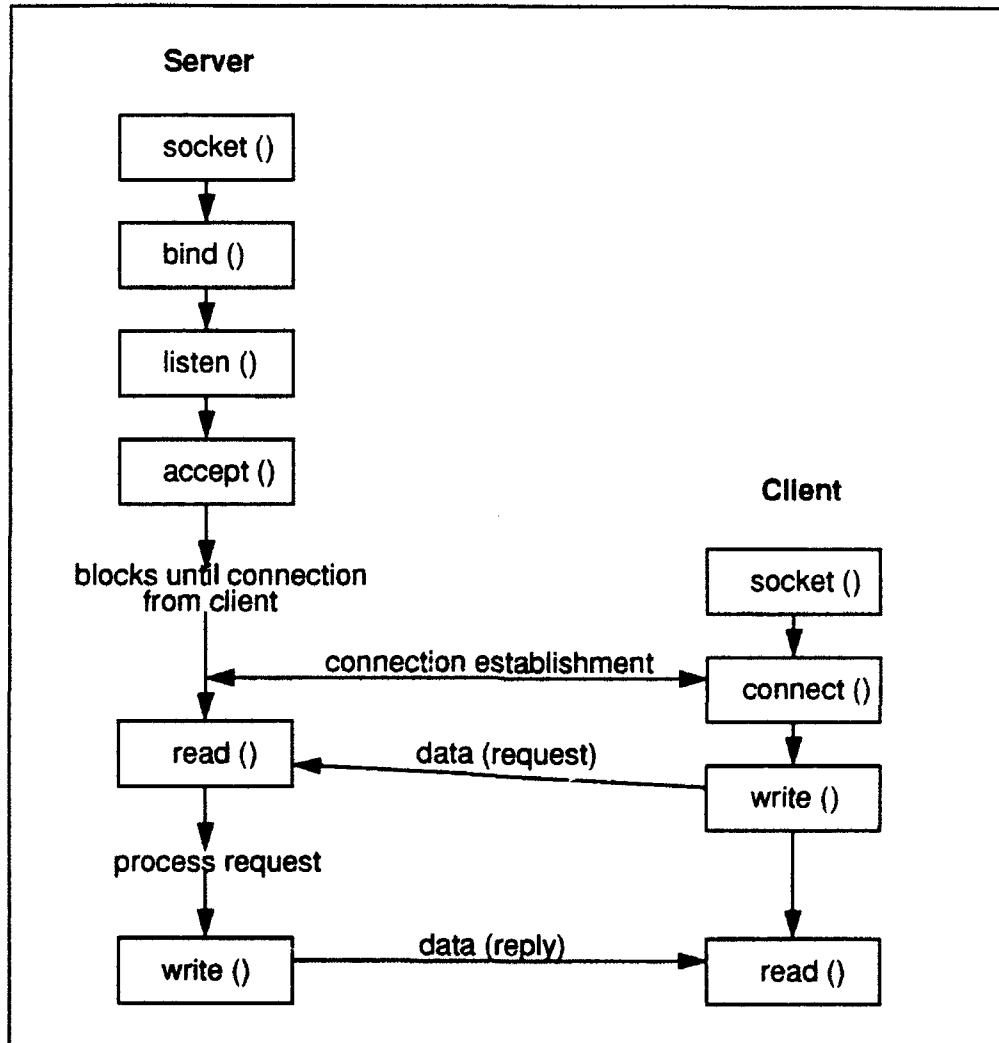


Figure 3.2 Socket System Calls for Connection-oriented Protocol (TCP)

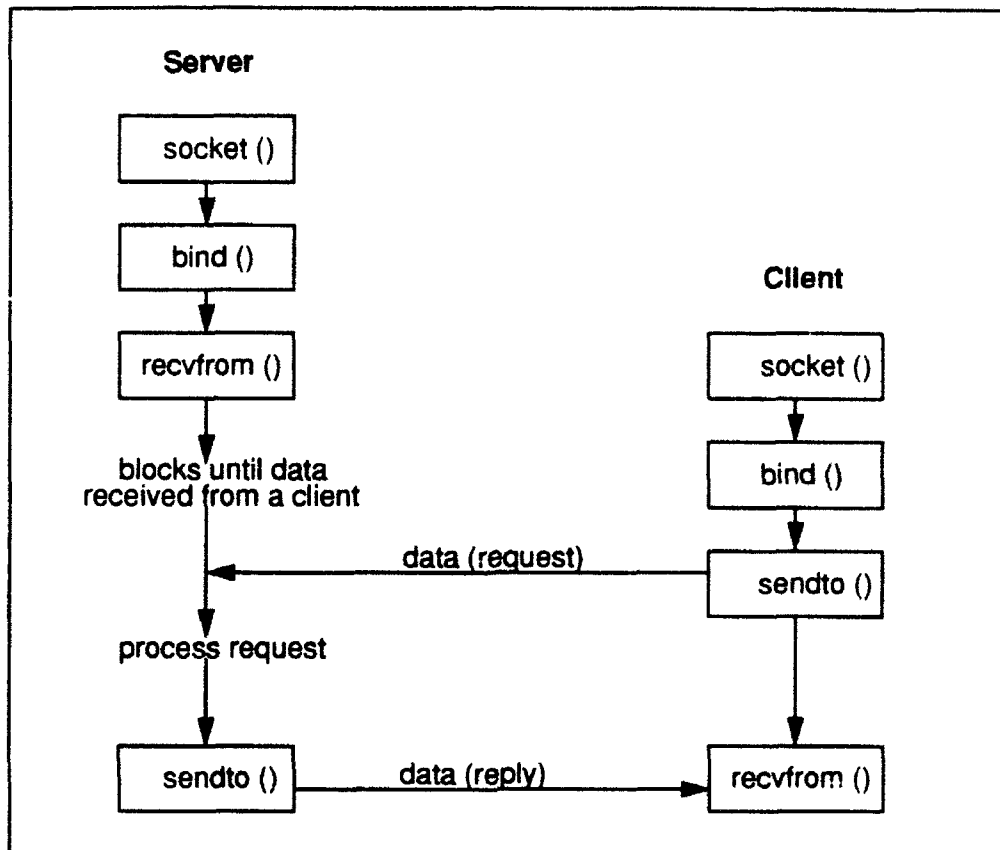


Figure 3.3 Socket System Calls for Connectionless Protocol (UDP)

The file transfer simulator follows the architectures described above and transfer data from client to server. It samples the file transfer time. Before the simulation starts, users must start the server, e.g. execute the 'receive' command, so that the client can access later. Then users can activate the 'send' data command at the client side. When the connection is established between the client and server, the file transfer is then started.

We now explain the mechanism of file transfer. The server sets the beginning timestamp when it receives the first data(request) from the client. The server sets the ending timestamp when it receives the last piece of data(request). These timestamp provide the total time needed for the file transfer. This time includes the disk I/O activities. The

program subtracts the disk I/O time of the file writing from the time of file transfer with disk I/O. One now obtains the time of file transfer without disk I/O. Beside the time of file transfer with disk I/O and time of file transfer without disk I/O, the server program will collect the file length received and the first timestamp of the file transfer. One file transfer time data sample contains file length, beginning timestamp of the file transfer, the time of file transfer with disk I/O and the time of file transfer without disk I/O. The server program will send the file transfer time data sample to the client when it finishes receiving the file sent by the client.

The client receives the file transfer time sample data and store it in a data file. This file will be used to generate the file transfer statistics report. The file transfer simulator will repeat the file transfer according to the measurement times. The number of sample of the data file is the measurement of the times of file transfer simulation.

File transfer includes two processes: the data transportation over the network, and the file read/write with the disk I/O. So the process of file transfer should include the two different process time. One is the time for transferring file through the network. The other is the time used to read/write disk. For example: The time of file transmission through the network is 20 seconds. The time of disk I/O is 0.5 seconds. The total file transfer time is 20.5 seconds.

If the file transfer time does not include the disk I/O time, one would need only 20 seconds in the above example. The file transfers without disk I/O time is available for users, too. This time is the time of file transmission through network. The file transfer time with disk I/O and the time without disk I/O let user know the relationship between the network performance and the disk file system. This is the primary reason to distinguish the file transfer time with disk I/O from the file transfer time without disk I/O.

The 'file transfer statistics reports' subfunction generates the statistics for users to analyze the network traffic situation between the client site and a server. This subfunction utilizes a collection file collected by the 'file transfer measurement' subfunction. First, the users need to specify which node they want to analyze. This

subfunction will check the node and its statistics files. If they are available, it will generate the statistics. When running interactively, one must enter parameters, such as the name of information file. The naming of the file transfer time information follows the format of 'protocol_buffersize.filesize'.

The users also need to set the sampling frequency and the sample size. The time interval of the file transfer let users specify a time period during which the program collects the statistics. Different sample number and time interval generate different statistics reports. They provide some useful file transfer information and network traffic situations for users to analyze the network performance.

The file transfer statistics report shows the basic file transfer information to users: the server address, file name, file length, number of samples, and the time period. It also shows starting time and the ending time of the file transfer, the distributions of the file transfer simulation in the time periods, and the measurement time. At last, the maximum, minimum, mean, variance, and standard deviation of file transfer time are all calculated and shown.

The 'data file maintenance utility' subfunction setups two kinds of files used in the 'file transfer measurement tools' function. The first type of files are the data files used in the file transfer testing. These data files can be of different lengths and can be maintained on demand. Another type of files are the 'file transfer simulation result' (FTSR) files, or resulting files in short. A FTSR file is generated by the file transfer simulator which logs the file transfer time. The file transfer simulator will make different server subdirectories according to the IP address of servers. These FTSR files will be stored in the corresponding subdirectories depending on which server the file transfer simulator measures. If the FTSR files of one server are no longer needed, the users can delete them by entering the symbolic name or IP address.

In general, the 'file transfer measurement tools' function allows a user to monitor the network status between a client and a server. It provides the real-time or precise network traffic information during the simulation. The measurements include the file

transfer time, its distributions, and the variance of the file transfer. Note that the file transfer simulation itself is a workload to the network and will increase the network congestion. Therefore, to relieve the traffic load this function cannot be used too often. Instead, one should use the simplified 'ECHO data packet' or 'ping' if only brief overhead network traffic is of concern.

3. Network Profile Traffic Status Query

A network profile is a text file representing the network's configuration. Each record in the file represents a node and contains either the symbolic name or the IP address of that node. The 'network profile traffic status query' function can be invoked in batch mode using the network profiles specified in the shell. The executing host measures the network traffic status originating from it to all the nodes specified in the profile. In other words, any installed node can act like a network management station. For distributed systems, one can install it in several nodes and remotely invoke them. The main function of network profile is to support the configuration management in a network. According to the configuration specified in the profile, this query provides the users information gathered from the UNIX command 'ping'. Note that the 'ping' command uses the ICMP protocol [Ref. 8]. The ICMP protocol's mandatory ECHO_REQUEST datagram elicits an ICMP ECHO_RESPONSE from the network node or network gateway specified in the network profile. It brings the information about the IP header, the ICMP header, datagram round trip time, and the length of data which are padded to the ICMP datagram. The information collected by the 'ping' command is then processed and summarized by the 'network profile traffic status query' function in reporting reachability and performance rating of the network nodes specified in the network profile.

There are three primary traffic status query utilities in the 'network profile traffic status query' utility: (1). reachability test, (2). traffic statistics and (3). network node traffic status rating. These three primary query utilities are now discussed as follows.

a. Reachability Test

The 'reachability test' function uses the symbolic name or IP address of each network node specified in the network profile to test whether a connection to that node is possible. The returned messages show that the specified network node is alive if the host receives the ECHO_RESPONSE datagram. Otherwise, there is no reachability between the host and that specified node. One can select specific network profile according to the network configuration of one's concern. The 'reachability test' is usually run in interactive mode. The reachability of a network node provides the basic network status to users. Any other application must start with the reachability test.

b. Traffic Statistics

The 'traffic statistics' function collects the traffic status and generates the traffic statistics reports from the nodes specified in the network profile. It can generate two reports: real-time network traffic statistics and the long-term network traffic statistics. The real-time option runs interactively according to the network profile supplied by the users. Having confirmed the network profile, users must specify the length of data which will be padded to the ICMP datagram and the count number that will be used to generate the network traffic status. The meaning of these parameters will become clear in the subsequent discussions. However, there is a trade off between the datagram length and the count number. If users want to get more precise network traffic status, they can select the larger datagram size and larger count number. But this combination will cause the network to spend more time in measurements and to increase the traffic load. For a rough estimate, users can select a smaller datagram size and count number. Users can also list the network traffic messages generated by the 'ping' command. The 'real-time network traffic query' starts its execution when all the parameters are given. A record in the traffic statistics report contains seven fields: 'transmitted packets number', 'received packets number', 'received packets percent', 'minimum round-trip time', 'average round-trip time', 'maximum round-

trip time' and 'network node'. An example of the real-time network profile traffic statistics report is shown in Figure 3.4.

*** Traffic Statistics Report ***						
transmitted packets number	received packets percent	received packets percent	minimum round-trip number	average round-trip number	maximum round-trip number	network node
5	5	100.00	0	2	10	nps
5	5	100.00	0	0	0	ece
5	5	100.00	0	8	10	oc
5	5	100.00	20	26	30	cc
5	5	100.00	0	2	10	pegasus
5	5	100.00	0	0	0	cs
5	5	100.00	0	2	10	taurus
5	5	100.00	-	-	-	rover
5	5	100.00	0	2	10	csrg
5	5	100.00	10	10	10	131.120.57.2

Figure 3.4 an Example of Real Time Network Profile Traffic Statistics Report

The 'transmitted packets number' specifies the number of ICMP ECHO_REQUEST datagrams transmitted from manager's node to a managed or monitored node. This number is set when the users specify the 'count' number. The 'received packet number' tells users how many ICMP ECHO_RESPONSE packets are received from the managed node. The number of received packets indicates the quality of data transmission: the more packets that get echoed back from the other side one, the better the connection will be. The 'received packets percent' is the ratio of the 'transmitted packets number' and the 'received packets number'. This measures the stability and quality of the network communications. The higher the better. If the ratio is less than 70%, a mark of '*' will be shown. Alerted by the '*' mark, one would pay attention to that specific network node.

The 'minimum round-trip time', 'average round-trip time', and 'maximum round-trip time' will show the range of round-trip time. The round-trip time is the time spent to send the ICMP ECHO_REQUEST datagram to the specified network node and

receive the ICMP ECHO_RESPONSE datagram from that specified node. The 'minimum round-trip time', 'average round-trip time', and 'maximum round-trip time' represent the minimum/average/maximum time of the round-trip time, respectively. These measures provide the basic information on the distribution of round-trip time. If the specified network node cannot be reached, there is no information on minimum/average/maximum round-trip time. These three fields will be marked by '-', telling network managers that there maybe a problem. The 'network node' shows the content of network profile by displaying either the symbolic name or the IP address of the node. This report can be used to adjust the network load by allocating distributed applications to nodes hence to achieve good performance.

The 'long-term network traffic statistics query' subfunction has two parts: the 'long-term network profile traffic monitoring', and the 'long-term network profile traffic statistics report'. The first part monitors the network traffic situation of network nodes and stores the network traffic status into a file. The second part uses the file collected by the first part to generate a long term network profile traffic statistics report. Instead of verbosity, the second part summarizes information collected in the first part.

The 'long-term network profile traffic monitoring' shows the network profile traffic over a period specified by the users. To use this utility, one must specify a network profile name, the duration of monitoring period, and the sampling period. Monitoring period can be minutes, hours, days, months, or years. A sampling period is the time between the consecutive records in the network profile traffic status file (the output of the monitoring results). Having specified the above parameters, one can develop a UNIX shell. This shell will collect the network profile traffic status according to the network profile, total network traffic monitoring period, and the sampling period.

The 'long-term network profile traffic statistics report' reports traffic statistics over the observation period. Before generating this report, users need to select a file for storing the monitoring results. Its name is a combination of the 'network profile', 'total monitoring period', and 'sampling period'. For example: 'nps_1_hour_by_

per_10_minutes', this file name means the nps network profile configuration monitored over a period of one hour and sampled every 10 minutes. This report can be divided into several blocks according to the network node number in the specified network profile. There are six fields in each block: 'network node name', 'time', 'received packets percent', 'minimum round-trip time', 'average round-trip time', and 'maximum round-trip time'. One example of the long-term network profile traffic statistics report is shown in Figure 3.5.

Generally speaking, the 'received packets percent', 'minimum round-trip time', 'average round-trip time', and 'maximum round-trip time' in the long-term traffic statistics report are the same as the four fields in that of the real-time mode. Additionally, there are two other fields in the long-term network profile traffic statistics report: network node name and time. The 'network node name' displays the symbolic name or IP address of each node. The 'time' shows the sampling time of one network profile traffic status record.

The 'long-term network profile traffic statistics report' provides the network profile traffic situation of each specified network node in the measurement duration. During the observation period, users monitor the traffic situation of each network node. With the measurements, one can analyze the network traffic characteristics and tendency of all nodes. The analysis may lead to some manual adjustments on the scheduling of applications and enhancement of the productivity and throughput over the network. The network managers can analyze long-term traffic measurements and do some tuning on the network configuration. This would achieve a better network performance.

***** Long Term Traffic Statistics Report *****				
network node name : nps				
time	percent packets received	time round-trip minimum	time round-trip average	time round-trip maximum
Wed Feb 17 17:00:14 PST 1993	100.00	0	2	9
Wed Feb 17 17:00:14 PST 1993	100.00	0	1	10
Wed Feb 17 17:27:15 PST 1993	100.00	0	0	10
network node name : ece				
time	percent packets received	time round-trip minimum	time round-trip average	time round-trip maximum
Wed Feb 17 17:00:14 PST 1993	100.00	0	2	10
Wed Feb 17 17:13:45 PST 1993	100.00	0	0	10
Wed Feb 17 17:27:15 PST 1993	100.00	0	1	10
network node name : oc				
time	percent packets received	time round-trip minimum	time round-trip average	time round-trip maximum
Wed Feb 17 17:00:14 PST 1993	100.00	0	8	10
Wed Feb 17 17:13:45 PST 1993	100.00	0	11	30
Wed Feb 17 17:27:15 PST 1993	100.00	0	8	10
network node name : cc				
time	percent packets received	time round-trip minimum	time round-trip average	time round-trip maximum
Wed Feb 17 17:00:14 PST 1993	100.00	20	35	110
Wed Feb 17 17:13:45 PST 1993	100.00	20	30	40
Wed Feb 17 17:27:15 PST 1993	100.00	20	37	170
network node name : pegasus				
time	percent packets received	time round-trip minimum	time round-trip average	time round-trip maximum
Wed Feb 17 17:00:14 PST 1993	100.00	0	0	10
Wed Feb 17 17:13:45 PST 1993	100.00	0	0	9
Wed Feb 17 17:27:15 PST 1993	100.00	0	0	10

Figure 3.5 an Example of Long Term Network Profile Traffic Statistics Report

network node name : cs				
time	percent packets received	time round-trip minimum	time round-trip average	time round-trip maximum
Wed Feb 17 17:00:14 PST 1993	100.00	0	0	0
Wed Feb 17 17:13:45 PST 1993	100.00	0	0	9
Wed Feb 17 17:27:15 PST 1993	100.00	0	0	10
network node name : taurus				
time	percent packets received	time round-trip minimum	time round-trip average	time round-trip maximum
Wed Feb 17 17:00:14 PST 1993	100.00	0	0	0
Wed Feb 17 17:13:45 PST 1993	100.00	0	0	10
Wed Feb 17 17:27:15 PST 1993	100.00	0	0	0
network node name : rover				
time	percent packets received	time round-trip minimum	time round-trip average	time round-trip maximum
Wed Feb 17 17:00:14 PST 1993	*0.00	-	-	-
Wed Feb 17 17:13:45 PST 1993	*0.00	-	-	-
Wed Feb 17 17:27:15 PST 1993	*0.00	-	-	-
network node name : csrg				
time	percent packets received	time round-trip minimum	time round-trip average	time round-trip maximum
Wed Feb 17 17:00:14 PST 1993	100.00	0	6	80
Wed Feb 17 17:13:45 PST 1993	100.00	0	0	9
Wed Feb 17 17:27:15 PST 1993	100.00	0	1	10
network node name : 131.120.57.2				
time	percent packets received	time round-trip minimum	time round-trip average	time round-trip maximum
Wed Feb 17 17:00:14 PST 1993	100.00	10	13	30
Wed Feb 17 17:13:45 PST 1993	100.00	10	10	20
Wed Feb 17 17:27:15 PST 1993	100.00	10	10	20

Figure 3.5 continued

c. Network Node Traffic Status Rating

Each communication link may have different traffic status and hence can be rated differently. This rating can help to decide the schedule of network applications over the network. The long-term rating also is useful to the network managers to diagnose the network. If there is one or more network nodes always having the bad rating, then they deserve some attention. This is the responsibility of the network manager. Through the rating information, users and network managers can choose the proper actions to improve the productivity and throughput over the network.

Before discussing how one rates the links between nodes, we must first discuss the characteristics of the network traffic. Links between nodes are generally of different lengths and propagation delays. The time of data packet transmission alone cannot be used as the only criteria to rate the network node traffic status. A good network traffic status means there is a stable data transmission ability between network nodes, and the variance of data transmission time should be small. Hence, the 'received packets percent' and 'round-trip time variance' become the two important criteria to judge the network node traffic status. The 'received packets percent' represents the data packet transmission capability between user's site and the managed node. The 'round-trip time variance' can specify the variance of data transmission round trip time between two network nodes. The data transmission quality is the most important factor to rate the network node. Therefore the 'received packets percent' is used as the primary rating factor. The secondary factor is the 'round-trip time variance'. A smaller 'round-trip time variance' means the data transmission capability is more stable between the two network nodes. The following pseudo code specifies the algorithm of the network node traffic status rating:

```

if (network_node1->received packets percent >
network_node2->received packets percent)
{
network_node1 get the higher network node traffic status rate
}
else
{
if (network_node1->received packets percent =
network_node2->received packets percent)
{
if (network_node1->round-trip time variance <=
network_node2->received packets percent)
{
network1_node get the higher network node traffic status
}
}
}
}

```

The 'network node traffic status rating' function collects, rates, and reports the traffic status. Two types of reports can be generated: the 'real-time network node traffic status rating report' and the 'long-term network node traffic status rating report'. The real-time subfunction allows users to select the network profile and set the ICMP datagram size, and count number. There are eight fields in the 'real time network node traffic status rating report': 'packet size', 'sample number', 'rating', 'received percent', 'round-trip time mean value', 'round-trip time variance', 'round-trip time std-deviation', and 'network node'. One sample of the report example is shown in Figure 3.6.

The 'packet size' specifies the length of data which is padded out in the ICMP datagram packet. Its value is set by users. The 'sample number' tells users how many ICMP datagram packets are used to measure the network traffic for each specified network node. The value of this number might determine the accuracy of the 'real-time network node traffic status rating report'. The larger this number is, the more accurate the result is. But too many samples imply that one must spend much time to measure the network traffic status, hence increase the network burden.

***** Network node traffic status rating report *****					
packet size : 512 data bytes			sample number : 5 times		
rating	received packet percent	round-trip time mean-value	round-trip time varinace	round-trip time std-deviation	network node
1	100.00	0.000	0.000	0.000	pegasus
2	100.00	2.000	20.000	4.472	ece
2	100.00	2.000	20.000	4.472	cs
2	100.00	2.000	20.000	4.472	taurus
2	100.00	2.000	20.000	4.472	csrg
3	100.00	6.000	30.000	5.477	nps
3	100.00	26.000	30.000	5.477	cc
4	100.00	8.000	170.000	13.038	oc
5	100.00	14.000	230.000	15.166	131.120.57.2
6	*0.00	0.000	0.000	0.000	rover

Figure 3.6 an Example of Real Time Network Node Traffic Status Rating report

The 'rating' displays the rate of each specified network node. This field is the most important indicator in the 'real-time network node traffic status rating report'. The 'received packets percent' indicates the data transmission ability. The higher the 'received packet percent' is, the better the data transmission ability between users' network node and the specified network node will be. If the 'received packet percent' is less than 70%, the special mark '*' will be put before the value of the 'received packet percent'. This special mark will warn users to notice the specified node that seems abnormal in the data transmission ability.

The 'round-trip time mean-value', 'round-trip time variance' and 'round-trip time std-deviation' give users the reference information about the network traffic situation between two network nodes. The 'round-trip time variance' will be used to rate the network node traffic situation if two network nodes have the same 'received packets percent' value.

The last field 'network node' displays the symbolic name or IP address of each specified node. Users can identify the network traffic information to each network node. The 'long-term network node traffic status report' will be available for users to realize the long term network node traffic situations and its rating. Before the 'long term network node traffic status rating report' is generated, users need to monitor the long term traffic situations of specified network nodes and rate them. Then 'long term network node traffic status rating report' can be generated. The long term traffic situations monitoring and rating can be accomplished by the 'long term network node rating traffic monitoring' subfunction. The 'long term network node traffic rating statistics report' subfunction will generate the 'long term network node traffic status rating report'.

The 'long-term network node rating traffic monitoring' subfunction would ask users to select the network profile name, input data packet length, and count number for network traffic measurement. Users also need to set the total measurement period and sampling period. According to the parameters setting, a UNIX shell will be generated and automatically executed in the UNIX background. The network node traffic status rating information will be generated and stored into a specific file which is named by users specified parameters.

The 'long-term network node traffic status rating report' subfunction generates the long-term rating. In this subfunction, users need to specify a 'long term network traffic rating information file' or 'long term data file' used in the rating. The file naming is the same as the long-term network traffic statistics information file which has been mentioned before. The format of this report is similar to the 'long term network profile traffic statistics report'. This report can be divided into several blocks. Each network node has one block to report its traffic status rating during the measurement period. There are seven fields to display the network node traffic status information: 'network node name', 'time', 'rating', 'received packets percent', 'round-trip time mean-value', 'round-trip time variance', and 'round- trip time std-deviation'. There is an example of the 'long term network node traffic status rating report' in Figure 3.7.

***** Long Term Network node traffic status rating report *****						
network node name : taurus						
time	rating	received packet percent	round-trip time mean-value	round-trip time variance	round-trip time std-deviation	
Thu Mar 4 10:35:59 PST1993	1	100.00	0.500	5.000	2.236	
Thu Mar 4 10:49:28 PST1993	1	100.00	0.000	0.000	0.000	
Thu Mar 4 11:02:56 PST1993	1	100.00	0.000	0.000	0.000	
network node name : nps						
time	rating	received packet percent	round-trip time mean-value	round-trip time variance	round-trip time std-deviation	
Thu Mar 4 10:35:59 PST1993	2	100.00	1.000	9.474	3.078	
Thu Mar 4 10:49:28 PST1993	3	100.00	1.450	12.576	3.546	
Thu Mar 4 11:02:56 PST1993	5	100.00	1.500	13.421	3.663	
network node name : ece						
time	rating	received packet percent	round-trip time mean-value	round-trip time variance	round-trip time std-deviation	
Thu Mar 4 10:35:59 PST1993	2	100.00	1.000	9.474	3.078	
Thu Mar 4 10:49:28 PST1993	2	100.00	0.950	8.576	2.929	
Thu Mar 4 11:02:56 PST1993	2	100.00	0.450	4.050	2.012	
network node name : pegasus						
time	rating	received packet percent	round-trip time mean-value	round-trip time variance	round-trip time std-deviation	
Thu Mar 4 10:35:59 PST1993	3	100.00	2.000	5.000	4.104	
Thu Mar 4 10:49:28 PST1993	7	100.00	4.000	130.525	11.425	
Thu Mar 4 11:02:56 PST1993	5	100.00	1.500	13.421	3.663	
network node name : 131.120.57.2						
time	rating	received packet percent	round-trip time mean-value	round-trip time variance	round-trip time std-deviation	
Thu Mar 4 10:35:59 PST1993	4	100.00	11.500	23.947	4.894	
Thu Mar 4 10:49:28 PST1993	5	100.00	11.500	23.947	4.894	
Thu Mar 4 11:02:56 PST1993	3	100.00	10.500	5.000	2.236	

Figure 3.7 an Example Long Term Network Node Traffic Status Rating Report

network node name : oc					
time	rating	received packet percent	round-trip time mean-value	round-trip time variance	round-trip time std-deviation
Thu Mar 4 10:35:59 PST1993	5	100.00	4.000	35.789	5.982
Thu Mar 4 10:49:28 PST1993	6	100.00	4.500	26.053	5.104
Thu Mar 4 11:02:56 PST1993	6	100.00	4.000	25.263	5.026
network node name : csrg					
time	rating	received packet percent	round-trip time mean-value	round-trip time variance	round-trip time std-deviation
Thu Mar 4 10:35:59 PST1993	6	100.00	1.950	47.629	6.091
Thu Mar 4 10:49:28 PST1993	5	100.00	1.500	23.947	4.894
Thu Mar 4 11:02:56 PST1993	4	100.00	0.950	8.576	2.929
network node name : cs					
time	rating	received packet percent	round-trip time mean-value	round-trip time variance	round-trip time std-deviation
Thu Mar 4 10:35:59 PST1993	7	100.00	3.500	55.526	7.452
Thu Mar 4 10:49:28 PST1993	4	100.00	1.000	20.000	4.472
Thu Mar 4 11:02:56 PST1993	1	100.00	0.000	0.000	0.000
network node name : cc					
time	rating	received packet percent	round-trip time mean-value	round-trip time variance	round-trip time std-deviation
Thu Mar 4 10:35:59 PST1993	8	100.00	71.000	5104.210	71.444
Thu Mar 4 10:49:28 PST1993	8	100.00	50.000	1431.579	37.836
Thu Mar 4 11:02:56 PST1993	7	100.00	54.950	2850.050	53.386
network node name : ece					
time	rating	received packet percent	round-trip time mean-value	round-trip time variance	round-trip time std-deviation
Thu Mar 4 10:35:59 PST1993	9	*0.00	0.000	0.000	0.000
Thu Mar 4 10:49:28 PST1993	9	*0.00	0.000	0.000	0.000
Thu Mar 4 11:02:56 PST1993	8	*0.00	0.000	0.000	0.000

Figure 3.7 continued

The 'long term network node traffic status rating report' is the accumulation of many 'real time network node traffic status rating report'. These real time reports are reorganized by the 'time' field. There are the following fields in both reports: 'rating', 'received packets percent', 'round-trip time mean value', 'round-trip time variance', 'round-trip time std-deviation'. The 'network node name' field is the same as the 'network node' field in the 'real time network node traffic status rating report'. The 'time' field is the only different field between these two reports. It tells users the sampling time of each network node traffic status rating record.

The 'long term network node traffic status rating report' displays the variance of network node traffic status rating in each network node. It also indicates the network traffic situation of each specified node. This report can be used as a diagnostic tool over the specified network nodes. It can detect the poor performance network nodes which always get the low rating. The network manager can do some adjustments on the network configuration and resources allocation according to this result. This may enhance the total performance and throughput over the specified network nodes.

4. Network Profile Maintenance

The network profile describes network nodes of users' interests in. It is a tool for users to manage network configuration. The network profile is the basis for network status query. A good network profile maintenance facility allow users to maintain the network profiles and adjust the network configuration. The 'network profile maintenance' utility provides the functions for users to maintain their network profiles. This utility has three primitive functions: add, delete and update.

The 'add a network profile' creates a new network profile. In this function, users must specify a new file name that is not currently in use. Since the special character '_' is reserved for use as a delimiter to check the name of network node traffic status file in the 'network traffic measurement utility'. Therefore, '_' cannot be used in a file name. Having specified the new network profile name, users can input either the symbolic names or IP

addresses of network nodes that they want to be included in this network profile. If a symbolic name or an IP address cannot be located in the name server, users will receive a warning message. Users can choose to add it into the new network profile. Users can exit the function by entering 'exit' or 'quit'.

The 'delete a network profile' allows users to delete a network profile. First, users may select the file to be deleted from a list of files that are currently in use. Users have the choice to browse the contents of the network profile to be deleted before the actual action. Two other options in deleting a file are browsing the contents and confirming the deletion request.

The 'update a network profile' provides three subfunctions for users to modify the contents of specified network profile. These three subfunctions are 'add a network node', 'delete a network node', and 'update a network node'. Having selected the specific network profile, users need to select an update subfunction to update the network nodes. Adding a node is done by appending the node to the profile. In this subfunction, users need to provide the symbolic name or IP address of the network node. The symbolic name/ IP address will be checked from the name server. If it does not exist in the named server, there is a warning message for users. Users have the choice to add it into the specified network profile. When deleting a node, the user has the option to browse the content of the profile and the option to confirm the deletion intention. When confirmation is done the specified node is deleted. When updating a node, one can browse the contents as usual. A node can be updated only when it is registered in the named server.

The 'network profile maintenance' utility also sets up a utility for users to do some network configuration management jobs. A good network profile maintenance and network configuration management are the basis of the 'network traffic measurement utility'. The 'network traffic measurement utility' can obtain the correct and efficient traffic information only when a good network profile maintenance is provided. The network profile maintenance is an important job for users.

D. SUMMARY

Network traffic status helps users to realize the network traffic situation of each specified network node. It assists users and network manager in achieving a better network performance and throughput by changing the network applications' schedule and adjusting the network configuration. In other words, available network traffic status is very important for network users and managers. The 'network traffic measurement utility' sets up the network traffic status query environment for users to get the network node traffic status. There are three primary functions in this utility: 'individual host network status query', 'network profile traffic status query', and 'network profile maintenance'. Each function provides the different query and maintenance capability for users and system/network managers.

The 'individual host network status query' function provides the facility for system/network manager and users to query individual computer host network information. This function requests information about a node from the name server. The individual reachability and network node traffic status of specified network node can be tested. There are some information about the systems of users' site which can be queried by the system manager. They will show the statistics reports and others information about the network status, input/output activities, and virtual memory activities. These measurements are useful for system managers to diagnose the computer system and detect the abnormal situation that causes the bottleneck to slow down network activities. The UNIX commands or shell scripts can be used in this utility. It provides a convenient interface for users to communicate with the UNIX operation system. At last, the 'file transfer measurement tools' is available for system/network managers to simulate file transfer from their node to a specified network node. This file transfer simulation measurement method let the managers get the most exact network traffic information between the two network nodes. It also can let users understand the relationships between packet size, file length, and file transfer time. These would offer users the idea about file transfer characteristics between these two network nodes.

Two functions can be used as the tools to get the network traffic status information: 'ECHO data packet-ping utility' and 'file transfer measurement tools'. Each has its own purpose. The 'ECHO data packet-ping utility' is mainly used for the reachability test. The 'file transfer measurement tools' let users get the network traffic status over a time duration. The period depends on the file length and packet size. This function offers users the continual, not discrete, network traffic status information and statistics report. These information will be that approaching the real network traffic situation. But users will pay the time and increase the traffic load on the network. If the specific network node is not so important, the 'ECHO data packet-ping utility' will be suggested for users to measure the network traffic situation. If the network nodes have close relationships with users' network node, it is worth for user to utilize the 'file transfer measurement tools' to get the network traffic situation in details.

The 'network profile traffic status query' function let users have the network profile query ability. In the network profile query, users can query the network traffic of several network nodes of interests at the same time. They will get the traffic statistics and network node traffic status rating reports in the two modes: real time mode and long term mode. The network profile traffic statistics report shows users the round-trip time from users' site to the specified network nodes in the network profile. This report let users realize the network profile traffic situation and traffic statistics information. The network profile traffic status rating subfunction uses the ICMP datagram packet round-trip time and the variance of this round-trip time to rate the network traffic status of each specified network node. The network profile traffic status rating report lists the rate and round-trip time statistics information for the users to analyze the network profile traffic situation.

The real time network traffic reports show the immediate traffic situation over the network. They help users understand the current network traffic. After understanding the current network traffic situation, users can make decisions on the network applications or configuration. So they can avoid the busy traffic time over the network and get a better performance. The long term traffic reports list the long term network traffic status and

statistics information. This would help the network manager find out the network traffic situation and tendency in a long period. The network manager can analyze the characteristics of network traffic situation and detect the possible network traffic problems in the future. The conclusions of network traffic situation analysis and prediction could help network manager to adjust the network configuration. Better network throughput and performance may come true.

The network reachability test will be available in the network profile query. In the network profile query, the network reachability will be tested on the specified network nodes. Because the network reachability is the basis of every network application, it is a very important information.

The 'network profile maintenance' function provides a tool for network users to manage the network configurations. There are three subfunctions in the 'network profile maintenance' function: add, delete and update.

The 'network traffic measurement utility' uses different subdirectories to maintain the network traffic status information. This provides the possible concurrent execution environment for users to monitor the long-term network profile traffic situations in the UNIX background and query the real-time network traffic status at the same time. This will save users' time to collect the network traffic situation and get the correct network traffic data.

In general, the 'network traffic measurement utility' provides an interactive environment for users to monitor the network traffic situation and generate the network traffic reports. These network traffic information would help users to measure the network traffic and network performance. The network measurement result can be used as a basis for the network performance management.

IV. NETWORK TRAFFIC MONITORING AND PERFORMANCE MANAGEMENT: - EXAMPLES AND ANALYSIS

A. INTRODUCTION

Performance and throughput are the primary goals for network performance management. The network traffic status is a good indicator that implies the performance. By monitoring the network traffic, the network managers can do some network performance tuning. The 'network traffic measurement utility' helps the network managers and users measure the network traffic status. According to the collected information, network managers may adjust on the network configuration and achieve a better network performance.

B. ANALYSIS - NPS CAMPUS NETWORK TRAFFIC MONITORING EXAMPLE

We use the campus network of Naval Postgraduate School as an example to demonstrate the network traffic monitoring for network performance management. Figure 4.1 shows the steps taken in monitoring the network traffic on NPS campus.

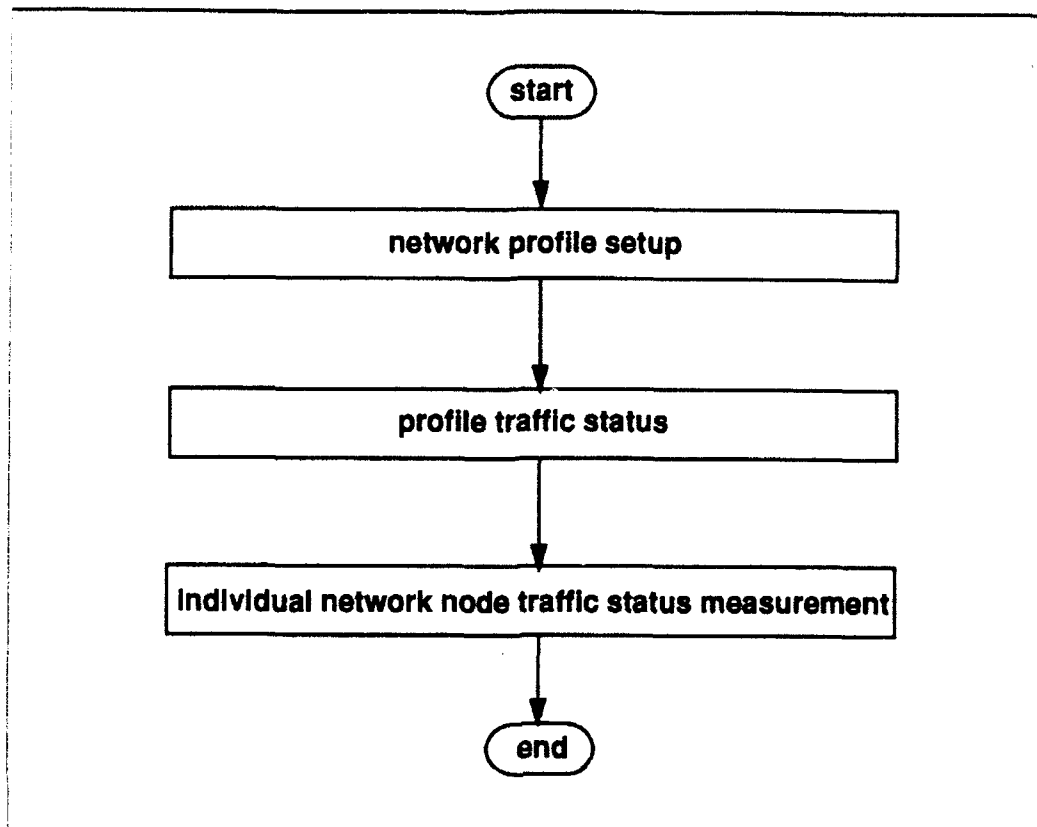


Figure 4.1 NPS Campus Network Traffic Monitoring Flow Chart

From this utility we can generate some network traffic reports on selected nodes. The detailed dialogue of NPS network traffic monitoring can be found in Appendix A. Reachability test is usually the most primitive test. Figure 4.2 shows the reachability test report of NPS campus network. In this report, we find the 'rover.nps.navy.mil' network node cannot be reached. It may be necessary to annotate this network node in the later network traffic status reports.

```
*** Reachability Report ***  
  
nps.navy.mil is alive  
ece.nps.navy.mil is alive  
oc.nps.navy.mil is alive  
cc.nps.navy.mil is alive  
pegasus.nps.navy.mil is alive  
mailboy.cs.nps.navy.mil is alive  
taurus.cs.nps.navy.mil is alive  
no answer from rover.nps.navy.mil  
csrg.nps.navy.mil is alive  
131.120.57.2 is alive
```

Figure 4.2 NPS Campus Network Nodes Reachability Test Report

There are two NPS long-term network traffic status reports that are produced by the sampling traffic status record hourly from each node for one day. Each NPS network node has 24 network traffic status records. These records cover one day's worth of monitoring results.

Figure 4.3 shows the 'long-term network traffic statistics report' of NPS campus network. This report lists the hourly percentage of received ICMP datagram packet and the ICMP datagram packet round trip time of each node. From this report, we find that the traffic is in good shape except that the 'rover' node is not reachable. The report shows that no data packet is received from the node 'rover'. This confirms the reachability test and we conclude that there must be problems in the node 'rover'. The 'oc' network node also has 0% received data packets in the first network traffic record. It tells us there is an abnormal condition in the first sampling period.

We found that the node 'cc' always has a longer data transmission round trip time than the other NPS network nodes. This monitored result can be validated by the physical NPS campus network topology. Most of the nodes are in the same building where the host is located, but the node 'cc' is located in another building. We also found out the idle time and busy time of each NPS node. This would help us to avoid the busy time and to utilize the idle time of each node. More efficient use of the campus network can be achieved. The

round trip time reflects the total data transmission delays of the monitored nodes in the simulation. It may help us in scheduling the network applications.

***** Long Term Traffic Statistics Report *****				
network node name : nps				
time	percent packets received	time round-trip minimum	time round-trip average	time round-trip maximum
Tue Feb 23 13:57:28 PST 1993	100.00	0	3	10
Tue Feb 23 15:00:51 PST 1993	100.00	0	0	10
Tue Feb 23 16:04:24 PST 1993	100.00	0	1	10
Tue Feb 23 17:07:57 PST 1993	100.00	0	0	10
Tue Feb 23 18:11:29 PST 1993	100.00	0	5	10
Tue Feb 23 19:15:00 PST 1993	100.00	0	0	0
Tue Feb 23 20:18:31 PST 1993	100.00	0	0	10
Tue Feb 23 21:22:02 PST 1993	100.00	0	1	10
Tue Feb 23 22:25:34 PST 1993	100.00	0	0	0
Tue Feb 23 23:23:29 PST 1993	100.00	0	1	10
Wed Feb 24 00:32:38 PST 1993	100.00	0	4	10
Wed Feb 24 01:36:10 PST 1993	100.00	0	3	10
Wed Feb 24 02:39:42 PST 1993	100.00	0	0	9
Wed Feb 24 03:43:15 PST 1993	100.00	0	10	20
Wed Feb 24 04:46:53 PST 1993	100.00	0	2	10
Wed Feb 24 05:50:25 PST 1993	100.00	0	1	10
Wed Feb 24 06:53:56 PST 1993	100.00	0	0	10
Wed Feb 24 07:57:27 PST 1993	100.00	0	2	10
Wed Feb 24 09:00:59 PST 1993	100.00	0	0	10
Wed Feb 24 10:04:31 PST 1993	100.00	0	1	20
Wed Feb 24 11:08:03 PST 1993	100.00	0	3	10
Wed Feb 24 12:11:37 PST 1993	100.00	0	3	10
Wed Feb 24 13:15:10 PST 1993	100.00	0	3	10
Wed Feb 24 14:18:43 PST 1993	100.00	0	0	10

Figure 4.3 NPS Campus One Day Network Traffic Statistics Report

network node name : ece				
time	percent packets received	time round-trip minimum	time round-trip average	time round-trip maximum
Tue Feb 23 13:57:28 PST 1993	100.00	0	1	10
Tue Feb 23 15:00:51 PST 1993	100.00	0	0	10
Tue Feb 23 16:04:24 PST 1993	100.00	0	0	9
Tue Feb 23 17:07:57 PST 1993	100.00	0	0	0
Tue Feb 23 18:11:29 PST 1993	100.00	0	0	10
Tue Feb 23 19:15:00 PST 1993	100.00	0	0	10
Tue Feb 23 20:18:31 PST 1993	100.00	0	0	10
Tue Feb 23 21:22:02 PST 1993	100.00	0	0	10
Tue Feb 23 22:25:34 PST 1993	100.00	0	0	9
Tue Feb 23 23:23:29 PST 1993	100.00	0	1	10
Wed Feb 24 00:32:38 PST 1993	100.00	0	0	9
Wed Feb 24 01:36:10 PST 1993	100.00	0	2	10
Wed Feb 24 02:39:42 PST 1993	100.00	0	1	10
Wed Feb 24 03:43:15 PST 1993	100.00	0	6	20
Wed Feb 24 04:46:53 PST 1993	100.00	0	1	10
Wed Feb 24 05:50:25 PST 1993	100.00	0	0	10
Wed Feb 24 06:53:56 PST 1993	100.00	0	0	10
Wed Feb 24 07:57:27 PST 1993	100.00	0	0	10
Wed Feb 24 09:00:59 PST 1993	100.00	0	0	10
Wed Feb 24 10:04:31 PST 1993	100.00	0	0	10
Wed Feb 24 11:08:03 PST 1993	100.00	0	2	10
Wed Feb 24 12:11:37 PST 1993	100.00	0	3	20
Wed Feb 24 13:15:10 PST 1993	100.00	0	4	20
Wed Feb 24 14:18:43 PST 1993	100.00	0	1	10

Figure 4.3, continued

network node name : oc				
time	percent packets received	time round-trip minimum	time round-trip average	time round-trip maximum
Tue Feb 23 13:57:28 PST 1993	*0.00	-	-	-
Tue Feb 23 15:00:51 PST 1993	100.00	0	5	10
Tue Feb 23 16:04:24 PST 1993	100.00	0	5	20
Tue Feb 23 17:07:57 PST 1993	100.00	0	4	10
Tue Feb 23 18:11:29 PST 1993	100.00	0	5	20
Tue Feb 23 19:15:00 PST 1993	100.00	0	2	19
Tue Feb 23 20:18:31 PST 1993	100.00	0	4	19
Tue Feb 23 21:22:02 PST 1993	100.00	0	8	10
Tue Feb 23 22:25:34 PST 1993	100.00	0	8	10
Tue Feb 23 23:23:29 PST 1993	100.00	0	5	10
Wed Feb 24 00:32:38 PST 1993	100.00	0	4	30
Wed Feb 24 01:36:10 PST 1993	100.00	0	7	10
Wed Feb 24 02:39:42 PST 1993	100.00	0	6	10
Wed Feb 24 03:43:15 PST 1993	100.00	0	9	20
Wed Feb 24 04:46:53 PST 1993	100.00	0	3	10
Wed Feb 24 05:50:25 PST 1993	100.00	0	2	10
Wed Feb 24 06:53:56 PST 1993	100.00	0	9	10
Wed Feb 24 07:57:27 PST 1993	100.00	0	10	20
Wed Feb 24 09:00:59 PST 1993	100.00	0	8	10
Wed Feb 24 10:04:31 PST 1993	100.00	9	10	20
Wed Feb 24 11:08:03 PST 1993	100.00	9	10	19
Wed Feb 24 12:11:37 PST 1993	100.00	10	10	10
Wed Feb 24 13:15:10 PST 1993	100.00	0	8	10
Wed Feb 24 14:18:43 PST 1993	100.00	0	2	10

Figure 4.3, continued

network node name : cc				
time	percent packets received	time round-trip minimum	time round-trip average	time round-trip maximum
Tue Feb 23 13:57:28 PST 1993	100.00	20	34	140
Tue Feb 23 15:00:51 PST 1993	100.00	20	32	80
Tue Feb 23 16:04:24 PST 1993	100.00	29	29	30
Tue Feb 23 17:07:57 PST 1993	100.00	29	30	40
Tue Feb 23 18:11:29 PST 1993	100.00	20	31	40
Tue Feb 23 19:15:00 PST 1993	100.00	30	213	1380
Tue Feb 23 20:18:31 PST 1993	100.00	30	30	39
Tue Feb 23 21:22:02 PST 1993	100.00	30	30	40
Tue Feb 23 22:25:34 PST 1993	100.00	30	31	40
Tue Feb 23 23:23:29 PST 1993	100.00	30	30	40
Wed Feb 24 00:32:38 PST 1993	100.00	30	30	30
Wed Feb 24 01:36:10 PST 1993	100.00	30	31	50
Wed Feb 24 02:39:42 PST 1993	100.00	29	32	50
Wed Feb 24 03:43:15 PST 1993	100.00	30	32	40
Wed Feb 24 04:46:53 PST 1993	100.00	29	29	30
Wed Feb 24 05:50:25 PST 1993	100.00	30	30	30
Wed Feb 24 06:53:56 PST 1993	100.00	30	30	40
Wed Feb 24 07:57:27 PST 1993	100.00	30	31	50
Wed Feb 24 09:00:59 PST 1993	100.00	20	31	40
Wed Feb 24 10:04:31 PST 1993	100.00	30	46	140
Wed Feb 24 11:08:03 PST 1993	100.00	30	36	90
Wed Feb 24 12:11:37 PST 1993	100.00	20	32	40
Wed Feb 24 13:15:10 PST 1993	100.00	30	32	40
Wed Feb 24 14:18:43 PST 1993	100.00	30	33	40

Figure 4.3, continued

network node name : pegasus				
time	percent packets received	time round-trip minimum	time round-trip average	time round-trip maximum
Tue Feb 23 13:57:28 PST 1993	100.00	0	0	10
Tue Feb 23 15:00:51 PST 1993	100.00	0	0	9
Tue Feb 23 16:04:24 PST 1993	100.00	0	0	0
Tue Feb 23 17:07:57 PST 1993	100.00	0	0	9
Tue Feb 23 18:11:29 PST 1993	100.00	0	0	9
Tue Feb 23 19:15:00 PST 1993	100.00	0	0	0
Tue Feb 23 20:18:31 PST 1993	100.00	0	0	10
Tue Feb 23 21:22:02 PST 1993	100.00	0	2	10
Tue Feb 23 22:25:34 PST 1993	100.00	0	1	20
Tue Feb 23 23:23:29 PST 1993	100.00	0	0	10
Wed Feb 24 00:32:38 PST 1993	100.00	0	0	10
Wed Feb 24 01:36:10 PST 1993	100.00	0	0	9
Wed Feb 24 02:39:42 PST 1993	100.00	0	0	0
Wed Feb 24 03:43:15 PST 1993	100.00	0	0	10
Wed Feb 24 04:46:53 PST 1993	100.00	0	0	0
Wed Feb 24 05:50:25 PST 1993	100.00	0	0	9
Wed Feb 24 06:53:56 PST 1993	100.00	0	0	9
Wed Feb 24 07:57:27 PST 1993	100.00	0	0	10
Wed Feb 24 09:00:59 PST 1993	100.00	0	1	10
Wed Feb 24 10:04:31 PST 1993	100.00	0	0	10
Wed Feb 24 11:08:03 PST 1993	100.00	0	0	0
Wed Feb 24 12:11:37 PST 1993	100.00	0	0	0
Wed Feb 24 13:15:10 PST 1993	100.00	0	0	0
Wed Feb 24 14:18:43 PST 1993	100.00	0	0	10

Figure 4.3, continued

network node name : cs				
time	percent packets received	time round-trip minimum	time round-trip average	time round-trip maximum
Tue Feb 23 13:57:28 PST 1993	100.00	0	1	10
Tue Feb 23 15:00:51 PST 1993	100.00	0	1	10
Tue Feb 23 16:04:24 PST 1993	100.00	0	2	20
Tue Feb 23 17:07:57 PST 1993	100.00	0	0	10
Tue Feb 23 18:11:29 PST 1993	100.00	0	1	10
Tue Feb 23 19:15:00 PST 1993	100.00	0	0	0
Tue Feb 23 20:18:31 PST 1993	100.00	0	0	10
Tue Feb 23 21:22:02 PST 1993	100.00	0	1	10
Tue Feb 23 22:25:34 PST 1993	100.00	0	0	10
Tue Feb 23 23:23:29 PST 1993	100.00	0	0	10
Wed Feb 24 00:32:38 PST 1993	100.00	0	0	9
Wed Feb 24 01:36:10 PST 1993	100.00	0	0	10
Wed Feb 24 02:39:42 PST 1993	100.00	0	0	10
Wed Feb 24 03:43:15 PST 1993	100.00	0	1	20
Wed Feb 24 04:46:53 PST 1993	100.00	0	0	10
Wed Feb 24 05:50:25 PST 1993	100.00	0	1	10
Wed Feb 24 06:53:56 PST 1993	100.00	0	0	10
Wed Feb 24 07:57:27 PST 1993	100.00	0	0	10
Wed Feb 24 09:00:59 PST 1993	100.00	0	0	10
Wed Feb 24 10:04:31 PST 1993	100.00	0	1	10
Wed Feb 24 11:08:03 PST 1993	100.00	0	0	10
Wed Feb 24 12:11:37 PST 1993	100.00	0	0	10
Wed Feb 24 13:15:10 PST 1993	100.00	0	0	10
Wed Feb 24 14:18:43 PST 1993	100.00	0	2	30

Figure 4.3, continued

network node name : taurus				
time	percent packets received	time round-trip minimum	time round-trip average	time round-trip maximum
Tue Feb 23 13:57:28 PST 1993	100.00	0	0	0
Tue Feb 23 15:00:51 PST 1993	100.00	0	0	10
Tue Feb 23 16:04:24 PST 1993	100.00	0	0	10
Tue Feb 23 17:07:57 PST 1993	100.00	0	0	0
Tue Feb 23 18:11:29 PST 1993	100.00	0	0	0
Tue Feb 23 19:15:00 PST 1993	100.00	0	0	0
Tue Feb 23 20:18:31 PST 1993	100.00	0	1	10
Tue Feb 23 21:22:02 PST 1993	100.00	0	0	0
Tue Feb 23 22:25:34 PST 1993	100.00	0	1	10
Tue Feb 23 23:23:29 PST 1993	100.00	0	0	0
Wed Feb 24 00:32:38 PST 1993	100.00	0	0	10
Wed Feb 24 01:36:10 PST 1993	100.00	0	0	10
Wed Feb 24 02:39:42 PST 1993	100.00	0	0	0
Wed Feb 24 03:43:15 PST 1993	100.00	0	3	50
Wed Feb 24 04:46:53 PST 1993	100.00	0	0	0
Wed Feb 24 05:50:25 PST 1993	100.00	0	0	10
Wed Feb 24 06:53:56 PST 1993	100.00	0	0	10
Wed Feb 24 07:57:27 PST 1993	100.00	0	0	0
Wed Feb 24 09:00:59 PST 1993	100.00	0	0	10
Wed Feb 24 10:04:31 PST 1993	100.00	0	3	60
Wed Feb 24 11:08:03 PST 1993	100.00	0	0	0
Wed Feb 24 12:11:37 PST 1993	100.00	0	0	10
Wed Feb 24 13:15:10 PST 1993	100.00	0	1	10
Wed Feb 24 14:18:43 PST 1993	100.00	0	0	10

Figure 4.3, continued

network node name : rover				
time	percent packets received	time round-trip minimum	time round-trip average	time round-trip maximum
Tue Feb 23 13:57:28 PST 1993	*0.00	-	-	-
Tue Feb 23 15:00:51 PST 1993	*0.00	-	-	-
Tue Feb 23 16:04:24 PST 1993	*0.00	-	-	-
Tue Feb 23 17:07:57 PST 1993	*0.00	-	-	-
Tue Feb 23 18:11:29 PST 1993	*0.00	-	-	-
Tue Feb 23 19:15:00 PST 1993	*0.00	-	-	-
Tue Feb 23 20:18:31 PST 1993	*0.00	-	-	-
Tue Feb 23 21:22:02 PST 1993	*0.00	-	-	-
Tue Feb 23 22:25:34 PST 1993	*0.00	-	-	-
Tue Feb 23 23:23:29 PST 1993	*0.00	-	-	-
Wed Feb 24 00:32:38 PST 1993	*0.00	-	-	-
Wed Feb 24 01:36:10 PST 1993	*0.00	-	-	-
Wed Feb 24 02:39:42 PST 1993	*0.00	-	-	-
Wed Feb 24 03:43:15 PST 1993	*0.00	-	-	-
Wed Feb 24 04:46:53 PST 1993	*0.00	-	-	-
Wed Feb 24 05:50:25 PST 1993	*0.00	-	-	-
Wed Feb 24 06:53:56 PST 1993	*0.00	-	-	-
Wed Feb 24 07:57:27 PST 1993	*0.00	-	-	-
Wed Feb 24 09:00:59 PST 1993	*0.00	-	-	-
Wed Feb 24 10:04:31 PST 1993	*0.00	-	-	-
Wed Feb 24 11:08:03 PST 1993	*0.00	-	-	-
Wed Feb 24 12:11:37 PST 1993	*0.00	-	-	-
Wed Feb 24 13:15:10 PST 1993	*0.00	-	-	-
Wed Feb 24 14:18:43 PST 1993	*0.00	-	-	-

Figure 4.3, continued

network node name : csrg				
time	percent packets received	time round-trip minimum	time round-trip average	time round-trip maximum
Tue Feb 23 13:57:28 PST 1993	100.00	0	1	10
Tue Feb 23 15:00:51 PST 1993	100.00	0	3	20
Tue Feb 23 16:04:24 PST 1993	100.00	0	0	9
Tue Feb 23 17:07:57 PST 1993	100.00	0	0	0
Tue Feb 23 18:11:29 PST 1993	100.00	0	0	10
Tue Feb 23 19:15:00 PST 1993	100.00	0	0	10
Tue Feb 23 20:18:31 PST 1993	100.00	0	0	10
Tue Feb 23 21:22:02 PST 1993	100.00	0	1	20
Tue Feb 23 22:25:34 PST 1993	100.00	0	1	10
Tue Feb 23 23:23:29 PST 1993	100.00	0	0	10
Wed Feb 24 00:32:38 PST 1993	100.00	0	0	10
Wed Feb 24 01:36:10 PST 1993	100.00	0	1	10
Wed Feb 24 02:39:42 PST 1993	100.00	0	1	10
Wed Feb 24 03:43:15 PST 1993	100.00	0	1	10
Wed Feb 24 04:46:53 PST 1993	100.00	0	0	10
Wed Feb 24 05:50:25 PST 1993	100.00	0	0	10
Wed Feb 24 06:53:56 PST 1993	100.00	0	1	10
Wed Feb 24 07:57:27 PST 1993	100.00	0	2	10
Wed Feb 24 09:00:59 PST 1993	100.00	0	0	10
Wed Feb 24 10:04:31 PST 1993	100.00	0	0	9
Wed Feb 24 11:08:03 PST 1993	100.00	0	0	10
Wed Feb 24 12:11:37 PST 1993	100.00	0	1	10
Wed Feb 24 13:15:10 PST 1993	100.00	0	2	20
Wed Feb 24 14:18:43 PST 1993	100.00	0	2	10

Figure 4.3, continued

network node name : 131.120.57.2				
time	percent packets received	time round-trip minimum	time round-trip average	time round-trip maximum
Tue Feb 23 13:57:28 PST 1993	100.00	10	10	10
Tue Feb 23 15:00:51 PST 1993	100.00	10	13	80
Tue Feb 23 16:04:24 PST 1993	100.00	9	12	60
Tue Feb 23 17:07:57 PST 1993	100.00	10	13	60
Tue Feb 23 18:11:29 PST 1993	100.00	10	10	20
Tue Feb 23 19:15:00 PST 1993	100.00	10	13	70
Tue Feb 23 20:18:31 PST 1993	100.00	10	11	20
Tue Feb 23 21:22:02 PST 1993	100.00	10	11	20
Tue Feb 23 22:25:34 PST 1993	100.00	10	17	130
Tue Feb 23 23:23:29 PST 1993	100.00	10	11	30
Wed Feb 24 00:32:38 PST 1993	100.00	9	11	40
Wed Feb 24 01:36:10 PST 1993	100.00	10	10	20
Wed Feb 24 02:39:42 PST 1993	100.00	10	10	20
Wed Feb 24 03:43:15 PST 1993	100.00	10	10	20
Wed Feb 24 04:46:53 PST 1993	100.00	10	10	20
Wed Feb 24 05:50:25 PST 1993	100.00	10	11	40
Wed Feb 24 06:53:56 PST 1993	100.00	10	10	10
Wed Feb 24 07:57:27 PST 1993	100.00	10	13	70
Wed Feb 24 09:00:59 PST 1993	100.00	10	12	60
Wed Feb 24 10:04:31 PST 1993	100.00	10	32	450
Wed Feb 24 11:08:03 PST 1993	100.00	10	16	70
Wed Feb 24 12:11:37 PST 1993	100.00	10	12	60
Wed Feb 24 13:15:10 PST 1993	100.00	10	16	140
Wed Feb 24 14:18:43 PST 1993	100.00	10	13	80

Figure 4.3, continued

Figure 4.4 shows the 'long-term network traffic rating report' at different times. The report also indicates the transmission stability of each node. It is calculated from the data packet transmission round trip time. We found that the 'received packets percent' of 'rover' network node is 0%. It means that there is no data transmission capability in node 'rover' at every sampling time. The node 'rover' gets the worst rating and deserves an attention. The node 'oc' is a bad node in the report as well. This result confirms the first network traffic record in the 'long-term network traffic statistics report'.

The data transmission status of other NPS network nodes show of that the network traffic is good. Although there are some fluctuation at different sampling time, it appears to be tolerable. This report shows that the network traffic situation over NPS campus network is good, except the 'rover' network node.

***** Long Term Network node traffic status rating report *****					
network node name : pegasus					
time	rating	received packet percent	round-trip time mean-value	round-trip time variance	round-trip time std-deviation
Tue Feb23 13:59:10PST1993	1	100.00	0.000	0.000	0.000
Tue Feb23 15:02:49PST1993	1	100.00	0.500	5.000	2.236
Tue Feb23 16:06:18PST1993	2	100.00	0.500	5.000	2.236
Tue Feb23 17:09:46PST1993	3	100.00	1.000	9.474	3.078
Tue Feb23 18:13:15PST1993	2	100.00	0.500	5.000	2.236
Tue Feb23 19:16:42PST1993	1	100.00	0.000	0.000	0.000
Tue Feb23 20:20:10PST1993	4	100.00	1.500	13.421	3.663
Tue Feb23 21:23:38PST1993	3	100.00	1.000	9.474	3.078
Tue Feb23 22:27:07PST1993	1	100.00	0.000	0.000	0.000
Tue Feb23 23:30:36PST1993	2	100.00	0.500	5.000	2.236
Wed Feb24 00:34:04PST1993	1	100.00	0.000	0.000	0.000
Wed Feb24 01:37:32PST1993	1	100.00	0.000	0.000	0.000
Wed Feb24 02:41:00PST1993	2	100.00	0.450	4.050	2.012
Wed Feb24 03:44:28PST1993	3	100.00	1.500	13.421	3.663
Wed Feb24 04:47:57PST1993	1	100.00	0.000	0.000	0.000
Wed Feb24 05:51:25PST1993	2	100.00	0.500	5.000	2.236
Wed Feb24 06:54:53PST1993	1	100.00	0.000	0.000	0.000
Wed Feb24 07:58:21PST1993	1	100.00	0.000	0.000	0.000
Wed Feb24 09:01:48PST1993	2	100.00	0.450	4.050	2.012
Wed Feb24 10:05:16PST1993	3	100.00	0.500	5.000	2.236
Wed Feb24 11:08:44PST1993	2	100.00	0.500	5.000	2.236
Wed Feb24 12:12:14PST1993	2	100.00	0.500	5.000	2.236
Wed Feb24 13:15:43PST1993	1	100.00	0.000	0.000	0.000
Wed Feb24 14:19:13PST1993	3	100.00	1.000	9.474	3.078

Figure 4.4 NPS Campus One Day Network Nodes Traffic Rating Report

network node name : cs					
time	rating	packet percent	received round-trip	round-trip	round-trip
			time mean-value	time variance	time std-deviation
Tue Feb23 13:59:10PST1993	1	100.00	0.000	0.000	0.000
Tue Feb23 15:02:49PST1993	4	100.00	2.500	19.737	4.443
Tue Feb23 16:06:18PST1993	3	100.00	1.000	9.474	3.078
Tue Feb23 17:09:46PST1993	4	100.00	1.500	23.947	4.894
Tue Feb23 18:13:15PST1993	1	100.00	0.000	0.000	0.000
Tue Feb23 19:16:42PST1993	6	100.00	1.000	20.000	4.472
Tue Feb23 20:20:10PST1993	3	100.00	1.000	9.474	3.078
Tue Feb23 21:23:38PST1993	3	100.00	1.000	9.474	3.078
Tue Feb23 22:27:07PST1993	5	100.00	2.000	37.895	6.156
Tue Feb23 23:30:36PST1993	2	100.00	0.500	5.000	2.236
Wed Feb24 00:34:04PST1993	4	100.00	2.000	16.842	4.104
Wed Feb24 01:37:32PST1993	3	100.00	0.500	5.000	2.236
Wed Feb24 02:41:00PST1993	4	100.00	2.000	8.576	2.929
Wed Feb24 03:44:28PST1993	1	100.00	0.500	5.000	2.236
Wed Feb24 04:47:57PST1993	4	100.00	0.950	9.474	3.078
Wed Feb24 05:51:25PST1993	2	100.00	0.500	5.000	2.236
Wed Feb24 06:54:53PST1993	4	100.00	1.000	9.474	3.078
Wed Feb24 07:58:21PST1993	3	100.00	0.500	5.000	2.236
Wed Feb24 09:01:48PST1993	1	100.00	0.000	0.000	0.000
Wed Feb24 10:05:16PST1993	3	100.00	0.500	5.000	2.236
Wed Feb24 11:08:44PST1993	2	100.00	0.500	5.000	2.236
Wed Feb24 12:12:14PST1993	7	100.00	4.000	35.789	5.982
Wed Feb24 13:15:43PST1993	4	100.00	1.450	12.576	3.546
Wed Feb24 14:19:13PST1993	1	100.00	0.000	0.000	0.000

Figure 4.4 continued

network node name : taurus						
time	rating	received packet percent	round-trip time mean-value	round-trip time variance	round-trip time std-deviation	
Tue Feb23 13:59:10PST1993	2	100.00	0.450	4.050	2.012	
Tue Feb23 15:02:49PST1993	3	100.00	2.000	16.842	4.104	
Tue Feb23 16:06:18PST1993	1	100.00	0.000	0.000	0.000	
Tue Feb23 17:09:46PST1993	6	100.00	2.500	30.263	5.501	
Tue Feb23 18:13:15PST1993	1	100.00	0.000	0.000	0.000	
Tue Feb23 19:16:42PST1993	1	100.00	0.000	0.000	0.000	
Tue Feb23 20:20:10PST1993	2	100.00	0.500	5.000	2.263	
Tue Feb23 21:23:38PST1993	3	100.00	1.000	9.474	3.078	
Tue Feb23 22:27:07PST1993	2	100.00	0.450	4.050	2.012	
Tue Feb23 23:30:36PST1993	1	100.00	0.000	0.000	0.000	
Wed Feb24 00:34:04PST1993	1	100.00	0.000	0.000	0.000	
Wed Feb24 01:37:32PST1993	3	100.00	0.500	5.000	2.236	
Wed Feb24 02:41:00PST1993	1	100.00	0.000	0.000	0.000	
Wed Feb24 03:44:28PST1993	1	100.00	0.500	5.000	2.236	
Wed Feb24 04:47:57PST1993	1	100.00	0.000	0.000	0.000	
Wed Feb24 05:51:25PST1993	1	100.00	0.000	0.000	0.000	
Wed Feb24 06:54:53PST1993	3	100.00	0.500	5.000	2.236	
Wed Feb24 07:58:21PST1993	6	100.00	2.000	27.386	5.231	
Wed Feb24 09:01:48PST1993	1	100.00	0.000	0.000	0.000	
Wed Feb24 10:05:16PST1993	2	100.00	0.450	4.050	2.012	
Wed Feb24 11:08:44PST1993	3	100.00	0.900	7.674	2.770	
Wed Feb24 12:12:14PST1993	4	100.00	0.950	18.050	4.249	
Wed Feb24 13:15:43PST1993	2	100.00	0.500	5.000	2.236	
Wed Feb24 14:19:13PST1993	2	100.00	0.500	5.000	2.236	

Figure 4.4 continued

network node name : csrg					
time	rating	packet percent	received round-trip	round-trip	round-trip
			time mean-value	time variance	time std-deviation
Tue Feb23 13:59:10PST1993	3	100.00	0.500	5.000	2.236
Tue Feb23 15:02:49PST1993	5	100.00	3.000	22.105	4.702
Tue Feb23 16:06:18PST1993	2	100.00	0.500	5.000	2.236
Tue Feb23 17:09:46PST1993	2	100.00	0.500	5.000	2.236
Tue Feb23 18:13:15PST1993	1	100.00	0.000	0.000	0.000
Tue Feb23 19:16:42PST1993	2	100.00	0.500	5.000	2.236
Tue Feb23 20:20:10PST1993	1	100.00	0.000	0.000	0.000
Tue Feb23 21:23:38PST1993	2	100.00	0.500	5.000	2.236
Tue Feb23 22:27:07PST1993	3	100.00	1.000	9.474	3.078
Tue Feb23 23:30:36PST1993	3	100.00	1.000	9.474	3.078
Wed Feb24 00:34:04PST1993	2	100.00	0.500	5.000	2.236
Wed Feb24 01:37:32PST1993	2	100.00	0.450	4.050	2.012
Wed Feb24 02:41:00PST1993	1	100.00	0.000	0.000	0.000
Wed Feb24 03:44:28PST1993	2	100.00	1.000	9.474	3.078
Wed Feb24 04:47:57PST1993	2	100.00	0.450	4.050	2.012
Wed Feb24 05:51:25PST1993	2	100.00	0.500	5.000	2.236
Wed Feb24 06:54:53PST1993	1	100.00	0.000	0.000	0.000
Wed Feb24 07:58:21PST1993	2	100.00	0.450	4.050	2.012
Wed Feb24 09:01:48PST1993	6	100.00	2.950	31.945	5.652
Wed Feb24 10:05:16PST1993	4	100.00	2.000	16.842	4.104
Wed Feb24 11:08:44PST1993	6	100.00	1.500	13.421	3.663
Wed Feb24 12:12:14PST1993	3	100.00	1.500	13.421	3.663
Wed Feb24 13:15:43PST1993	6	100.00	2.500	19.737	4.443
Wed Feb24 14:19:13PST1993	6	100.00	6.500	402.895	20.072

Figure 4.4 continued

network node name : ece					
time	rating	packet percent	received round-trip	round-trip	round-trip
			time mean-value	time variance	time std-deviation
Tue Feb23 13:59:10PST1993	4	100.00	1.500	13.421	3.663
Tue Feb23 15:02:49PST1993	2	100.00	1.500	13.421	3.663
Tue Feb23 16:06:18PST1993	3	100.00	1.000	9.474	3.078
Tue Feb23 17:09:46PST1993	2	100.00	0.500	5.000	2.236
Tue Feb23 18:13:15PST1993	2	100.00	0.500	5.000	2.236
Tue Feb23 19:16:42PST1993	3	100.00	1.000	9.474	3.078
Tue Feb23 20:20:10PST1993	3	100.00	1.000	9.474	3.078
Tue Feb23 21:23:38PST1993	2	100.00	0.500	5.000	2.236
Tue Feb23 22:27:07PST1993	3	100.00	1.000	9.474	3.078
Tue Feb23 23:30:36PST1993	4	100.00	1.500	13.421	3.663
Wed Feb24 00:34:04PST1993	3	100.00	1.500	13.421	3.663
Wed Feb24 01:37:32PST1993	3	100.00	0.500	5.000	2.236
Wed Feb24 02:41:00PST1993	5	100.00	1.450	12.576	3.546
Wed Feb24 03:44:28PST1993	5	100.00	4.000	25.263	5.026
Wed Feb24 04:47:57PST1993	3	100.00	0.500	5.000	2.236
Wed Feb24 05:51:25PST1993	3	100.00	1.000	9.474	3.078
Wed Feb24 06:54:53PST1993	4	100.00	1.000	9.474	3.078
Wed Feb24 07:58:21PST1993	7	100.00	5.000	68.421	8.272
Wed Feb24 09:01:48PST1993	3	100.00	0.950	8.576	2.929
Wed Feb24 10:05:16PST1993	4	100.00	2.000	16.842	4.104
Wed Feb24 11:08:44PST1993	4	100.00	1.000	9.474	3.078
Wed Feb24 12:12:14PST1993	5	100.00	2.450	18.997	4.359
Wed Feb24 13:15:43PST1993	7	100.00	2.000	27.368	5.231
Wed Feb24 14:19:13PST1993	3	100.00	1.000	9.474	3.078

Figure 4.4 continued

network node name : nps					
time	rating	received	round-trip	round-trip	round-trip
		packet	time	time	time
		percent	mean-value	variance	std-deviation
Tue Feb23 13:59:10PST1993	5	100.00	6.500	23.947	4.894
Tue Feb23 15:02:49PST1993	4	100.00	2.500	19.737	4.443
Tue Feb23 16:06:18PST1993	5	100.00	1.950	16.050	4.006
Tue Feb23 17:09:46PST1993	5	100.00	2.450	29.524	5.434
Tue Feb23 18:13:15PST1993	2	100.00	0.500	5.000	2.236
Tue Feb23 19:16:42PST1993	5	100.00	1.500	13.421	3.663
Tue Feb23 20:20:10PST1993	5	100.00	2.450	18.997	4.359
Tue Feb23 21:23:38PST1993	4	100.00	1.450	12.576	3.546
Tue Feb23 22:27:07PST1993	4	100.00	2.000	16.842	4.014
Tue Feb23 23:30:36PST1993	5	100.00	1.950	16.050	4.006
Wed Feb24 00:34:04PST1993	3	100.00	1.500	13.421	3.663
Wed Feb24 01:37:32PST1993	4	100.00	1.000	9.474	3.078
Wed Feb24 02:41:00PST1993	7	100.00	3.500	23.947	4.894
Wed Feb24 03:44:28PST1993	4	100.00	2.500	19.737	4.443
Wed Feb24 04:47:57PST1993	4	100.00	1.000	9.474	3.078
Wed Feb24 05:51:25PST1993	2	100.00	0.500	5.000	2.236
Wed Feb24 06:54:53PST1993	5	100.00	5.000	26.316	5.130
Wed Feb24 07:58:21PST1993	5	100.00	4.000	25.263	5.026
Wed Feb24 09:01:48PST1993	4	100.00	1.000	9.474	3.078
Wed Feb24 10:05:16PST1993	3	100.00	0.500	5.000	2.236
Wed Feb24 11:08:44PST1993	5	100.00	1.400	11.726	3.424
Wed Feb24 12:12:14PST1993	9	100.00	8.500	87.105	9.333
Wed Feb24 13:15:43PST1993	5	100.00	1.500	13.421	3.663
Wed Feb24 14:19:13PST1993	4	100.00	1.950	16.050	4.006

Figure 4.4 continued

network node name : 131.120.57.2						
time	rating	received round-trip		round-trip		round-trip
		packet	time	time	time	
		percent	mean-value	variance	std-deviation	
Tue Feb23 13:59:10PST1993	6	100.00	13.500	45.000	6.708	
Tue Feb23 15:02:49PST1993	1	100.00	10.500	5.000	2.236	
Tue Feb23 16:06:18PST1993	1	100.00	10.000	0.000	0.000	
Tue Feb23 17:09:46PST1993	1	100.00	10.000	0.000	0.000	
Tue Feb23 18:13:15PST1993	2	100.00	10.500	5.000	2.236	
Tue Feb23 19:16:42PST1993	1	100.00	10.000	0.000	0.000	
Tue Feb23 20:20:10PST1993	1	100.00	10.000	0.000	0.000	
Tue Feb23 21:23:38PST1993	2	100.00	10.500	5.000	2.236	
Tue Feb23 22:27:07PST1993	1	100.00	10.000	0.000	0.000	
Tue Feb23 23:30:36PST1993	2	100.00	10.500	5.000	2.236	
Wed Feb24 00:34:04PST1993	1	100.00	10.000	0.000	0.000	
Wed Feb24 01:37:32PST1993	1	100.00	10.000	0.000	0.000	
Wed Feb24 02:41:00PST1993	1	100.00	10.000	0.000	0.000	
Wed Feb24 03:44:28PST1993	2	100.00	11.000	9.474	3.078	
Wed Feb24 04:47:57PST1993	1	100.00	10.000	0.000	0.000	
Wed Feb24 05:51:25PST1993	1	100.00	10.000	0.000	0.000	
Wed Feb24 06:54:53PST1993	4	100.00	11.000	9.474	3.078	
Wed Feb24 07:58:21PST1993	1	100.00	10.000	0.000	0.000	
Wed Feb24 09:01:48PST1993	1	100.00	10.000	0.000	0.000	
Wed Feb24 10:05:16PST1993	1	100.00	10.000	0.000	0.000	
Wed Feb24 11:08:44PST1993	1	100.00	10.000	0.000	0.000	
Wed Feb24 12:12:14PST1993	1	100.00	10.000	0.000	0.000	
Wed Feb24 13:15:43PST1993	3	100.00	10.400	5.200	2.280	
Wed Feb24 14:19:13PST1993	1	100.00	10.000	0.000	0.000	

Figure 4.4 continued

network node name : cc						
time	rating	packet	received round-trip		round-trip round-trip	
			percent	time mean-value	time variance	time std-deviation
Tue Feb23 13:59:10PST1993	7	100.00	159.500523	10.262	228.714	
Tue Feb23 15:02:49PST1993	1	100.00	29.500	5.000	2.236	
Tue Feb23 16:06:18PST1993	4	100.00	31.500	13.421	3.663	
Tue Feb23 17:09:46PST1993	3	100.00	31.000	9.474	3.078	
Tue Feb23 18:13:15PST1993	2	100.00	30.500	5.000	2.236	
Tue Feb23 19:16:42PST1993	4	100.00	30.000	10.526	2.244	
Tue Feb23 20:20:10PST1993	1	100.00	30.000	0.000	0.000	
Tue Feb23 21:23:38PST1993	2	100.00	29.500	5.000	2.236	
Tue Feb23 22:27:07PST1993	6	100.00	32.950	127.734	11.302	
Tue Feb23 23:30:36PST1993	7	100.00	33.500	87.105	9.333	
Wed Feb24 00:34:04PST1993	5	100.00	31.500	23.947	4.894	
Wed Feb24 01:37:32PST1993	6	100.00	32.000	27.368	5.231	
Wed Feb24 02:41:00PST1993	3	100.00	30.500	5.000	2.236	
Wed Feb24 03:44:28PST1993	6	100.00	34.000	56.842	7.539	
Wed Feb24 04:47:57PST1993	2	100.00	30.450	4.050	2.012	
Wed Feb24 05:51:25PST1993	3	100.00	31.000	9.474	3.078	
Wed Feb24 06:54:53PST1993	2	100.00	29.950	0.050	0.224	
Wed Feb24 07:58:21PST1993	4	100.00	30.900	9.994	3.144	
Wed Feb24 09:01:48PST1993	7	100.00	35.500	362.895	19.050	
Wed Feb24 10:05:16PST1993	5	100.00	61.500	3202.895	56.594	
Wed Feb24 11:08:44PST1993	7	100.00	41.500	560.789	23.681	
Wed Feb24 12:12:14PST1993	6	100.00	32.900	31.253	5.590	
Wed Feb24 13:15:43PST1993	8	100.00	32.450	82.155	9.064	
Wed Feb24 14:19:13PST1993	3	100.00	31.000	9.474	3.078	

Figure 4.4 continued

network node name : oc					
time	rating	received round-trip		round-trip	round-trip
		packet	time	time	time
		percent	mean-value	variance	std-deviation
Tue Feb23 13:59:10PST1993	8	*0.00	0.000	0.000	0.000
Tue Feb23 15:02:49PST1993	2	100.00	8.500	13.421	3.663
Tue Feb23 16:06:18PST1993	6	100.00	6.450	23.629	4.861
Tue Feb23 17:09:46PST1993	4	100.00	3.500	23.947	4.894
Tue Feb23 18:13:15PST1993	3	100.00	5.000	26.316	5.130
Tue Feb23 19:16:42PST1993	7	100.00	4.000	25.263	5.026
Tue Feb23 20:20:10PST1993	1	100.00	10.000	0.000	0.000
Tue Feb23 21:23:38PST1993	1	100.00	10.000	0.000	0.000
Tue Feb23 22:27:07PST1993	7	100.00	10.000	157.895	12.566
Tue Feb23 23:30:36PST1993	6	100.00	7.950	16.682	4.084
Wed Feb24 00:34:04PST1993	6	100.00	5.500	36.579	6.048
Wed Feb24 01:37:32PST1993	5	100.00	4.000	25.263	5.026
Wed Feb24 02:41:00PST1993	6	100.00	11.000	20.000	4.472
Wed Feb24 03:44:28PST1993	2	100.00	11.000	9.474	3.078
Wed Feb24 04:47:57PST1993	1	100.00	10.000	0.000	0.000
Wed Feb24 05:51:25PST1993	1	100.00	10.000	0.000	0.000
Wed Feb24 06:54:53PST1993	1	100.00	10.000	0.000	0.000
Wed Feb24 07:58:21PST1993	3	100.00	10.500	5.000	2.236
Wed Feb24 09:01:48PST1993	5	100.00	6.900	21.568	4.644
Wed Feb24 10:05:16PST1993	3	100.00	9.500	5.000	2.236
Wed Feb24 11:08:44PST1993	1	100.00	10.000	0.000	0.000
Wed Feb24 12:12:14PST1993	8	100.00	12.000	48.421	6.959
Wed Feb24 13:15:43PST1993	2	100.00	9.500	5.000	2.236
Wed Feb24 14:19:13PST1993	5	100.00	7.450	19.524	4.419

Figure 4.4 continued

network node name : rover						
time	rating	received packet percent	round-trip time mean-value	round-trip time variance	round-trip time std-deviation	
Tue Feb23 13:59:10PST1993	8	*0.00	0.000	0.000	0.000	
Tue Feb23 15:02:49PST1993	6	*0.00	0.000	0.000	0.000	
Tue Feb23 16:06:18PST1993	7	*0.00	0.000	0.000	0.000	
Tue Feb23 17:09:46PST1993	7	*0.00	0.000	0.000	0.000	
Tue Feb23 18:13:15PST1993	4	*0.00	0.000	0.000	0.000	
Tue Feb23 19:16:42PST1993	8	*0.00	0.000	0.000	0.000	
Tue Feb23 20:20:10PST1993	6	*0.00	0.000	0.000	0.000	
Tue Feb23 21:23:38PST1993	5	*0.00	0.000	0.000	0.000	
Tue Feb23 22:27:07PST1993	8	*0.00	0.000	0.000	0.000	
Tue Feb23 23:30:36PST1993	8	*0.00	0.000	0.000	0.000	
Wed Feb24 00:34:04PST1993	7	*0.00	0.000	0.000	0.000	
Wed Feb24 01:37:32PST1993	7	*0.00	0.000	0.000	0.000	
Wed Feb24 02:41:00PST1993	8	*0.00	0.000	0.000	0.000	
Wed Feb24 03:44:28PST1993	7	*0.00	0.000	0.000	0.000	
Wed Feb24 04:47:57PST1993	5	*0.00	0.000	0.000	0.000	
Wed Feb24 05:51:25PST1993	4	*0.00	0.000	0.000	0.000	
Wed Feb24 06:54:53PST1993	6	*0.00	0.000	0.000	0.000	
Wed Feb24 07:58:21PST1993	8	*0.00	0.000	0.000	0.000	
Wed Feb24 09:01:48PST1993	8	*0.00	0.000	0.000	0.000	
Wed Feb24 10:05:16PST1993	6	*0.00	0.000	0.000	0.000	
Wed Feb24 11:08:44PST1993	8	*0.00	0.000	0.000	0.000	
Wed Feb24 12:12:14PST1993	10	*0.00	0.000	0.000	0.000	
Wed Feb24 13:15:43PST1993	9	*0.00	0.000	0.000	0.000	
Wed Feb24 14:19:13PST1993	7	*0.00	0.000	0.000	0.000	

Figure 4.4 continued

Observing the NPS campus network traffic situation, we select the 'ece' network node as the server and our local host as the client to simulate the file transfer process. Through the file transfer simulation, we can get the on-going network traffic status information between the server and client network nodes. First the reachability of 'ece' network node is tested by using the 'ECHO data packet-ping utility' function. We then start the file transfer simulation program at both sites of the server and the client. After the file transfer simulation began, the 'show network status utility' function, 'I/O statistics report utility' function and 'virtual memory statistics report utility' function were used to monitor further

network status. It also generates I/O and the virtual memory statistics reports of the client computer host.

Figure 4.5 shows the current active sockets in our network node. There the first TCP active socket record displays the network traffic information in our file transfer simulation.

It includes the following network information:

1. active socket protocol: TCP
2. data packet size: 1024 bytes
3. local address: 131.120.5.22 (IP address of our network node)
4. port number of local address: 1187
5. foreign address: 131.120.20.2 (IP address of 'ece' network node)
6. port number of foreign address: 7003
7. state: ESTABLISHED

These information showed the situation of an active socket invoked by the file transfer simulation. From these information it possible to find out that the file transfer simulation works well.

Active Internet Connection (including servers)				
PCB	Proto	Recv-Q	Send-Q	Local Address Foreign Address (state)
ffa5ee0c	udp	0	0	*.1804 **
ffa6408c	udp	0	0	*.1798 **
ffa6b78c	udp	0	0	*.645 **
ffa69b8c	udp	0	0	*.1022 **
ffa6768c	udp	0	0	*.1020 **
ffa7980c	udp	0	0	* **
ffa5a50c	udp	0	0	*.1345 **
ffa4e90c	udp	0	0	*.695 **
ffa4e30c	udp	0	0	*.696 **
ffa5700c	udp	0	0	*.1053 **
ffa5718c	udp	0	0	*.1052 **
ffa5748c	udp	0	0	*.1051 **
ffa5740c	udp	0	0	*.1050 **
ffa5750c	udp	0	0	*.1049 **
ffa5768c	udp	0	0	*.19 **
ffa5790c	udp	0	0	*.13 **
ffa57b8c	udp	0	0	*.9 **
ffa57e8c	udp	0	0	*.7 **
ffa5408c	udp	0	0	*.37 **
ffa5458c	udp	0	0	*.517 **
ffa5470c	udp	0	0	*.512 **
ffa54e0c	udp	0	0	*.42 **
ffa54b8c	udp	0	0	*.736 **
ffa5548c	udp	0	0	*.733 **
ffa4e80c	udp	0	0	*.728 **
ffa4e40c	udp	0	0	*.705 **
ffa5578c	udp	0	0	*.725 **
ffa5568c	udp	0	0	*.724 **
ffa55a8c	udp	0	0	*.1048 **
ffa55b8c	udp	0	0	*.1047 **
ffa55d8c	udp	0	0	*.1046 **
ffa55d0c	udp	0	0	*.1045 **
ffa4e70c	udp	0	0	*.706 **
ffa4f08c	udp	0	0	* **
ffa4ea0c	udp	0	0	*.699 **
ffa4e98c	udp	0	0	*.514 **
ffa4ed0c	udp	0	0	*.1023 **
ffa4f40c	udp	0	0	*.520 **
ffa4f60c	udp	0	0	*.659 **
ffa4f80c	udp	0	0	*.1027 **
ffa4fc0c	udp	0	0	*.1043 **
ffa4fa0c	udp	0	0	*.111 **
ffa4fc8c	udp	0	0	*.1025 **
ffa6d80c	tcp	0	1024	131.120.5.22.1187 131.120.20.2.7003 ESTABLISHED

Figure 4.5 Active Sockets Information Report

ffa6c38c tcp	0	0 *.6000	**	LISTEN
ffa7040c tcp	0	0 1056	**	LISTEN
ffa56e8c tcp	0	0 *.1027	**	LISTEN
ffa5788c tcp	0	0 *.19	**	LISTEN
ffa57a0c tcp	0	0 *.13	**	LISTEN
ffa57d0c tcp	0	0 *.9	**	LISTEN
ffa5400c tcp	0	0 *.7	**	LISTEN
ffa5420c tcp	0	0 *.37	**	LISTEN
ffa5438c tcp	0	0 *.79	**	LISTEN
ffa5448c tcp	0	0 *.540	**	LISTEN
ffa5478c tcp	0	0 *.512	**	LISTEN
ffa5480c tcp	0	0 *.513	**	LISTEN
ffa5500c tcp	0	0 *.514	**	LISTEN
ffa54d0c tcp	0	0 *.515	**	LISTEN
ffa54e8c tcp	0	0 *.23	**	LISTEN
ffa5540c tcp	0	0 *.21	**	LISTEN
ffa5570c tcp	0	0 *.721	**	LISTEN
ffa5590c tcp	0	0 *.718	**	LISTEN
ffa55c0c tcp	0	0 *.713	**	LISTEN
ffa4e08c tcp	0	0 *.709	**	LISTEN
ffa4e38c tcp	0	0 *.708	**	LISTEN
ffa4f10c tcp	0	0 *.25	**	LISTEN
ffa4f48c tcp	0	0 *	**	LISTEN
ffa4fd8c tcp	0	0 *.1024	**	LISTEN
ffa4f88c tcp	0	0 *.111	**	LISTEN

Figure 4.5 continued

We also queried the I/O and others system information during the file transfer simulation. Figure 4.6 and Figure 4.7 showed the I/O activities and other system information including the process, real/virtual memory, page swapping, page fault, CPU time, and disk operation in our computer host. (The meaning of each field in Figure 4.6 & 4.7, can be found in Appendix B.) The system/network manager could use these system information to monitor and diagnose the computer system operations. With the reports, the network managers may repair or remove problems that caused the bottleneck of the network traffic.

tty				sr0			cpu		
tin	tout	rpi	wpi	util	us	ni	sy	id	
0	4	29608	36094	0.4	1	0	2	97	
0	38	0	0	0.0	4	0	21	74	
0	478	1	0	2.0	17	0	18	65	
0	37	0	0	0.0	3	0	12	84	
0	37	0	0	0.0	2	0	24	74	
0	37	0	0	0.0	4	0	19	77	
0	37	0	0	0.0	5	0	19	76	
0	37	0	0	0.0	4	0	21	75	
0	37	0	0	0.0	8	0	29	63	
0	485	0	7	10.9	18	0	21	61	

Figure 4.6 I/O Statistics Report

procs			memory			page			disk			faults			cpu						
r	b	w	avm	fre	si	so	pi	po	fr	de	sr	s0	s1	s2	s3	in	sy	cs	us	sy	id
1	0	0	0	1632	45	45	0	0	0	0	0	0	0	0	0	0	71	14	1	2	97
0	0	0	0	1608	0	0	0	0	0	0	0	0	0	0	0	0	432	173	15	24	62
0	0	0	0	1608	0	0	0	0	0	0	0	0	0	0	0	0	404	169	4	28	77
0	0	0	0	1552	0	0	0	0	0	0	0	0	0	0	0	0	388	168	1	23	76
0	0	0	0	1552	0	0	0	0	0	0	0	0	0	0	0	0	378	168	3	24	73
0	0	0	0	1552	0	0	0	0	0	0	0	0	0	0	0	0	372	167	4	24	71
0	0	0	0	1536	0	0	0	0	0	0	0	0	0	0	0	0	411	166	5	25	80
0	0	0	0	1536	0	0	0	0	0	0	0	4	0	0	0	0	392	165	5	23	72
0	0	0	0	1536	0	0	0	0	0	0	0	0	0	0	0	0	382	163	15	15	70
0	0	0	0	1536	0	0	0	0	0	0	0	0	0	0	0	0	374	164	5	20	75

Figure 4.7 Virtual Memory Statistics Report

```

***** File Transfer Simulation Statistics Report *****
The address of server      : 131.120.20.2
The file name is          : TCP_1024.1MB
The file length is        : 1048576 bytes
The sample number of record is : 100
The time hour interval range is : per 10 minutes

*** File transfer simulation time range ***

File transfer simulation start time: Wed Mar 3 01:28:49 1993
File transfer simulation end time : Wed Mar 3 01:51:16 1993

The file transfer simulation time hour interval begin: Wed Mar 3 01:28:49 1993
The file transfer simulation time hour interval end   : Wed Mar 3 01:38:49 1993
The percent in this time hour interval is           : 44.00

The file transfer simulation time hour interval begin: Wed Mar 3 01:38:49 1993
The file transfer simulation time hour interval end   : Wed Mar 3 01:48:49 1993
The percent in this time hour interval is           : 45.00

The file transfer simulation time hour interval begin: Wed Mar 3 01:48:49 1993
The file transfer simulation time hour interval end   : Wed Mar 3 01:51:16 1993
The percent in this time hour interval is           : 11.00

*** File transfer with disk I/O ***

The mean value is          : 13.187021 seconds
The variance               : 0.194408 seconds
The standard deviation is   : 0.440918 seconds
The max file transfer time is : 15 sec, 131321 usec
The min file transfer time is : 12 sec, 796478 usec

The file transfer time range from 12 seconds to 13 seconds is : 39.00 percent
The file transfer time range from 13 seconds to 14 seconds is : 53.00 percent
The file transfer time range from 14 seconds to 15 seconds is : 7.00 percent
The file transfer time range from 15 seconds to 16 seconds is : 1.00 percent

*** File transfer with disk I/O ***

The mean value is          : 12.499682seconds
The variance               : 0.182435 seconds
The standard deviation is   : 0.427124 seconds
The max file transfer time is : 14 sec,468310 usec
The min file transfer time is : 12 sec,130410 usec

The file transfer time range from 12 seconds to 13 seconds is : 91.00 percent
The file transfer time range from 13 seconds to 14 seconds is : 8.00 percent
The file transfer time range from 14 seconds to 15 seconds is : 1.00 percent

*** End of file transfer data statistics function ***

```

Figure 4.8 File Transfer Simulation Statistics Report

When we finish the file transfer simulation, we generate the 'file transfer simulation statistics report' in Figure 4.8 to show the network traffic status between our host and the 'ece' node. In this 100 samples file transfer simulation report, we get a lot of network traffic measurements. First, It found that this file transfer simulation used about 23 minutes. There are 44% file transfers in the first ten minutes and 45% file transfers in the interval between 10th minute and 19th minute. In the last three minutes, there are 11% file transfers. This showed that the file transfer simulation samples distributed evenly in the simulation duration. It showed that there was a stable network traffic between these two network nodes during the simulation period. Let us look at the file transfer time used to transfer an one Mbytes file. There are two types of file transfer time: file transfer with disk I/O time and file transfer without disk I/O time. The file transfer with disk I/O is the actual time spent in the file transfer simulation. The file transfer without disk I/O time is the time spent in the network. The average time of file transfer with disk I/Os is 13.19 seconds. There are 92 samples in file transfer simulation using 12-14 seconds. As for the file transfer without disk I/O time, the average time is 12.50 seconds. The total 91 file transfers used 12-13 seconds. From the average file transfer time and the percentage, the continuous network traffic between our host node and the 'ece' network node appears to be very stable.

C. SUMMARY

The 'network traffic measurement utility' implements many facilities for users and network managers to query the specified network traffic status. The query results will help them do adjustments and tune the system for better performance and throughput. In NPS campus network traffic monitoring example, the 'network traffic measurement utility' is used to monitor the NPS campus network traffic status for one day. The long-term monitoring produced good results. For example, we noticed that the node 'rover' is down. Additionally, a 'ece' network node was selected for testing the performance of file transfer simulation. The continual network traffic throughput was measured. During the file transfer simulation, system information of our computer host were gathered. It is possible to notice

whether there is any network bottleneck. In our experiments, both results show a stable campus network viewing from our local host. Timely and accurate information on the network status will help a network manager enhance the network throughput and performance. A proper use of the tool will achieve the primary goal of the network performance management.

V. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

The network traffic monitoring utility is designed and composed of three major functions: 'individual host network status query', 'network profile traffic status query' and 'network profile maintenance'. The 'individual host network status query' function includes several subfunctions for users to retrieve the network and system status of the host running the utility. The 'network profile traffic status query' function uses a network profile to monitor the network traffic status of a star topology that conforms the Internet SNMP model. Users may learn the network reachability and network traffic status, and traffic rating. This profile traffic monitoring can be done either in real-time or long-term modes. The 'network profile maintenance' function provides a tool for user to maintain the network profile. The profile maintenance allows managers to reconfigure the network.

The network traffic status is an important indicator of the network's health. Understanding the network traffic situation, the network users can schedule the network applications by avoiding network congestions. The network applications become more efficient and effective. Additionally, the network traffic status can be used to diagnose the network. Through the network configuration management, network managers may improve the performance and throughput for network performance management.

In short, this utility provides a network traffic status query environment for network managers. These useful network traffic information are the basis for the network performance management: efficient performance and high throughput.

B. SUGGESTION FOR FURTHER RESEARCH

There are still several areas we need to discuss and improve this utility and network performance management:

1. Improvement to Users Interfaces

In this utility, users input the network traffic query parameters interactively (See Appendix A) in the ASCII codes. Graphical users interfaces would make the software much more user friendly. About the network traffic query output, although there are many network traffic information available for users, these output are all in text reports. If the output information is in graphics display, it may help users more easily understand the network traffic situation. It is possible to setup the multiple windows environment to query and display different network traffic situations simultaneously. This can help users to realize the network traffic situation more precisely and efficiently.

2. Portability Improvement

This utility uses several UNIX system commands to collect the network and system status. This would constrain the portability of this utility. It's possible to include the source code of these UNIX system commands into the utility source code. It may be desirable to write the programs having the same functions as these UNIX system commands. It will improve the portability of this utility.

3. Developing a Scheduling/Dispatching Algorithm for Network Performance Management

This utility provides some network traffic information. It shows only the network traffic situation, but it does not tell users how to schedule the network applications or reconfigure the network. This seems to be inadequate for users and network managers. If we could use the result of this utility as the input to a Scheduling/Dispatching algorithm, recommendation report for users can be generated. Users could use this recommendation report to schedule the network applications. The network managers could reallocate the network resources and configurations according to this recommendation report. This algorithm might help users to get better performance over the network. This would be a positive contribution in the network performance management.

APPENDIX A - A DIALOGUE OF NPS CAMPUS NETWORK TRAFFIC

MONITORING

This appendix intends to document the dialogue of traffic monitoring of using the NPS campus network. It is divided into three parts: 'network profile setup', 'profile traffic status', and 'individual network node traffic status measurement'.

A. NETWORK PROFILE SETUP

A network profile describes the network nodes at NPS campus, which is created by using nodes available on the NPS campus. Initially it is necessary to get all the names or addresses of these nodes. The 'Internet domain name server query' function can be used to obtain node names and addresses. When using this function, we need to set the default name server to the 'nps.navy.mil', the domain name of the NPS campus. Within the server query, one can retrieve detailed information by using the 'ls' command. The following two figures show the dialogues of NPS campus network nodes query.

```

<<<<< Network Traffic Measure Individual Host Query Menu >>>>>

1. Host Name and Address Query
2. Query Internet Domain Name Servers Utility
3. ECHO Data Packets - Ping Utility
4. Show Network Status Utility
5. I/O Statistics Report Utility
6. Virtual Memory Statistics Report Utility
7. UNIX Command Tools
8. File Transfer Measurement Tools
9. Exit

Please select one function (1-9) ==>>

```

Figure A.1 Individual Host Network Status Query Menu

In Figure A.2, one finds that many NPS network node information are retrieved from the same server. But the same network node may have several copies in the above dialogue. How can we check network nodes listed in the above dialogue? The 'host name and address query' function can validate the information of nodes by using the 'ls' command in the 'Internet domain name server query' function. Figure A.3 shows the dialogue.

```

*** Do you want to use the local host's name server
    instead of the default servers ==> (Y/N) y
*** Please input the symbolic name/internet address of server ==>
nps.navy.mil
!!! Type <?> or <help> for interactive query help !!!
!!! Type <exit> to exit the Internet Domain Name Servers Utility !!!
Default Server : nps.navy.mil
Address : 131.120.254.52
> ls nps.navy.mil
[nps.navy.mil]
Host or domain name

```

Figure A.2 A Dialogue of NPS Campus Network Nodes Query

```

nps          server = nps.navy.mil
nps          131.120.254.52
nps          131.120.254.52
loghost      127.0.0.1
localhost    127.0.0.1
aa           server = nps.navy.mil
nps          131.120.254.52
ece          server = srv2-gw.ece.nps.navy.mil
srv2-gw      131.120.254.20
ece          131.120.20.2
oc           server = oc.nps.navy.mil
oc           131.120.60.11
oc           131.120.60.11
ns           server = nps.navy.mil
nps          131.120.254.52
mis          131.120.80.1
or           server = nps.navy.mil
nps          131.120.254.52
as           server = is1.as.nps.navy.mil
is1          131.120.39.2
cc           server = galaxy.cc.nps.navy.mil
galaxy       131.120.50.55
cc           131.120.50.50
alpha        131.120.254.55
beta         131.120.254.56
pegasus      131.120.57.6
physics      server = nps.navy.mil
nps          131.120.254.52
cs           server = taurus.cs.nps.navy.mil
taurus       131.120.254.1
sagan        131.120.254.58
rover        131.120.254.8
seawolf      131.120.81.4
gamma        131.120.254.57
csrg         server = csrg.nps.navy.mil
csrg         131.120.29.1
csrg         131.120.29.1
mtry         131.120.57.11
met          server = killdeer.met.nps.navy.mil
killdeer     131.120.254.163
trac         131.120.57.2
math         server = nps.navy.mil
nps          131.120.254.52
me           server = nps.navy.mil
nps          131.120.254.52
tpdc         server = nps.navy.mil
nps          131.120.254.52
tpdc         131.120.50.58
> exit

```

Figure A.2 continued

```

*** Please input the symbolic name/Internet address of server ==>
nps
*** 131.120.254.52 host information ***
official host name : nps.navy.mil
alias listing      :
address type       : 2
address length     : 4
Internet address   : 131.120.254.52
*** Do you want to query another host information (Y/N) y

*** Please input the symbolic name/Internet address of server ==>
ece
*** 131.120.20.2 host information ***
official host name : ece.nps.navy.mil
alias listing      :
address type       : 2
address length     : 4
Internet address   : 131.120.20.2
*** Do you want to query another host information (Y/N) y

*** Please input the symbolic name/Internet address of server ==>
mis
!!! the host name error : mis, please input the correct name. !!!
!!! Network host name/address is error !!!
*** Do you want to query another host information (Y/N) y

*** Please input the symbolic name/Internet address of server ==>
beta
!!! the host name error : beta, please input the correct name. !!!
!!! Network host name/address is error !!!

```

Figure A.3 A dialogue of Host Name and Address Query

The 'host name and address query' function checks the symbolic names and IP addresses of the listed network nodes and finds there are only about ten network nodes' are correct information. Figure A.3 shows the dialogues. Having collected the correct information of the network nodes, we can create an NPS network profile by using the network profile maintenance function.

```
<<<<< Network Profile Maintain >>>>>

1. Add a network profile
2. Delete a network profile
3. Update a network profile
4. Exit

Please select one function (1 - 4) ==>> 1
```

Figure A.4 A Menu of Network Profile Maintain function

We can select the 'add a network profile' subfunction to create the network profile (see Figure A.4). After selecting the 'add a network profile' subfunction, it is necessary to specify a network profile name and confirm it. Then it is allowed to input the correct symbolic names or IP addresses of NPS network nodes. To exit the profile creation, one can use either the 'exit' or 'quit' command. Figure A.5 shows the dialogue of a new network profile creation.

```

*** Please input a file name for a new network profile ===>
nps
!!! Check profile name, please wait !!!
*** Do you want to create this network profile : nps (Y/N) y
!!! nps network profile create is proceeding !!!
*** Please input the symbolic name/IP address of a network node ***
!!! Please type <exit> or <quit> to end network node input !!!

add profile record > nps
!!! nps is written into the nps file !!!

add profile record > ece
!!! ece is written into the nps file !!!

add profile record > oc
!!! oc is written into the nps file !!!

add profile record > cc
!!! cc is written into the nps file !!!

add profile record > pegasus
!!! pegasus is written into the nps file !!!

add profile record > cs
!!! cs is written into the nps file !!!

add profile record > taurus
!!! taurus is written into the nps file !!!

add profile record > rover
!!! rover is written into the nps file !!!

add profile record > csrg
!!! csrg is written into the nps file !!!

add profile record > 131.120.57.2
!!! 131.120.57.2 is written into the nps file !!!

add profile record > exit

*** Do you want to add another network profile (Y/N) n

```

Figure A.5 A Dialogue of Add a Network Profile Command

B. PROFILE TRAFFIC STATUS

In this example, we will use the NPS campus network to measure its network traffic situations. The NPS campus network traffic can be monitored by the 'network profile

traffic status query' function. Network profile is a means to represent the underlying network configuration. This function will use the monitored traffic status to generate several network statistics. Reachability test shown in Figure A.6 on each node must be performed before starting actual traffic monitoring. In this case, the network profile used is 'nps' in the reachability test. The dialogue of profile selection is shown in Figure A.7.

<<<<< Network Profile Query >>>>>

1. Reachability Test
2. Traffic Statistics
3. Network Node Traffic Status Rating
4. Exit

Please select one function (1 - 4) ==>>>

Figure A.6 A Menu of Network Profile Traffic Status Query Function

```
*** Network profile listing ***  
  
1. ac  
2. nps  
3. test1  
4. test2  
5. test3  
  
Please select one, <6> to exit ==> 2  
  
*** Do you want to browse the content of this network profile : nps (Y/N) y  
  
nps  
ece  
oc  
cc  
pegasus  
cs  
taurus  
rover  
csrg  
131.120.57.2  
  
*** Do you really want to use this network profile : nps --> (Y/N) y
```

Figure A.7 A Dialogue of Network Profile Selection

Figure A.8 displays the reachability report of the NPS campus network. In this report, there is no ICMP ECHO_RESPONSE datagram packet received from the 'rover.nps.navy.mil' network node. It means this network node can not be reached.

*** Reachability Report ***

nps.navy.mil is alive
ece.nps.navy.mil is alive
oc.nps.navy.mil is alive
cc.nps.navy.mil is alive
pegasus.nps.navy.mil is alive
mailboy.cs.nps.navy.mil is alive
taurus.cs.nps.navy.mil is alive
no answer from rover.nps.navy.mil
csrg.nps.navy.mil is alive
131.120.57.2 is alive

Figure A.8 NPS Campus Network Nodes Reachability Test Report

Having tested the reachabilities of NPS network nodes, one may monitor one day time length of network traffic of NPS campus and sample hourly network traffic status in records. The 'traffic statistics' and 'network node traffic status rating' subfunctions collect the traffic status and generate the rating reports. The selection of these two subfunctions is shown in Figure A.6. First, the selection of the 'traffic statistics' subfunction will retrieve the NPS network traffic information. In the 'traffic statistics' subfunction shown in Figure A.9, we need to select the 'long-term network traffic monitoring' subfunction to monitor the NPS campus network traffic.

Monitoring the NPS campus network traffic requires the user's specification of a network profile, 'nps' shown Figure A.7. Besides, it is necessary set several relevant parameters to describe our scenario. See Figure A.10. A UNIX shell file can be used in the background for the monitoring task.

******* Network Traffic Query Mode Selection *******

1. real time network traffic query
2. long term network traffic monitoring
3. long term network traffic statistics report
4. Exit

Please select one function (1 - 4) ===>>

Figure A.9 A Menu of Traffic Statistics Subfunction

```

*** Please select the time unit of traffic monitoring ***
1. minute
2. hour
3. day
4. month(30 days)
5. year

Please select one time unit (1 - 5) ==> 2

*** How many hours do you want to log the network traffic ==> 1

*** Please select one as the one traffic sample time unit ***
1. minute
2. hour
3. day

Please select one time unit (1 - 3) ==> 1

*** How many minutes do you want to collect one sample
the network traffic status ==> 10

*** Please input the datagram packet size (10 ~ 1024)
for the per traffics status collection ==> 512

*** Please input the sampling number (5 ~ 100)
in per traffic status collection ==> 10

```

Figure A.10 A Dialogue of Long Term Network Traffic Parameters Setting

Once the NPS's network traffic records are collected, we can use the 'long-term network traffic statistic report' subfunction shown in Figure A.9 to generate the network traffic statistics report. In this subfunction, we can select the 'nps_1_day_by_per_1_hour' traffic status file as the source file to generate the NPS campus network traffic statistics report shown Figure A.11.

***** Long Term Traffic Statistics Report *****				
network node name : nps				
time	percent packets received	time round-trip minimum	time round-trip average	time round-trip maximum
Tue Feb 23 13:57:28 PST 1993	100.00	0	3	10
Tue Feb 23 15:00:51 PST 1993	100.00	0	0	10
Tue Feb 23 16:04:24 PST 1993	100.00	0	1	10
Tue Feb 23 17:07:57 PST 1993	100.00	0	0	10
Tue Feb 23 18:11:29 PST 1993	100.00	0	5	10
Tue Feb 23 19:15:00 PST 1993	100.00	0	0	0
Tue Feb 23 20:18:31 PST 1993	100.00	0	0	10
Tue Feb 23 21:22:02 PST 1993	100.00	0	1	10
Tue Feb 23 22:25:34 PST 1993	100.00	0	0	0
Tue Feb 23 23:23:29 PST 1993	100.00	0	1	10
Wed Feb 24 00:32:38 PST 1993	100.00	0	4	10
Wed Feb 24 01:36:10 PST 1993	100.00	0	3	10
Wed Feb 24 02:39:42 PST 1993	100.00	0	0	9
Wed Feb 24 03:43:15 PST 1993	100.00	0	10	20
Wed Feb 24 04:46:53 PST 1993	100.00	0	2	10
Wed Feb 24 05:50:25 PST 1993	100.00	0	1	10
Wed Feb 24 06:53:56 PST 1993	100.00	0	0	10
Wed Feb 24 07:57:27 PST 1993	100.00	0	2	10
Wed Feb 24 09:00:59 PST 1993	100.00	0	0	10
Wed Feb 24 10:04:31 PST 1993	100.00	0	1	20
Wed Feb 24 11:08:03 PST 1993	100.00	0	3	10
Wed Feb 24 12:11:37 PST 1993	100.00	0	3	10
Wed Feb 24 13:15:10 PST 1993	100.00	0	3	10
Wed Feb 24 14:18:43 PST 1993	100.00	0	0	10

Figure A.11 NPS Campus One Day Network Traffic Statistics Report

network node name : ece				
time	percent packets received	time round-trip minimum	time round-trip average	time round-trip maximum
Tue Feb 23 13:57:28 PST 1993	100.00	0	1	10
Tue Feb 23 15:00:51 PST 1993	100.00	0	0	10
Tue Feb 23 16:04:24 PST 1993	100.00	0	0	9
Tue Feb 23 17:07:57 PST 1993	100.00	0	0	0
Tue Feb 23 18:11:29 PST 1993	100.00	0	0	10
Tue Feb 23 19:15:00 PST 1993	100.00	0	0	10
Tue Feb 23 20:18:31 PST 1993	100.00	0	0	10
Tue Feb 23 21:22:02 PST 1993	100.00	0	0	10
Tue Feb 23 22:25:34 PST 1993	100.00	0	0	9
Tue Feb 23 23:23:29 PST 1993	100.00	0	1	10
Wed Feb 24 00:32:38 PST 1993	100.00	0	0	9
Wed Feb 24 01:36:10 PST 1993	100.00	0	2	10
Wed Feb 24 02:39:42 PST 1993	100.00	0	1	10
Wed Feb 24 03:43:15 PST 1993	100.00	0	6	20
Wed Feb 24 04:46:53 PST 1993	100.00	0	1	10
Wed Feb 24 05:50:25 PST 1993	100.00	0	0	10
Wed Feb 24 06:53:56 PST 1993	100.00	0	0	10
Wed Feb 24 07:57:27 PST 1993	100.00	0	0	10
Wed Feb 24 09:00:59 PST 1993	100.00	0	0	10
Wed Feb 24 10:04:31 PST 1993	100.00	0	0	10
Wed Feb 24 11:08:03 PST 1993	100.00	0	2	10
Wed Feb 24 12:11:37 PST 1993	100.00	0	3	20
Wed Feb 24 13:15:10 PST 1993	100.00	0	4	20
Wed Feb 24 14:18:43 PST 1993	100.00	0	1	10

Figure A.11 continued

network node name : oc				
time	percent packets received	time round-trip minimum	time round-trip average	time round-trip maximum
Tue Feb 23 13:57:28 PST 1993	*0.00	-	-	-
Tue Feb 23 15:00:51 PST 1993	100.00	0	5	10
Tue Feb 23 16:04:24 PST 1993	100.00	0	5	20
Tue Feb 23 17:07:57 PST 1993	100.00	0	4	10
Tue Feb 23 18:11:29 PST 1993	100.00	0	5	20
Tue Feb 23 19:15:00 PST 1993	100.00	0	2	19
Tue Feb 23 20:18:31 PST 1993	100.00	0	4	19
Tue Feb 23 21:22:02 PST 1993	100.00	0	8	10
Tue Feb 23 22:25:34 PST 1993	100.00	0	8	10
Tue Feb 23 23:23:29 PST 1993	100.00	0	5	10
Wed Feb 24 00:32:38 PST 1993	100.00	0	4	30
Wed Feb 24 01:36:10 PST 1993	100.00	0	7	10
Wed Feb 24 02:39:42 PST 1993	100.00	0	6	10
Wed Feb 24 03:43:15 PST 1993	100.00	0	9	20
Wed Feb 24 04:46:53 PST 1993	100.00	0	3	10
Wed Feb 24 05:50:25 PST 1993	100.00	0	2	10
Wed Feb 24 06:53:56 PST 1993	100.00	0	9	10
Wed Feb 24 07:57:27 PST 1993	100.00	0	10	20
Wed Feb 24 09:00:59 PST 1993	100.00	0	8	10
Wed Feb 24 10:04:31 PST 1993	100.00	9	10	20
Wed Feb 24 11:08:03 PST 1993	100.00	9	10	19
Wed Feb 24 12:11:37 PST 1993	100.00	10	10	10
Wed Feb 24 13:15:10 PST 1993	100.00	0	8	10
Wed Feb 24 14:18:43 PST 1993	100.00	0	2	10

Figure A.11 continued

network node name : cc				
time	percent packets received	time round-trip minimum	time round-trip average	time round-trip maximum
Tue Feb 23 13:57:28 PST 1993	100.00	20	34	140
Tue Feb 23 15:00:51 PST 1993	100.00	20	32	80
Tue Feb 23 16:04:24 PST 1993	100.00	29	29	30
Tue Feb 23 17:07:57 PST 1993	100.00	29	30	40
Tue Feb 23 18:11:29 PST 1993	100.00	20	31	40
Tue Feb 23 19:15:00 PST 1993	100.00	30	213	1380
Tue Feb 23 20:18:31 PST 1993	100.00	30	30	39
Tue Feb 23 21:22:02 PST 1993	100.00	30	30	40
Tue Feb 23 22:25:34 PST 1993	100.00	30	31	40
Tue Feb 23 23:23:29 PST 1993	100.00	30	30	40
Wed Feb 24 00:32:38 PST 1993	100.00	30	30	30
Wed Feb 24 01:36:10 PST 1993	100.00	30	31	50
Wed Feb 24 02:39:42 PST 1993	100.00	29	32	50
Wed Feb 24 03:43:15 PST 1993	100.00	30	32	40
Wed Feb 24 04:46:53 PST 1993	100.00	29	29	30
Wed Feb 24 05:50:25 PST 1993	100.00	30	30	30
Wed Feb 24 06:53:56 PST 1993	100.00	30	30	40
Wed Feb 24 07:57:27 PST 1993	100.00	30	31	50
Wed Feb 24 09:00:59 PST 1993	100.00	20	31	40
Wed Feb 24 10:04:31 PST 1993	100.00	30	46	140
Wed Feb 24 11:08:03 PST 1993	100.00	30	36	90
Wed Feb 24 12:11:37 PST 1993	100.00	20	32	40
Wed Feb 24 13:15:10 PST 1993	100.00	30	32	40
Wed Feb 24 14:18:43 PST 1993	100.00	30	33	40

Figure A.11 continued

network node name : pegasus				
time	percent packets received	time round-trip minimum	time round-trip average	time round-trip maximum
Tue Feb 23 13:57:28 PST 1993	100.00	0	0	10
Tue Feb 23 15:00:51 PST 1993	100.00	0	0	9
Tue Feb 23 16:04:24 PST 1993	100.00	0	0	0
Tue Feb 23 17:07:57 PST 1993	100.00	0	0	9
Tue Feb 23 18:11:29 PST 1993	100.00	0	0	9
Tue Feb 23 19:15:00 PST 1993	100.00	0	0	0
Tue Feb 23 20:18:31 PST 1993	100.00	0	0	10
Tue Feb 23 21:22:02 PST 1993	100.00	0	2	10
Tue Feb 23 22:25:34 PST 1993	100.00	0	1	20
Tue Feb 23 23:23:29 PST 1993	100.00	0	0	10
Wed Feb 24 00:32:38 PST 1993	100.00	0	0	10
Wed Feb 24 01:36:10 PST 1993	100.00	0	0	9
Wed Feb 24 02:39:42 PST 1993	100.00	0	0	0
Wed Feb 24 03:43:15 PST 1993	100.00	0	0	10
Wed Feb 24 04:46:53 PST 1993	100.00	0	0	0
Wed Feb 24 05:50:25 PST 1993	100.00	0	0	9
Wed Feb 24 06:53:56 PST 1993	100.00	0	0	9
Wed Feb 24 07:57:27 PST 1993	100.00	0	0	10
Wed Feb 24 09:00:59 PST 1993	100.00	0	1	10
Wed Feb 24 10:04:31 PST 1993	100.00	0	0	10
Wed Feb 24 11:08:03 PST 1993	100.00	0	0	0
Wed Feb 24 12:11:37 PST 1993	100.00	0	0	0
Wed Feb 24 13:15:10 PST 1993	100.00	0	0	0
Wed Feb 24 14:18:43 PST 1993	100.00	0	0	10

Figure A.11 continued

network node name : cs				
time	percent packets received	time round-trip minimum	time round-trip average	time round-trip maximum
Tue Feb 23 13:57:28 PST 1993	100.00	0	1	10
Tue Feb 23 15:00:51 PST 1993	100.00	0	1	10
Tue Feb 23 16:04:24 PST 1993	100.00	0	2	20
Tue Feb 23 17:07:57 PST 1993	100.00	0	0	10
Tue Feb 23 18:11:29 PST 1993	100.00	0	1	10
Tue Feb 23 19:15:00 PST 1993	100.00	0	0	0
Tue Feb 23 20:18:31 PST 1993	100.00	0	0	10
Tue Feb 23 21:22:02 PST 1993	100.00	0	1	10
Tue Feb 23 22:25:34 PST 1993	100.00	0	0	10
Tue Feb 23 23:23:29 PST 1993	100.00	0	0	10
Wed Feb 24 00:32:38 PST 1993	100.00	0	0	9
Wed Feb 24 01:36:10 PST 1993	100.00	0	0	10
Wed Feb 24 02:39:42 PST 1993	100.00	0	0	10
Wed Feb 24 03:43:15 PST 1993	100.00	0	1	20
Wed Feb 24 04:46:53 PST 1993	100.00	0	0	10
Wed Feb 24 05:50:25 PST 1993	100.00	0	1	10
Wed Feb 24 06:53:56 PST 1993	100.00	0	0	10
Wed Feb 24 07:57:27 PST 1993	100.00	0	0	10
Wed Feb 24 09:00:59 PST 1993	100.00	0	0	10
Wed Feb 24 10:04:31 PST 1993	100.00	0	1	10
Wed Feb 24 11:08:03 PST 1993	100.00	0	0	10
Wed Feb 24 12:11:37 PST 1993	100.00	0	0	10
Wed Feb 24 13:15:10 PST 1993	100.00	0	0	10
Wed Feb 24 14:18:43 PST 1993	100.00	0	2	30

Figure A.11 continued

network node name : taurus				
time	percent packets received	time round-trip minimum	time round-trip average	time round-trip maximum
Tue Feb 23 13:57:28 PST 1993	100.00	0	0	0
Tue Feb 23 15:00:51 PST 1993	100.00	0	0	10
Tue Feb 23 16:04:24 PST 1993	100.00	0	0	10
Tue Feb 23 17:07:57 PST 1993	100.00	0	0	0
Tue Feb 23 18:11:29 PST 1993	100.00	0	0	0
Tue Feb 23 19:15:00 PST 1993	100.00	0	0	0
Tue Feb 23 20:18:31 PST 1993	100.00	0	1	10
Tue Feb 23 21:22:02 PST 1993	100.00	0	0	0
Tue Feb 23 22:25:34 PST 1993	100.00	0	1	10
Tue Feb 23 23:23:29 PST 1993	100.00	0	0	0
Wed Feb 24 00:32:38 PST 1993	100.00	0	0	10
Wed Feb 24 01:36:10 PST 1993	100.00	0	0	10
Wed Feb 24 02:39:42 PST 1993	100.00	0	0	0
Wed Feb 24 03:43:15 PST 1993	100.00	0	3	50
Wed Feb 24 04:46:53 PST 1993	100.00	0	0	0
Wed Feb 24 05:50:25 PST 1993	100.00	0	0	10
Wed Feb 24 06:53:56 PST 1993	100.00	0	0	10
Wed Feb 24 07:57:27 PST 1993	100.00	0	0	0
Wed Feb 24 09:00:59 PST 1993	100.00	0	0	10
Wed Feb 24 10:04:31 PST 1993	100.00	0	3	60
Wed Feb 24 11:08:03 PST 1993	100.00	0	0	0
Wed Feb 24 12:11:37 PST 1993	100.00	0	0	10
Wed Feb 24 13:15:10 PST 1993	100.00	0	1	10
Wed Feb 24 14:18:43 PST 1993	100.00	0	0	10

Figure A.11 continued

network node name : rover				
time	percent packets received	time round-trip minimum	time round-trip average	time round-trip maximum
Tue Feb 23 13:57:28 PST 1993	*0.00	-	-	-
Tue Feb 23 15:00:51 PST 1993	*0.00	-	-	-
Tue Feb 23 16:04:24 PST 1993	*0.00	-	-	-
Tue Feb 23 17:07:57 PST 1993	*0.00	-	-	-
Tue Feb 23 18:11:29 PST 1993	*0.00	-	-	-
Tue Feb 23 19:15:00 PST 1993	*0.00	-	-	-
Tue Feb 23 20:18:31 PST 1993	*0.00	-	-	-
Tue Feb 23 21:22:02 PST 1993	*0.00	-	-	-
Tue Feb 23 22:25:34 PST 1993	*0.00	-	-	-
Tue Feb 23 23:23:29 PST 1993	*0.00	-	-	-
Wed Feb 24 00:32:38 PST 1993	*0.00	-	-	-
Wed Feb 24 01:36:10 PST 1993	*0.00	-	-	-
Wed Feb 24 02:39:42 PST 1993	*0.00	-	-	-
Wed Feb 24 03:43:15 PST 1993	*0.00	-	-	-
Wed Feb 24 04:46:53 PST 1993	*0.00	-	-	-
Wed Feb 24 05:50:25 PST 1993	*0.00	-	-	-
Wed Feb 24 06:53:56 PST 1993	*0.00	-	-	-
Wed Feb 24 07:57:27 PST 1993	*0.00	-	-	-
Wed Feb 24 09:00:59 PST 1993	*0.00	-	-	-
Wed Feb 24 10:04:31 PST 1993	*0.00	-	-	-
Wed Feb 24 11:08:03 PST 1993	*0.00	-	-	-
Wed Feb 24 12:11:37 PST 1993	*0.00	-	-	-
Wed Feb 24 13:15:10 PST 1993	*0.00	-	-	-
Wed Feb 24 14:18:43 PST 1993	*0.00	-	-	-

Figure A.11 continued

network node name : csrg				
time	percent packets received	time round-trip minimum	time round-trip average	time round-trip maximum
Tue Feb 23 13:57:28 PST 1993	100.00	0	1	10
Tue Feb 23 15:00:51 PST 1993	100.00	0	3	20
Tue Feb 23 16:04:24 PST 1993	100.00	0	0	9
Tue Feb 23 17:07:57 PST 1993	100.00	0	0	0
Tue Feb 23 18:11:29 PST 1993	100.00	0	0	10
Tue Feb 23 19:15:00 PST 1993	100.00	0	0	10
Tue Feb 23 20:18:31 PST 1993	100.00	0	0	10
Tue Feb 23 21:22:02 PST 1993	100.00	0	1	20
Tue Feb 23 22:25:34 PST 1993	100.00	0	1	10
Tue Feb 23 23:23:29 PST 1993	100.00	0	0	10
Wed Feb 24 00:32:38 PST 1993	100.00	0	0	10
Wed Feb 24 01:36:10 PST 1993	100.00	0	1	10
Wed Feb 24 02:39:42 PST 1993	100.00	0	1	10
Wed Feb 24 03:43:15 PST 1993	100.00	0	1	10
Wed Feb 24 04:46:53 PST 1993	100.00	0	0	10
Wed Feb 24 05:50:25 PST 1993	100.00	0	0	10
Wed Feb 24 06:53:56 PST 1993	100.00	0	1	10
Wed Feb 24 07:57:27 PST 1993	100.00	0	2	10
Wed Feb 24 09:00:59 PST 1993	100.00	0	0	10
Wed Feb 24 10:04:31 PST 1993	100.00	0	0	9
Wed Feb 24 11:08:03 PST 1993	100.00	0	0	10
Wed Feb 24 12:11:37 PST 1993	100.00	0	1	10
Wed Feb 24 13:15:10 PST 1993	100.00	0	2	20
Wed Feb 24 14:18:43 PST 1993	100.00	0	2	10

Figure A.11 continued

network node name : 131.120.57.2				
time	percent packets received	time round-trip minimum	time round-trip average	time round-trip maximum
Tue Feb 23 13:57:28 PST 1993	100.00	10	10	10
Tue Feb 23 15:00:51 PST 1993	100.00	10	13	80
Tue Feb 23 16:04:24 PST 1993	100.00	9	12	60
Tue Feb 23 17:07:57 PST 1993	100.00	10	13	60
Tue Feb 23 18:11:29 PST 1993	100.00	10	10	20
Tue Feb 23 19:15:00 PST 1993	100.00	10	13	70
Tue Feb 23 20:18:31 PST 1993	100.00	10	11	20
Tue Feb 23 21:22:02 PST 1993	100.00	10	11	20
Tue Feb 23 22:25:34 PST 1993	100.00	10	17	130
Tue Feb 23 23:23:29 PST 1993	100.00	10	11	30
Wed Feb 24 00:32:38 PST 1993	100.00	9	11	40
Wed Feb 24 01:36:10 PST 1993	100.00	10	10	20
Wed Feb 24 02:39:42 PST 1993	100.00	10	10	20
Wed Feb 24 03:43:15 PST 1993	100.00	10	10	20
Wed Feb 24 04:46:53 PST 1993	100.00	10	10	20
Wed Feb 24 05:50:25 PST 1993	100.00	10	11	40
Wed Feb 24 06:53:56 PST 1993	100.00	10	10	10
Wed Feb 24 07:57:27 PST 1993	100.00	10	13	70
Wed Feb 24 09:00:59 PST 1993	100.00	10	12	60
Wed Feb 24 10:04:31 PST 1993	100.00	10	32	450
Wed Feb 24 11:08:03 PST 1993	100.00	10	16	70
Wed Feb 24 12:11:37 PST 1993	100.00	10	12	60
Wed Feb 24 13:15:10 PST 1993	100.00	10	16	140
Wed Feb 24 14:18:43 PST 1993	100.00	10	13	80

Figure A.11 continued

The 'network node traffic status rating' subfunction shown in Figure A.6 can generate the 'long-term network traffic status rating report'. In this subfunction, we can follow the steps used in the 'traffic statistics' subfunction: select the 'long-term network node rating traffic monitoring' subfunction in Figure A.12 and then set the parameters for NPS campus network traffic monitoring in Figure A.10. Finishing the NPS campus network traffic status collection, selecting the 'long-term network node traffic rating statistic report' subfunction in Figure A.12 to generate the report shown in Figure A.13 is allowed.

***** Network Traffic Rating Query Mode Selection *****

1. real time network node rating traffic query
2. long term network node rating traffic monitoring
3. long term network node traffic rating statistics report
4. Exit

Please select one function (1 - 4) ==>>

Figure A.12 A Menu of Network Node Traffic Status Rating Subfunction

***** Long Term Network node traffic status rating report *****						
network node name : pegasus						
time	rating	received packets percent	round-trip time mean-value	round-trip time variance	round-trip time std-deviation	
Tue Feb23 13:59:10PST1993	1	100.00	0.000	0.000	0.000	
Tue Feb23 15:02:49PST1993	1	100.00	0.500	5.000	2.236	
Tue Feb23 16:06:18PST1993	2	100.00	0.500	5.000	2.236	
Tue Feb23 17:09:46PST1993	3	100.00	1.000	9.474	3.078	
Tue Feb23 18:13:15PST1993	2	100.00	0.500	5.000	2.236	
Tue Feb23 19:16:42PST1993	1	100.00	0.000	0.000	0.000	
Tue Feb23 20:20:10PST1993	4	100.00	1.500	13.421	3.663	
Tue Feb23 21:23:33PST1993	3	100.00	1.000	9.474	3.078	
Tue Feb23 22:27:07PST1993	1	100.00	0.000	0.000	0.000	
Tue Feb23 23:30:36PST1993	2	100.00	0.500	5.000	2.236	
Wed Feb24 00:34:04PST1993	1	100.00	0.000	0.000	0.000	
Wed Feb24 01:37:32PST1993	1	100.00	0.000	0.000	0.000	
Wed Feb24 02:41:00PST1993	2	100.00	0.450	4.050	2.012	
Wed Feb24 03:44:28PST1993	3	100.00	1.500	13.421	3.663	
Wed Feb24 04:47:57PST1993	1	100.00	0.000	0.000	0.000	
Wed Feb24 05:51:25PST1993	2	100.00	0.500	5.000	2.236	
Wed Feb24 06:54:53PST1993	1	100.00	0.000	0.000	0.000	
Wed Feb24 07:58:21PST1993	1	100.00	0.000	0.000	0.000	
Wed Feb24 09:01:48PST1993	2	100.00	0.450	4.050	2.012	
Wed Feb24 10:05:16PST1993	3	100.00	0.500	5.000	2.236	
Wed Feb24 11:08:44PST1993	2	100.00	0.500	5.000	2.236	
Wed Feb24 12:12:14PST1993	2	100.00	0.500	5.000	2.236	
Wed Feb24 13:15:43PST1993	1	100.00	0.000	0.000	0.000	
Wed Feb24 14:19:13PST1993	3	100.00	1.000	9.474	3.078	

Figure A.13 NPS Campus One Day Network Nodes Traffic Rating Report

network node name : cs					
time	rating	received packets percent	round-trip time mean-value	round-trip time variance	round-trip time std-deviation
Tue Feb23 13:59:10PST1993	1	100.00	0.000	0.000	0.000
Tue Feb23 15:02:49PST1993	4	100.00	2.500	19.737	4.443
Tue Feb23 16:06:18PST1993	3	100.00	1.000	9.474	3.078
Tue Feb23 17:09:46PST1993	4	100.00	1.500	23.947	4.894
Tue Feb23 18:13:15PST1993	1	100.00	0.000	0.000	0.000
Tue Feb23 19:16:42PST1993	6	100.00	1.000	20.000	4.472
Tue Feb23 20:20:10PST1993	3	100.00	1.000	9.474	3.078
Tue Feb23 21:23:38PST1993	3	100.00	1.000	9.474	3.078
Tue Feb23 22:27:07PST1993	5	100.00	2.000	37.895	6.156
Tue Feb23 23:30:36PST1993	2	100.00	0.500	5.000	2.236
Wed Feb24 00:34:04PST1993	4	100.00	2.000	16.842	4.104
Wed Feb24 01:37:32PST1993	3	100.00	0.500	5.000	2.236
Wed Feb24 02:41:00PST1993	4	100.00	2.000	8.576	2.929
Wed Feb24 03:44:28PST1993	1	100.00	0.500	5.000	2.236
Wed Feb24 04:47:57PST1993	4	100.00	0.950	9.474	3.078
Wed Feb24 05:51:25PST1993	2	100.00	0.500	5.000	2.236
Wed Feb24 06:54:53PST1993	4	100.00	1.000	9.474	3.078
Wed Feb24 07:58:21PST1993	3	100.00	0.500	5.000	2.236
Wed Feb24 09:01:48PST1993	1	100.00	0.000	0.000	0.000
Wed Feb24 10:05:16PST1993	3	100.00	0.500	5.000	2.236
Wed Feb24 11:08:44PST1993	2	100.00	0.500	5.000	2.236
Wed Feb24 12:12:14PST1993	7	100.00	4.000	35.789	5.982
Wed Feb24 13:15:43PST1993	4	100.00	1.450	12.576	3.546
Wed Feb24 14:19:13PST1993	1	100.00	0.000	0.000	0.000

Figure A.13 continued

network node name : taurus					
time	rating	received packets percent	round-trip time mean-value	round-trip time variance	round-trip time std-deviation
Tue Feb23 13:59:10PST1993	2	100.00	0.450	4.050	2.012
Tue Feb23 15:02:49PST1993	3	100.00	2.000	16.842	4.104
Tue Feb23 16:06:18PST1993	1	100.00	0.000	0.000	0.000
Tue Feb23 17:09:46PST1993	6	100.00	2.500	30.263	5.501
Tue Feb23 18:13:15PST1993	1	100.00	0.000	0.000	0.000
Tue Feb23 19:16:42PST1993	1	100.00	0.000	0.000	0.000
Tue Feb23 20:20:10PST1993	2	100.00	0.500	5.000	2.263
Tue Feb23 21:23:38PST1993	3	100.00	1.000	9.474	3.078
Tue Feb23 22:27:07PST1993	2	100.00	0.450	4.050	2.012
Tue Feb23 23:30:36PST1993	1	100.00	0.000	0.000	0.000
Wed Feb24 00:34:04PST1993	1	100.00	0.000	0.000	0.000
Wed Feb24 01:37:32PST1993	3	100.00	0.500	5.000	2.236
Wed Feb24 02:41:00PST1993	1	100.00	0.000	0.000	0.000
Wed Feb24 03:44:28PST1993	1	100.00	0.500	5.000	2.236
Wed Feb24 04:47:57PST1993	1	100.00	0.000	0.000	0.000
Wed Feb24 05:51:25PST1993	1	100.00	0.000	0.000	0.000
Wed Feb24 06:54:53PST1993	3	100.00	0.500	5.000	2.236
Wed Feb24 07:58:21PST1993	6	100.00	2.000	27.386	5.231
Wed Feb24 09:01:48PST1993	1	100.00	0.000	0.000	0.000
Wed Feb24 10:05:16PST1993	2	100.00	0.450	4.050	2.012
Wed Feb24 11:08:44PST1993	3	100.00	0.900	7.674	2.770
Wed Feb24 12:12:14PST1993	4	100.00	0.950	18.050	4.249
Wed Feb24 13:15:43PST1993	2	100.00	0.500	5.000	2.236
Wed Feb24 14:19:13PST1993	2	100.00	0.500	5.000	2.236

Figure A.13 continued

network node name : csrg					
time	rating	received packets percent	round-trip time mean-value	round-trip time variance	round-trip time std-deviation
Tue Feb23 13:59:10PST1993	3	100.00	0.500	5.000	2.236
Tue Feb23 15:02:49PST1993	5	100.00	3.000	22.105	4.702
Tue Feb23 16:06:18PST1993	2	100.00	0.500	5.000	2.236
Tue Feb23 17:09:46PST1993	2	100.00	0.500	5.000	2.236
Tue Feb23 18:13:15PST1993	1	100.00	0.000	0.000	0.000
Tue Feb23 19:16:42PST1993	2	100.00	0.500	5.000	2.236
Tue Feb23 20:20:10PST1993	1	100.00	0.000	0.000	0.000
Tue Feb23 21:23:38PST1993	2	100.00	0.500	5.000	2.236
Tue Feb23 22:27:07PST1993	3	100.00	1.000	9.474	3.078
Tue Feb23 23:30:36PST1993	3	100.00	1.000	9.474	3.078
Wed Feb24 00:34:04PST1993	2	100.00	0.500	5.000	2.236
Wed Feb24 01:37:32PST1993	2	100.00	0.450	4.050	2.012
Wed Feb24 02:41:00PST1993	1	100.00	0.000	0.000	0.000
Wed Feb24 03:44:28PST1993	2	100.00	1.000	9.474	3.078
Wed Feb24 04:47:57PST1993	2	100.00	0.450	4.050	2.012
Wed Feb24 05:51:25PST1993	2	100.00	0.500	5.000	2.236
Wed Feb24 06:54:53PST1993	1	100.00	0.000	0.000	0.000
Wed Feb24 07:58:21PST1993	2	100.00	0.450	4.050	2.012
Wed Feb24 09:01:48PST1993	6	100.00	2.950	31.945	5.652
Wed Feb24 10:05:16PST1993	4	100.00	2.000	16.842	4.104
Wed Feb24 11:08:44PST1993	6	100.00	1.500	13.421	3.663
Wed Feb24 12:12:14PST1993	3	100.00	1.500	13.421	3.663
Wed Feb24 13:15:43PST1993	6	100.00	2.500	19.737	4.443
Wed Feb24 14:19:13PST1993	6	100.00	6.500	402.895	20.072

Figure A.13 continued

network node name : ece						
time	rating	received packets percent	round-trip time mean-value	round-trip time variance	round-trip time std-deviation	
Tue Feb23 13:59:10PST1993	4	100.00	1.500	13.421	3.663	
Tue Feb23 15:02:49PST1993	2	100.00	1.500	13.421	3.663	
Tue Feb23 16:06:18PST1993	3	100.00	1.000	9.474	3.078	
Tue Feb23 17:09:46PST1993	2	100.00	0.500	5.000	2.236	
Tue Feb23 18:13:15PST1993	2	100.00	0.500	5.000	2.236	
Tue Feb23 19:16:42PST1993	3	100.00	1.000	9.474	3.078	
Tue Feb23 20:20:10PST1993	3	100.00	1.000	9.474	3.078	
Tue Feb23 21:23:38PST1993	2	100.00	0.500	5.000	2.236	
Tue Feb23 22:27:07PST1993	3	100.00	1.000	9.474	3.078	
Tue Feb23 23:30:36PST1993	4	100.00	1.500	13.421	3.663	
Wed Feb24 00:34:04PST1993	3	100.00	1.500	13.421	3.663	
Wed Feb24 01:37:32PST1993	3	100.00	0.500	5.000	2.236	
Wed Feb24 02:41:00PST1993	5	100.00	1.450	12.576	3.546	
Wed Feb24 03:44:28PST1993	5	100.00	4.000	25.263	5.026	
Wed Feb24 04:47:57PST1993	3	100.00	0.500	5.000	2.236	
Wed Feb24 05:51:25PST1993	3	100.00	1.000	9.474	3.078	
Wed Feb24 06:54:53PST1993	4	100.00	1.000	9.474	3.078	
Wed Feb24 07:58:21PST1993	7	100.00	5.000	68.421	8.272	
Wed Feb24 09:01:48PST1993	3	100.00	0.950	8.576	2.929	
Wed Feb24 10:05:16PST1993	4	100.00	2.000	16.842	4.104	
Wed Feb24 11:08:44PST1993	4	100.00	1.000	9.474	3.078	
Wed Feb24 12:12:14PST1993	5	100.00	2.450	18.997	4.359	
Wed Feb24 13:15:43PST1993	7	100.00	2.000	27.368	5.231	
Wed Feb24 14:19:13PST1993	3	100.00	1.000	9.474	9.474	

Figure A.13 continued

network node name : nps					
time	rating	received packets percent	round-trip time mean-value	round-trip time variance	round-trip time std-deviation
Tue Feb23 13:59:10PST1993	5	100.00	6.500	23.947	4.894
Tue Feb23 15:02:49PST1993	4	100.00	2.500	19.737	4.443
Tue Feb23 16:06:18PST1993	5	100.00	1.950	16.050	4.006
Tue Feb23 17:09:46PST1993	5	100.00	2.450	29.524	5.434
Tue Feb23 18:13:15PST1993	2	100.00	0.500	5.000	2.236
Tue Feb23 19:16:42PST1993	5	100.00	1.500	13.421	3.663
Tue Feb23 20:20:10PST1993	5	100.00	2.450	18.997	4.359
Tue Feb23 21:23:38PST1993	4	100.00	1.450	12.576	3.546
Tue Feb23 22:27:07PST1993	4	100.00	2.000	16.842	4.014
Tue Feb23 23:30:36PST1993	5	100.00	1.950	16.050	4.006
Wed Feb24 00:34:04PST1993	3	100.00	1.500	13.421	3.663
Wed Feb24 01:37:32PST1993	4	100.00	1.000	9.474	3.078
Wed Feb24 02:41:00PST1993	7	100.00	3.500	23.947	4.894
Wed Feb24 03:44:28PST1993	4	100.00	2.500	19.737	4.443
Wed Feb24 04:47:57PST1993	4	100.00	1.000	9.474	3.078
Wed Feb24 05:51:25PST1993	2	100.00	0.500	5.000	2.236
Wed Feb24 06:54:53PST1993	5	100.00	5.000	26.316	5.130
Wed Feb24 07:58:21PST1993	5	100.00	4.000	25.263	5.026
Wed Feb24 09:01:48PST1993	4	100.00	1.000	9.474	3.078
Wed Feb24 10:05:16PST1993	3	100.00	0.500	5.000	2.236
Wed Feb24 11:08:44PST1993	5	100.00	1.400	11.726	3.424
Wed Feb24 12:12:14PST1993	9	100.00	8.500	87.105	9.333
Wed Feb24 13:15:43PST1993	5	100.00	1.500	13.421	3.663
Wed Feb24 14:19:13PST1993	4	100.00	1.950	16.050	4.006

Figure A.13 continued

network node name : 131.120.57.2					
time	rating	received packets percent	round-trip time mean-value	round-trip time variance	round-trip time std-deviation
Tue Feb23 13:59:10PST1993	6	100.00	13.500	45.000	6.708
Tue Feb23 15:02:49PST1993	1	100.00	10.500	5.000	2.236
Tue Feb23 16:06:18PST1993	1	100.00	10.000	0.000	0.000
Tue Feb23 17:09:46PST1993	1	100.00	10.000	0.000	0.000
Tue Feb23 18:13:15PST1993	2	100.00	10.500	5.000	2.236
Tue Feb23 19:16:42PST1993	1	100.00	10.000	0.000	0.000
Tue Feb23 20:20:10PST1993	1	100.00	10.000	0.000	0.000
Tue Feb23 21:23:38PST1993	2	100.00	10.500	5.000	2.236
Tue Feb23 22:27:07PST1993	1	100.00	10.000	0.000	0.000
Tue Feb23 23:30:36PST1993	2	100.00	10.500	5.000	2.236
Wed Feb24 00:34:04PST1993	1	100.00	10.000	0.000	0.000
Wed Feb24 01:37:32PST1993	1	100.00	10.000	0.000	0.000
Wed Feb24 02:41:00PST1993	1	100.00	10.000	0.000	0.000
Wed Feb24 03:44:28PST1993	2	100.00	11.000	9.474	3.078
Wed Feb24 04:47:57PST1993	1	100.00	10.000	0.000	0.000
Wed Feb24 05:51:25PST1993	1	100.00	10.000	0.000	0.000
Wed Feb24 06:54:53PST1993	4	100.00	11.000	9.474	3.078
Wed Feb24 07:58:21PST1993	1	100.00	10.000	0.000	0.000
Wed Feb24 09:01:48PST1993	1	100.00	10.000	0.000	0.000
Wed Feb24 10:05:16PST1993	1	100.00	10.000	0.000	0.000
Wed Feb24 11:08:44PST1993	1	100.00	10.000	0.000	0.000
Wed Feb24 12:12:14PST1993	1	100.00	10.000	0.000	0.000
Wed Feb24 13:15:43PST1993	3	100.00	10.400	5.200	2.280
Wed Feb24 14:19:13PST1993	1	100.00	10.000	0.000	0.000

Figure A.13 continued

network node name : cc						
time	rating	received packets percent	round-trip time mean-value	round-trip time variance	round-trip time std-deviation	
Tue Feb23 13:59:10PST1993	7	100.00	159.500523	10.262	228.714	
Tue Feb23 15:02:49PST1993	1	100.00	29.500	5.000	2.236	
Tue Feb23 16:06:18PST1993	4	100.00	31.500	13.421	3.663	
Tue Feb23 17:09:46PST1993	3	100.00	31.000	9.474	3.078	
Tue Feb23 18:13:15PST1993	2	100.00	30.500	5.000	2.236	
Tue Feb23 19:16:42PST1993	4	100.00	30.000	10.526	3.244	
Tue Feb23 20:20:10PST1993	1	100.00	30.000	0.000	0.000	
Tue Feb23 21:23:38PST1993	2	100.00	29.500	5.000	2.236	
Tue Feb23 22:27:07PST1993	6	100.00	32.950	127.734	11.302	
Tue Feb23 23:30:36PST1993	7	100.00	33.500	87.105	9.333	
Wed Feb24 00:34:04PST1993	5	100.00	31.500	23.947	4.894	
Wed Feb24 01:37:32PST1993	6	100.00	32.000	27.368	5.231	
Wed Feb24 02:41:00PST1993	3	100.00	30.500	5.000	2.236	
Wed Feb24 03:44:28PST1993	6	100.00	34.000	56.842	7.539	
Wed Feb24 04:47:57PST1993	2	100.00	30.450	4.050	2.012	
Wed Feb24 05:51:25PST1993	3	100.00	31.000	9.474	3.078	
Wed Feb24 06:54:53PST1993	2	100.00	29.950	0.050	0.224	
Wed Feb24 07:58:21PST1993	4	100.00	30.900	9.994	3.144	
Wed Feb24 09:01:48PST1993	7	100.00	35.500	362.895	19.050	
Wed Feb24 10:05:16PST1993	5	100.00	61.500	3202.895	56.594	
Wed Feb24 11:08:44PST1993	7	100.00	41.500	560.789	23.681	
Wed Feb24 12:12:14PST1993	6	100.00	32.900	31.253	5.590	
Wed Feb24 13:15:43PST1993	8	100.00	32.450	82.155	9.064	
Wed Feb24 14:19:13PST1993	3	100.00	31.000	9.474	3.078	

Figure A.13 continued

network node name : oc					
time	rating	received packets percent	round-trip time mean-value	round-trip time variance	round-trip time std-deviation
Tue Feb23 13:59:10PST1993	8	*0.00	0.000	0.000	0.000
Tue Feb23 15:02:49PST1993	2	100.00	8.500	13.421	3.663
Tue Feb23 16:06:18PST1993	6	100.00	6.450	23.629	4.861
Tue Feb23 17:09:46PST1993	4	100.00	3.500	23.947	4.894
Tue Feb23 18:13:15PST1993	3	100.00	5.000	26.316	5.130
Tue Feb23 19:16:42PST1993	7	100.00	4.000	25.263	5.026
Tue Feb23 20:20:10PST1993	1	100.00	10.000	0.000	0.000
Tue Feb23 21:23:38PST1993	1	100.00	10.000	0.000	0.000
Tue Feb23 22:27:07PST1993	7	100.00	10.000	157.895	12.566
Tue Feb23 23:30:36PST1993	6	100.00	7.950	16.682	4.084
Wed Feb24 00:34:04PST1993	6	100.00	5.500	36.579	6.048
Wed Feb24 01:37:32PST1993	5	100.00	4.000	25.263	5.026
Wed Feb24 02:41:00PST1993	6	100.00	11.000	20.000	4.472
Wed Feb24 03:44:28PST1993	2	100.00	11.000	9.474	3.078
Wed Feb24 04:47:57PST1993	1	100.00	10.000	0.000	0.000
Wed Feb24 05:51:25PST1993	1	100.00	10.000	0.000	0.000
Wed Feb24 06:54:53PST1993	1	100.00	10.000	0.000	0.000
Wed Feb24 07:58:21PST1993	3	100.00	10.500	5.000	2.236
Wed Feb24 09:01:48PST1993	5	100.00	6.900	21.568	4.644
Wed Feb24 10:05:16PST1993	3	100.00	9.500	5.000	2.236
Wed Feb24 11:08:44PST1993	1	100.00	10.000	0.000	0.000
Wed Feb24 12:12:14PST1993	8	100.00	12.000	48.421	6.959
Wed Feb24 13:15:43PST1993	2	100.00	9.500	5.000	2.236
Wed Feb24 14:19:13PST1993	5	100.00	7.450	19.524	4.419

Figure A.13 continued

network node name : rover					
time	rating	received packets percent	round-trip time mean-value	round-trip time variance	round-trip time std-deviation
Tue Feb23 13:59:10PST1993	8	*0.00	0.000	0.000	0.000
Tue Feb23 15:02:49PST1993	6	*0.00	0.000	0.000	0.000
Tue Feb23 16:06:18PST1993	7	*0.00	0.000	0.000	0.000
Tue Feb23 17:09:46PST1993	7	*0.00	0.000	0.000	0.000
Tue Feb23 18:13:15PST1993	4	*0.00	0.000	0.000	0.000
Tue Feb23 19:16:42PST1993	8	*0.00	0.000	0.000	0.000
Tue Feb23 20:20:10PST1993	6	*0.00	0.000	0.000	0.000
Tue Feb23 21:23:38PST1993	5	*0.00	0.000	0.000	0.000
Tue Feb23 22:27:07PST1993	8	*0.00	0.000	0.000	0.000
Tue Feb23 23:30:36PST1993	8	*0.00	0.000	0.000	0.000
Wed Feb24 00:34:04PST1993	7	*0.00	0.000	0.000	0.000
Wed Feb24 01:37:32PST1993	7	*0.00	0.000	0.000	0.000
Wed Feb24 02:41:00PST1993	8	*0.00	0.000	0.000	0.000
Wed Feb24 03:44:28PST1993	7	*0.00	0.000	0.000	0.000
Wed Feb24 04:47:57PST1993	5	*0.00	0.000	0.000	0.000
Wed Feb24 05:51:25PST1993	4	*0.00	0.000	0.000	0.000
Wed Feb24 06:54:53PST1993	6	*0.00	0.000	0.000	0.000
Wed Feb24 07:58:21PST1993	8	*0.00	0.000	0.000	0.000
Wed Feb24 09:01:48PST1993	8	*0.00	0.000	0.000	0.000
Wed Feb24 10:05:16PST1993	6	*0.00	0.000	0.000	0.000
Wed Feb24 11:08:44PST1993	8	*0.00	0.000	0.000	0.000
Wed Feb24 12:12:14PST1993	10	*0.00	0.000	0.000	0.000
Wed Feb24 13:15:43PST1993	9	*0.00	0.000	0.000	0.000
Wed Feb24 14:19:13PST1993	7	*0.00	0.000	0.000	0.000

Figure A.13 continued

C. INDIVIDUAL NETWORK NODE TRAFFIC STATUS MEASUREMENT

After getting the long-term NPS campus network traffic information, we can select the 'ece' network node as the server node and our network node as the client node. Under this server/client network architecture, it is possible to use the file transfer to simulate the network traffic load between these two nodes. This file transfer simulation measures the continual network traffic between the client and the server.

Before starting the file transfer simulation, it is straight forward to use the 'ECHO data packets - Ping Utility' function in Figure A.1 to test the reachability of the server network node. Having tested it, we found the 'ece' network node can be reached in Figure A.14. Then, we can execute the server part file transfer program in the 'ece' network node. After activating the server part program, we will select the 'file transfer measurement tools' function in Figure A.1 to activate the client part file transfer program in our network node. In the 'file transfer measurement tools' function, it is necessary to set the file transfer simulation's parameters shown in Figure A.16 in the 'file transfer measurement' subfunction as shown in Figure A.15.

```
*** Welcome to the ECHO Data Packets - Ping Utility ***
*** Please input the symbolic name/internet Address of server ==> ece
*** Please select one function for Ping Utility ***
    1. Reachability test
    2. Send echo_request and get echo_response
        with options of routing types, verbosity, datagram sizing
Please select one ==> 1
*** Please input the seconds of timeout (default : 20 seconds) ==> 30
!!! Please wait for Ping Utility Process !!!
ece.nps.navy.mil is alive
```

Figure A.14 A Dialogue of 'ece' Network Node Reachability Test

<<<<< File Transfer Measurement Tools Main Menu >>>>>

1. File Transfer Measurement
2. Measurement Statistics Data Report
3. Data File Maintenance Utility
4. Exit

Please select one function (1 - 4) ==>>>

Figure A.15 A Menu of File Transfer Measurement Tools function

```
*** Please input the symbolic name/internet address of server ==> ece
*** Please select the protocol for measurement ***
    1. TCP
    2. UDP
Please select one ==> 1
*** Please select the message buffer size for measurement ***
    1. 1024 bytes
    2. 512 bytes
    3. 256 bytes
Please select one ==> 1
!!! Please wait a second for the transferred file size proceeding !!!
    1. 1Mbytes
    2. 2 Mbytes
    3. 3 Mbytes
    4. 128 Kbytes
    5. 256 Kbytes
    6. 512 Kbytes
    7. 64 Kbytes
Please select one, <8> to exit ==> 1
*** Please input the measurement times that you want ==> 100
```

Figure A.16 A Dialogue of File Transfer Simulation Parameters Setting

When the file transfer simulation begins, we can use the 'individual host network status query' function to retrieve the network and system status in the local computer host. There are several query functions for the network status, input/output, and virtual memory statistics reports. The 'Show Network Status Utility' function in Figure A.1 retrieves the network status of the host. There are two execution modes and three categories of network status measurements shown in Figure A.17.

```
*** Welcome to the Network Status Utility ***  
*** Please select the execution mode ***  
1. BATCH mode  
2. INTERACTIVE mode  
Please select one ==> 2  
*** Please select one function for Network Status Utility ***  
1. Active Sockets  
   Display a list of active sockets for each protocol  
2. Network Data Structure/Routing Table  
   Select one from various other network data structures  
3. Cumulative Traffic Statistics of packet traffic  
   on configured network interfaces  
Please select one ==> 1
```

Figure A.17 A dialogue of Show Network Status Utility Function for Execution Mode and Network Information Selection

We can retrieve the active sockets information first to make sure that the file transfer simulation works well. Before querying the active sockets, it is necessary to set the required parameters for it as shown in Figure A.18. Figure A.19 shows the active sockets network information. It is found that the first TCP active socket record displays the active TCP socket used by the file transfer simulation.

*** Please select options for function 1 ***

Active Sockets

- a- Show the stat of all sockets
- A- Show the address of any protocol control blocks
associated with sockets (for debugging)
- n- Show network addresses as numbers

Please select them ==> [aAn] aAn

*** Please select address family for function 1 ***

- 1. AF_INET address family
- 2. AF_UNIX address family

Please select one ==> 1

*** Please input the system argument (default value : /vmunix) ==>

*** Please input the core argument (default value : /dev/kmem) ==>

Figure A.18 A Dialogue of Show Network Status Utility Function for
Active Sockets Parameters Setting

Active Internet Connection (including servers)				
PCB	Proto	Recv-Q	Send-Q	Local Address Foreign Address (state)
ffa5ee0c	udp	0	0	*.1804 **
ffa6408c	udp	0	0	*.1798 **
ffa6b78c	udp	0	0	*.645 **
ffa69b8c	udp	0	0	*.1022 **
ffa6768c	udp	0	0	*.1020 **
ffa7980c	udp	0	0	* **
ffa5a50c	udp	0	0	*.1345 **
ffa4e90c	udp	0	0	*.695 **
ffa4e30c	udp	0	0	*.696 **
ffa5700c	udp	0	0	*.1053 **
ffa5718c	udp	0	0	*.1052 **
ffa5748c	udp	0	0	*.1051 **
ffa5740c	udp	0	0	*.1050 **
ffa5750c	udp	0	0	*.1049 **
ffa5768c	udp	0	0	*.19 **
ffa5790c	udp	0	0	*.13 **
ffa57b8c	udp	0	0	*.9 **
ffa57e8c	udp	0	0	*.7 **
ffa5408c	udp	0	0	*.37 **
ffa5458c	udp	0	0	*.517 **
ffa5470c	udp	0	0	*.512 **
ffa54e0c	udp	0	0	*.42 **
ffa54b8c	udp	0	0	*.736 **
ffa5548c	udp	0	0	*.733 **
ffa4e80c	udp	0	0	*.728 **
ffa4e40c	udp	0	0	*.705 **
ffa5578c	udp	0	0	*.725 **
ffa5568c	udp	0	0	*.724 **
ffa55a8c	udp	0	0	*.1048 **
ffa55b8c	udp	0	0	*.1047 **
ffa55d8c	udp	0	0	*.1046 **
ffa55d0c	udp	0	0	*.1045 **
ffa4e70c	udp	0	0	*.706 **
ffa4f08c	udp	0	0	* **
ffa4ea0c	udp	0	0	*.699 **
ffa4e98c	udp	0	0	*.514 **
ffa4ed0c	udp	0	0	*.1023 **
ffa4f40c	udp	0	0	*.520 **
ffa4f60c	udp	0	0	*.659 **
ffa4f80c	udp	0	0	*.1027 **
ffa4fc0c	udp	0	0	*.1043 **
ffa4fa0c	udp	0	0	*.111 **
ffa4fc8c	udp	0	0	*.1025 **
ffa6d80c	tcp	0	1024	131.120.5.22.1187131.120.20.2.7003ESTABLISHED

Figure A.19 Active Sockets Information Report

ffa6c38c tcp	0	0*.6000	**	LISTEN
ffa7040c tcp	0	0*.1056	**	LISTEN
ffa56e8c tcp	0	0*.1027	**	LISTEN
ffa5788c tcp	0	0*.19	**	LISTEN
ffa57a0c tcp	0	0*.13	**	LISTEN
ffa57d0c tcp	0	0*.9	**	LISTEN
ffa5400c tcp	0	0*.7	**	LISTEN
ffa5420c tcp	0	0*.37	**	LISTEN
ffa5438c tcp	0	0*.79	**	LISTEN
ffa5448c tcp	0	0*.540	**	LISTEN
ffa5478c tcp	0	0*.512	**	LISTEN
ffa5480c tcp	0	0*.513	**	LISTEN
ffa5500c tcp	0	0*.514	**	LISTEN
ffa54d0c tcp	0	0*.515	**	LISTEN
ffa54e8c tcp	0	0*.23	**	LISTEN
ffa5540c tcp	0	0*.21	**	LISTEN
ffa5570c tcp	0	0*.721	**	LISTEN
ffa5590c tcp	0	0*.718	**	LISTEN
ffa55c0c tcp	0	0*.713	**	LISTEN
ffa4e08c tcp	0	0*.709	**	LISTEN
ffa4e38c tcp	0	0*.708	**	LISTEN
ffa4f10c tcp	0	0*.25	**	LISTEN
ffa4f48c tcp	0	0*	**	LISTEN
ffa4fd8c tcp	0	0*.1024	**	LISTEN
ffa4f88c tcp	0	0*.111	**	LISTEN

Figure A.19 continued

During the file transfer simulation, one may measure the input/output activities of the local computer host. The 'I/O statistics report utility' function (See Figure A.1) is performing the operation. There are several I/O statistics reports and each is generated by the parameters given by the users. Figure A.20 shows a parameters setting.

```

*** Welcome to the I/O Statistics Report Utility ***

*** Please select the execution mode ***

1. BATCH mode
2.INTERACTIVE mode

Please select one ==> 2

*** Please select options for I/O Statistics Report Utility ***

c- Report percentage of time the system has spent in user mode
d- For each disk, report
  the number of Kbytes transferred per second,
  the number of transfers per second,
  the milliseconds per average seek
D- For each disk, report
  the reads/writes per second,
  the percentage of disk utilization
l- Report the counts in each interval
t- Report the read/write character number to terminal

Please select them ==> [cdDlt] cdDlt

*** Do you want to limit the number of disk
    included in the report (Y/N) ==>
n

*** Please explicitly specify the disks to be reported ==>

*** Please set once each interval seconds ==> (default : 1 second)
2

*** Please set count reports
10

```

Figure A.20 A Dialogue of I/O Statistics Report Utility Function for
I/O Statistics Report Parameters Setting

tty				sr0			cpu		
tin	tout	rpi	wpi	util	us	ni	sy	id	
0	4	29608	36094	0.4	1	0	2	97	
0	38	0	0	0.0	4	0	21	74	
0	478	1	0	2.0	17	0	18	65	
0	37	0	0	0.0	3	0	12	84	
0	37	0	0	0.0	2	0	24	74	
0	37	0	0	0.0	4	0	19	77	
0	37	0	0	0.0	5	0	19	76	
0	37	0	0	0.0	4	0	21	75	
0	37	0	0	0.0	8	0	29	63	
0	485	0	7	10.9	18	0	21	61	

Figure A.21 I/O Statistics Report

Figure A.21 shows the I/O activities during the file transfer simulation. This I/O statistics report includes terminal, disk and CPU activities. For detail information about each field, please refer to Appendix B.

The 'virtual memory statistics report utility' in Figure A.1 function collects the following information: process, real/virtual memory, page fault, paging activities, disk operation, trap/interrupt rate, and an usage of CPU time. These system information reveals the status of a system during the file transfer simulation. Figure A.22 displays the dialogue of the parameter settings in the 'virtual memory statistics report utility'. Figure A.23 shows the system information during the file transfer simulation.

```

*** Welcome to the Virtual Memory Statistics Report Utility ***

*** Please select the execution mode ***

1. BATCH mode
2. INTERACTIVE mode

Please select one ==> 2

*** Please select options for Virtual Memory Statistics Report Utility ***

f- Report on
  the number of forks and vforks since system startup,
  the number of pages of virtual memory involved
  in each kind of fork
i- Report the number of interrupts per device
s- Display the contents of sum structure
S- Report on swapping activity

Please select one ==> [fisS]

*** Please set once each interval seconds ==>
2

*** Please set count reports ==>
10

```

Figure A.22 A Dialogue of Virtual Memory Statistics Report Utility Function for Virtual Memory Statistics Report Parameters Setting

procs			memory				page				disk					faults			cpu			
r	b	w	avm	fre	si	so	pi	po	fr	de	sr	s0	s1	s2	s3	in	sy	cs	us	sy	id	
1	0	0	0	1632	45	45	0	0	0	0	0	0	0	0	0	0	71	14	1	2	97	
0	0	0	0	1608	0	0	0	0	0	0	0	0	0	0	0	0	432	173	15	24	62	
0	0	0	0	1608	0	0	0	0	0	0	0	0	0	0	0	0	404	169	4	28	77	
0	0	0	0	1552	0	0	0	0	0	0	0	0	0	0	0	0	388	168	1	23	76	
0	0	0	0	1552	0	0	0	0	0	0	0	0	0	0	0	0	378	168	3	24	73	
0	0	0	0	1552	0	0	0	0	0	0	0	0	0	0	0	0	372	167	4	24	71	
0	0	0	0	1536	0	0	0	0	0	0	0	0	0	0	0	0	411	166	5	25	80	
0	0	0	0	1536	0	0	0	0	0	0	0	4	0	0	0	0	392	165	5	23	72	
0	0	0	0	1536	0	0	0	0	0	0	0	0	0	0	0	0	382	163	15	15	70	
0	0	0	0	1536	0	0	0	0	0	0	0	0	0	0	0	0	374	164	5	20	75	

Figure A.23 Virtual Memory Statistics Report

After querying the network and system information, we will look at the file transfer simulation situation. Figure A.24 shows the first two file transfer simulation dialogue.

```
====>> Measurement times is : 1

!!! File trafsfer simulation is proceeding !!!

!!! File transfer finished !!!

*** SUMMARY ***

The transfered file length : 1048576
The file transfer time is : Wed Mar 3 01:28:49 1993
The sec for file transfer with disk I/O is : 14
The mirco sec for file transfer with disk I/O is : 119570
The sec for file transfer without disk I/O is : 13
the mirco sec for file transfer without disk I/O is : 425316

====>> Measurement times is : 2

!!! File trafsfer is proceeding !!!

!!! File transfer finished !!!

*** SUMMARY ***

The transfered file length : 1048576
The file transfer time is : Wed Mar 3 01:29:03 1993
The sec for file transfer with disk I/O is : 13
The mirco sec for file transfer with disk I/O is : 193655
The sec for file transfer without disk I/O is : 12
the mirco sec for file transfer without disk I/O is : 530537
```

Figure A.24 A output of File Transfer Simulation

When the 'file transfer measurement' subfunction in Figure A.15 finishes, we can select the 'measurement statistic data report' subfunction to generate the statistics report for this simulation. Before the statistics report is generated, it is necessary to input the parameters to the 'file transfer measurement' subfunction as shown in Figure A.25.

```

*** Please input the server address (name, alias) ==>>
ece

!!! Please wait a second for file transfer data file listing !!!

1. TCP_1024.1MB
2. TCP_1024.2MB
3. TCP_1024.3MB
4. TCP_256.1MB
5. TCP_256.2MB
6. TCP_256.3MB
7. TCP_512.1MB
8. TCP_512.2MB
9. TCP_512.3MB
10. UDP_256.128KB
11. UDP_256.64KB

Please select one, <12> to exit ==> 1

*** How Many file transfer simulation samples(10~)
    will be used in one statistics ==> 100

*** How many minutes(10~) do you want to be a time hour interval range ==>
10

```

Figure A.25 a Parameters Setting Dialogue in
Measurement Statistics Data Report Subfunction

Figure A.26 shows the statistics report of the file transfer simulation. This report would reveal the on-going network traffic situation from local node to the 'ece' network node during the file transfer simulation.

**** File Transfer Simulation Statistics Report ****

The address of server : 131.120.20.2
The file name is : TCP_1024.1MB
The file length is : 1048576 bytes
The sample number of record is : 100
The time hour interval range is : per 10 minutes

*** File transfer simulation time range ***

File transfer simulation start time: Wed Mar 3 01:28:49 1993

File transfer simulation end time : Wed Mar 3 01:51:16 1993

The file transfer simulation time hour interval begin: Wed Mar 3 01:28:49 1993

The file transfer simulation time hour interval end : Wed Mar 3 01:38:49 1993

The percent in this time hour interval is : 44.00

The file transfer simulation time hour interval begin: Wed Mar 3 01:38:49 1993

The file transfer simulation time hour interval end : Wed Mar 3 01:48:49 1993

The percent in this time hour interval is : 45.00

The file transfer simulation time hour interval begin: Wed Mar 3 01:48:49 1993

The file transfer simulation time hour interval end : Wed Mar 3 01:51:16 1993

The percent in this time hour interval is : 11.00

*** File transfer with disk I/O ***

The mean value is : 13.187021 seconds

The variance : 0.194408 seconds

The standard deviation is : 0.440918 seconds

The max file transfer time is : 15 sec, 131321 usec

The min file transfer time is : 12 sec, 796478 usec

The file transfer time range from 12 seconds to 13 seconds is : 39.00 percent

The file transfer time range from 13 seconds to 14 seconds is : 53.00 percent

The file transfer time range from 14 seconds to 15 seconds is : 7.00 percent

The file transfer time range from 15 seconds to 16 seconds is : 1.00 percent

*** File transfer with disk I/O ***

The mean value is : 12.499682seconds

The variance : 0.182435 seconds

The standard deviation is : 0.427124 seconds

The max file transfer time is : 14 sec, 468310 usec

The min file transfer time is : 12 sec, 130410 usec

The file transfer time range from 12 seconds to 13 seconds is : 91.00 percent

The file transfer time range from 13 seconds to 14 seconds is : 8.00 percent

The file transfer time range from 14 seconds to 15 seconds is : 1.00 percent

*** End of file transfer data statistics function ***

Figure A.26 File Transfer Simulation Statistics Report

D. SUMMARY

This appendix shows the application of the 'network traffic measurement utility' on the NPS campus network. Potential users may use this appendix to guide their reading of Chapter III. Hopefully, with this utility one may achieve certain performance management over the network in a regular user's capacity. In other words, one does not need to bother a system manager to use this utility.

APPENDIX B - I/O & VIRTUAL MEMORY STATISTICS REPORTS' FIELDS MAPPING TABLE

1. I/O statistics report :

cpu Report the percentage of system time :

us user mode

ni in user mode running low priority processes

sy system mode id idling

disk Report the read/write operation and percentage disk utilization :

rpi the reads per second (interval)

wpi the writes per second (interval)

util percentage disk utilization

terminal Report the number of read/write character to terminal :

tin the number of input characters

tout the number of output characters

2. virtual memory statistics report :

procs Report the number of processes in each of the three following states:

r in run queue

b blocked for resources (i/o, paging, etc.)

w runnable or short sleeper (< 20 secs) but swapped

memory Report on usage of virtual and real memory. Virtual memory is considered active if it belongs to processes which are running or have run in the last 20 seconds.

avm number of active virtual Kbytes

fre size of the free list in Kbytes

page Report information about page faults and paging activity.

The information on each of the following activities is averaged each five seconds, and given in units per second.

re page reclaims

at number of attaches

pi kilobytes per second paged in
po kilobytes per second paged out
fr kilobytes freed per second
de anticipated short term memory short fall in Kbytes
sr pages scanned by clock algorithm, per-second

disk Report number of disk operations per second (this field is system dependent).

s0 disk identification number
s1 disk identification number
s2 disk identification number
s3 disk identification number

faults Report trap/interrupt rate averages per second over last 5 seconds.

in (non clock) device interrupts per second
sy system calls per second
cs CPU context switch rate (switches/sec)

cpu Give a breakdown of percentage usage of CPU time.

us user time for normal and low priority processes
sy system time
id CPU idle

APPENDIX C - SOURCE CODE OF NETWORK TRAFFIC MONITORING MAIN PROGRAM

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <sys/types.h>
#include <sys/socket.h>
#include </usr/sys/netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <fcntl.h>
#include <string.h>

#define SERV_TCP_PORT 7003
#define SERV_UDP_PORT 7102

#ifndef INADDR_NONE
#define INADDR_NONE 0xffffffff
#endif

main(argc, argv)

int argc;
char **argv;

{
    int nslookup_status_m, ping_status_m, netstat_status_m;
    int opt_fun_m, exit_flag_m, continue_flag_m;
    char argument_m[256];

    nslookup_status_m=0;
    ping_status_m=0;
    netstat_status_m=0;
    printf("\n\n!!! Check unix system command, ");
    printf("Please wait !!!\n\n");
    check_unix_system_command(&nslookup_status_m, &ping_status_m,
                             &netstat_status_m);
}
```

```

opt_fun_m=0;
exit_flag_m=0;
while ((opt_fun_m < 1) || (opt_fun_m > 4) || (exit_flag_m == 0))
{
    continue_flag_m=1;
    opt_fun_m=display_main_menu(ping_status_m);
    switch (opt_fun_m)
    {
        case 1 :
            printf("====>> Do you really want to execute this function (Y/N) ? ");
            continue_flag_m=confirm_continue();
            if (continue_flag_m == 1)
            {
                individual_host_query(nslookup_status_m,ping_status_m,netstat_status_m);
            }
            break;

        case 2 :
            if (ping_status_m == 1)
            {
                printf("\n!!! PING command ");
                printf("is not available in this host !!!\n\n");
                getchar();
                opt_fun_m=0;
            }
            else
            {
                printf("====>> Do you really want to execute this function (Y/N) ? ");
                continue_flag_m=confirm_continue();
                if (continue_flag_m == 1)
                {
                    argument_m[0]='\0';
                    if (argc > 1)
                    {
                        strcpy(argument_m,argv[1]);
                    }
                    network_profile_query(argc,argument_m);
                }
            }
            break;

        case 3 :
            printf("====>> Do you really want to execute this function (Y/N) ? ");

```

```

        continue_flag_m=confirm_continue();
        if (continue_flag_m == 1)
        {
            network_profile_maintain();
        }
        break;

    case 4 :
        printf("\n Do you really want to exit ? (Y/N) ");
        exit_flag_m=confirm_continue();
        break;

    default :
        printf("\n Main function selection is error, please check it !!! \n");
        break;
    }
}

printf("\n\n!!!! Exit the Network Traffic Measurement ");
printf(" !!!!!\n\n");
getchar();

}

/* This procedure will check the 'nslookup', 'ping', 'netstat'
   in this computer host which this program be executed.
   If some of them are not available in this computer host, this
   procedure will return the status code to the main procedure */
check_unix_system_command(nslookup_status_cusc,ping_status_cusc,
                           netstat_status_cusc,iostat_status_cusc,
                           vmstat_status_cusc)

int *nslookup_status_cusc;
int *ping_status_cusc;
int *netstat_status_cusc;
int *iostat_status_cusc;
int *vmstat_status_cusc;

{
    FILE *exit_cusc;
    char exit_cmd_cusc[6],system_cmd_cusc[30];

```

```

exit_cusc=fopen("exit","w");
strcpy(exit_cmd_cusc,"exit");
exit_cmd_cusc[strlen(exit_cmd_cusc)]='\0';
fprintf(exit_cusc,"%s",exit_cmd_cusc);
fclose(exit_cusc);
if (system("nslookup > status.tmp < exit") == 256)
{
    *nslookup_status_cusc=1;
}
else
{
    system("rm status.tmp");
}
system("rm exit");
if (system("ping taurus > status.tmp") == 256)
{
    *ping_status_cusc=1;
}
else
{
    system("rm status.tmp");
}
if (system("netstat > status.tmp") == 256)
{
    *netstat_status_cusc=1;
}
else
{
    system("rm status.tmp");
}
if (system("iostat > status.tmp") == 256)
{
    *iostat_status_cusc=1;
}
else
{
    system("rm status.tmp");
}
if (system("vmstat > status.tmp") == 256)
{
    *vmstat_status_cusc=1;
}

```

```

else
{
    system("rm status.tmp");
}

}

/* This procedure will display main functions and
   let user select one of them */
int display_main_menu(ping_status_dmm)

int ping_status_dmm;

{
    int opt_fun_dmm;

    printf("\n\n\n    <<<<< Network Traffic Measurement Main Menu >>>>> \n\n");
    printf("        1. Query Individual Host ");
    printf("Network Status \n\n");
    if (ping_status_dmm == 1)
    {
        printf("*not available ");
    }
    else
    {
        printf("        ");
    }
    printf("2. Query Network Status by Network Profile\n\n");
    printf("        3. Maintain Network Profile \n\n");
    if (ping_status_dmm == 1)
    {
        printf("        4. Exit \n\n");
        printf("        (*not available UNIX COMMAND, \n");
        printf("        Please contact the administrator of ");
        printf("this host \n");
        printf("        to get the solution) \n\n\n");
    }
    else
    {
        printf("        4. Exit \n\n\n");
    }
    printf("        Please select one function (1-4) ==>> ");

```

```

scanf("%d",&opt_fun_dmm);
getchar();
printf("\n");
return opt_fun_dmm;

}

```

```

/* Tools for Network Traffic Measurement by individual query*/
individual_host_query(nslookup_status_ihq,ping_status_ihq,
                    netstat_status_ihq,iostat_status_ihq,
                    vmstat_status_ihq)

```

```

int nslookup_status_ihq;
int ping_status_ihq;
int netstat_status_ihq;
int iostat_status_ihq;
int vmstat_status_ihq;

```

```

{
int opt_fun_ihq;
int continue_flag_ihq,exit_flag_ihq;

```

```

opt_fun_ihq=0;
exit_flag_ihq=0;
while ((opt_fun_ihq < 1) || (opt_fun_ihq > 9) || (exit_flag_ihq == 0))
{
continue_flag_ihq=1;
display_individual_host_query_menu(nslookup_status_ihq,
                                ping_status_ihq,
                                netstat_status_ihq,
                                iostat_status_ihq,
                                vmstat_status_ihq,
                                &opt_fun_ihq);

```

```

switch (opt_fun_ihq)
{
case 1 :
printf("==>> Do you really want to execute this function (Y/N) ? ");
continue_flag_ihq=confirm_continue();
if (continue_flag_ihq == 1)
{
query_host_name_and_address();

```

```

    }
    break;

case 2 :
    if (nslookup_status_ihq == 1)
    {
        printf("\n!!! NSLOOKUP command is not available in this host !!!\n\n");
        getchar();
        opt_fun_ihq=0;
    }
    else
    {
        printf("====>> Do you really want to execute this function (Y/N) ? ");
        continue_flag_ihq=confirm_continue();
        if (continue_flag_ihq == 1)
        {
            nslookup_utility();
        }
    }
    break;

case 3 :
    if (ping_status_ihq == 1)
    {
        printf("\n!!! PING command is not available in this host !!!\n\n");
        getchar();
        opt_fun_ihq=0;
    }
    else
    {
        printf("====>> Do you really want to execute this function (Y/N) ? ");
        continue_flag_ihq=confirm_continue();
        if (continue_flag_ihq == 1)
        {
            ping_utility();
        }
    }
    break;

case 4 :
    if (netstat_status_ihq == 1)
    {
        printf("\n!!! NETSTAT command is not available in this host !!!\n\n");

```

```

    getchar();
    opt_fun_ihq=0;
}
else
{
    printf("==>> Do you really want to execute this function (Y/N) ? ");
    continue_flag_ihq=confirm_continue();
    if (continue_flag_ihq == 1)
    {
        network_status_utility();
    }
}
break;

case 5 :
    if (iostat_status_ihq == 1)
    {
        printf("\n!!! IOSTAT command ");
        printf("is not available in this host !!!\n\n");
        getchar();
        opt_fun_ihq=0;
    }
    else
    {
        printf("==>> Do you really want to execute this function (Y/N) ? ");
        continue_flag_ihq=confirm_continue();
        if (continue_flag_ihq == 1)
        {
            io_statistics_report_utility();
        }
    }
    break;

case 6 :
    if (vmstat_status_ihq == 1)
    {
        printf("\n!!! VMSTAT command ");
        printf("is not available in this host !!!\n\n");
        getchar();
        opt_fun_ihq=0;
    }
    else
    {

```



```

        printf("==>> Do you really want to execute this function (Y/N) ? ");
        continue_flag_ihq=confirm_continue();
        if (continue_flag_ihq == 1)
        {
            vm_statistics_report_utility();
        }
    }
    break;

case 7 :
    printf("==>> Do you really want to execute this function (Y/N) ? ");
    continue_flag_ihq=confirm_continue();
    if (continue_flag_ihq == 1)
    {
        unix_command_execution();
    }
    break;

case 8 :
    printf("==>> Do you really want to execute this function (Y/N) ? ");
    continue_flag_ihq=confirm_continue();
    if (continue_flag_ihq == 1)
    {
        file_transfer_measurement_tools();
    }
    break;

case 9 :
    printf("\n Do you really want to exit ? (Y/N) ");
    exit_flag_ihq=confirm_continue();
    break;

default :
    printf("\n Function selection is error, please check it !!! \n");
    break;
}
}

printf("\n\n!!!! Exit the Network Traffic Measurement ");
printf("Utility for individual host query !!!!\n\n");
getchar();

}

```

```

/* This function will display the main menu for user to select
   the function of the network traffic measurement */
display_individual_host_query_menu(nslookup_status_dihqm,
                                   ping_status_dihqm,
                                   netstat_status_dihqm,
                                   iostat_status_dihqm,
                                   vmstat_status_dihqm,
                                   opt_fun_dihqm)

int nslookup_status_dihqm;
int ping_status_dihqm;
int netstat_status_dihqm;
int iostat_status_dihqm;
int vmstat_status_dihqm;
int *opt_fun_dihqm;

{
int opt_buf_dihqm;

printf("\n\n\n <<<<< Network Traffic Measurement Individual");
printf(" Host Query Menu >>>>> \n\n");
printf("      1. Host Name and Address Query \n\n");
if (nslookup_status_dihqm == 1)
{
printf("*not available ");
}
else
{
printf("      ");
}
printf("2. Query Internet Domain Name Servers ");
printf("Utility \n\n");
if (ping_status_dihqm == 1)
{
printf("*not available ");
}
else
{
printf("      ");
}
}

```

```

printf("3. ECHO data packets - Ping Utility\n\n");
if (netstat_status_dihqm == 1)
{
    printf("*not available ");
}
else
{
    printf(" ");
}
printf("4. Show Network status Utility\n\n");
if (netstat_status_dihqm == 1)
{
    printf("*not available ");
}
else
{
    printf(" ");
}
printf("5. I/O statistics report utility\n\n");
if (netstat_status_dihqm == 1)
{
    printf("*not available ");
}
else
{
    printf(" ");
}
printf("6. Virtual memory statistics report utility ");
printf("\n\n");
printf("      7. Unix Command Tools\n\n");
printf("      8. File Transfer Measurement Tools\n\n");
if (nslookup_status_dihqm == 1 || ping_status_dihqm == 1 ||
    netstat_status_dihqm == 1 || iostat_status_dihqm == 1 ||
    vmstat_status_dihqm == 1)
{
    printf("      9. Exit\n\n");
    printf("      (*not available UNIX COMMAND,\n");
    printf("      Please contact the adminstrator of this host\n");
    printf("      to get the solution)\n\n\n");
}
else
{
    printf("      9. Exit\n\n\n");
}

```

```

    }
    printf("          Please select one function (1-9) ==>> ");
    scanf("%d",&opt_buf_dihqm);
    getchar();
    printf("\n");
    *opt_fun_dihqm=opt_buf_dihqm;
}

```

```

/* This function will let users confirm that the function will be
   executed or not. */

```

```

int confirm_continue()
{
    char continue_cc[4];
    int continue_flag_cc;

    gets(continue_cc);
    printf("\n");
    if (strcmp(continue_cc,"Y") == 0 || strcmp(continue_cc,"y") == 0 ||
        strcmp(continue_cc,"Yes") == 0 || strcmp(continue_cc,"yes") == 0 ||
        strcmp(continue_cc,"YES") == 0)
    {
        continue_flag_cc=1;
    }
    else
    {
        continue_flag_cc=0;
    }
    return continue_flag_cc;
}

```

```

/* This procedure provides the host name and address query.
   Users can input the name(alias) of the host and get the address.
   Or users can input the address of the host and get the name
   (alias). */

```

```

query_host_name_and_address()

{
    int continue_flag_qhnaa;
    unsigned long inaddr_qhnaa;
    register char *alias_ptr_qhnaa;

```

```

char host_name_qhnaa[256],host_address_qhnaa[17];
struct in_addr *addr_ptr_qhnaa;
struct hostent *host_info_qhnaa;

continue_flag_qhnaa=1;
host_name_qhnaa[0]='\0';
host_address_qhnaa[0]='\0';
while (continue_flag_qhnaa == 1)
{
    printf("\n *** Please input the symbolic name/internet address of server ==> ");
    scanf("%s",host_address_qhnaa);
    getchar();
    printf("\n");
    if ((inaddr_qhnaa = inet_addr(host_address_qhnaa)) == INADDR_NONE)
    {
        convert_host_name_to_address(host_address_qhnaa,&host_info_qhnaa);
    }
    else
    {
        convert_host_address_information(host_address_qhnaa,&host_info_qhnaa);
    }
    if (strlen(host_address_qhnaa) == 0)
    {
        printf("!!! Network host name/address %s is error !!!\n\n",host_address_qhnaa);
    }
    else
    {
        if (host_info_qhnaa != NULL)
        {
            printf(" *** %s host information ***\n\n",host_address_qhnaa);
            printf("   official host name : %s\n",host_info_qhnaa->h_name);
            printf("   alias listing   : ");
            while ((alias_ptr_qhnaa=(host_info_qhnaa->h_aliases)) != NULL)
            {
                printf(" : %s\n",alias_ptr_qhnaa);
                host_info_qhnaa->h_aliases++;
                printf(" ");
            }
            printf("\n   address type      : %d\n",host_info_qhnaa->h_addrtype);
            printf("   address length   : %d\n",host_info_qhnaa->h_length);
            switch (host_info_qhnaa->h_addrtype)
            {
                case AF_INET :

```

```

        addr_ptr_qhnaa=(struct in_addr *) host_info_qhnaa->h_addr_list[0];
        printf("    Internet address : %s\n",inet_ntoa(addr_ptr_qhnaa));
        break;

    default :
        printf("!!! Unknown address type !!!\n\n");
        break;
    }
}
}
printf("\n *** Do you want to query another host information (Y/N) ");
continue_flag_qhnaa=confirm_continue();
}

}

/* This procedure will use the system call 'nslookup'.
   Let users query the internet domain name servers */
nslookup_utility()
{
    int local_host_server_nu;
    int local_host_server_err_nu;
    char nslookup_cmd_nu[30],serv_host_addr_nu[17];
    unsigned long inaddr_nu;
    struct hostent *dummy_ptr_nu;
    /* this is a dummy variable for the convert_host_name_to_address
       procedure in this procedure. */

    local_host_server_nu=1;
    nslookup_cmd_nu[0]='\0';
    serv_host_addr_nu[0]='\0';
    local_host_server_err_nu=0;
    printf("*** Welcome to ");
    printf("the Internet Domain Name Servers Query Utility ***\n\n");
    printf("*** Do you want to use the local host's name server\n");
    printf("    instead of the default servers ==> (Y/N) ");
    local_host_server_nu=confirm_continue();
    if (local_host_server_nu == 1)
    {
        printf("\n *** Please input the symbolic name/internet address of server ==> ");
        scanf("%s",serv_host_addr_nu);
        printf("\n");
    }
}

```

```

    if ((inaddr_nu = inet_addr(serv_host_addr_nu))
        == INADDR_NONE)
    {
        convert_host_name_to_address(serv_host_addr_nu,&dummy_ptr_nu);
    }
    else
    {
        convert_host_address_information(serv_host_addr_nu,&dummy_ptr_nu);
    }
    if (strlen(serv_host_addr_nu) == 0)
    {
        printf("!!! Local host name/address is error !!!\n\n");
        local_host_server_err_nu=1;
    }
    else
    {
        strcpy(nslookup_cmd_nu,"nslookup -l ");
        strcat(nslookup_cmd_nu,serv_host_addr_nu);
    }
}
else
{
    strcpy(nslookup_cmd_nu,"nslookup");
}
if (local_host_server_err_nu == 0)
{
    printf("\n!!! Type <?> or <help> for interactive query ");
    printf("help !!!\n");
    printf("!!! Type <exit> to exit the Internet Domain Name Servers Utility !!!\n\n");
    nslookup_cmd_nu[strlen(nslookup_cmd_nu)]='\0';
    system(nslookup_cmd_nu);
}
}

```

```

/* This procedure will use the system call 'ping'.
   Let users send ICMP ECHO_REQUEST packet to network host
   and get the information about the network host. */

```

```

ping_utility()
{
    int continue_flag_pu;
    int ping_fun_pu, batch_mode_pu;

```

```

int network_host_err_pu,ping_opt_err_pu,char_index_pu;
long timeout_seconds_pu;
char datagram_packet_size_s_pu[34],count_s_pu[34];
char timeout_seconds_s_pu[34];
char ping_cmd_pu[256],ping_opt_pu[6];
char ping_batch_command_pu[256];
char serv_host_addr_pu[17],tmp_host_addr_pu[17];
unsigned long inaddr_pu;
struct hostent *dummy_ptr_pu;
/* this is a dummy variable for the convert_host_name_to_address
   procedure in this procedure. */

continue_flag_pu=1;
while (continue_flag_pu == 1)
{
    ping_cmd_pu[0]='\0';
    tmp_host_addr_pu[0]='\0';
    serv_host_addr_pu[0]='\0';
    network_host_err_pu=0;
    printf("\n\n*** Welcome to ");
    printf("the ECHO data packets - Ping Utility ***\n\n");
    batch_mode_pu=select_execution_mode();
    if (batch_mode_pu == 1)
    {
        ping_batch_command_pu[0]='\0';
        printf("\n!!! Type <exit> or <quit> to exit the ping utility batch mode !!!\n\n");
        while ((strcmp(ping_batch_command_pu,"exit")) != 0 &&
            (strcmp(ping_batch_command_pu,"quit")) != 0)
        {
            printf("\nping command > ");
            gets(ping_batch_command_pu);
            if ((strcmp(ping_batch_command_pu,"exit")) != 0 &&
                (strcmp(ping_batch_command_pu,"quit")) != 0)
            {
                if (strcmp(ping_batch_command_pu,"ping",4) == 0)
                {
                    system(ping_batch_command_pu);
                }
                else
                {
                    printf("\n!!! Unknown command (not ping command) !!!\n\n");
                }
            }
        }
    }
}

```



```

    }
    printf("\n!!! Exit the ping utility batch mode execution !!!\n\n");
    getchar();
}
else
{
    printf("\n*** Please input the symbolic name/internet address of server ==> ");
    scanf("%s",tmp_host_addr_pu);
    strcpy(serv_host_addr_pu,tmp_host_addr_pu);
    printf("\n");
    if ((inaddr_pu = inet_addr(tmp_host_addr_pu)) == INADDR_NONE)
    {
        convert_host_name_to_address(tmp_host_addr_pu,&dummy_ptr_pu);
    }
    else
    {
        convert_host_address_information(tmp_host_addr_pu,&dummy_ptr_pu);
    }
    if (strlen(tmp_host_addr_pu) == 0)
    {
        printf("!!! The network host name/address is error !!!\n\n");
        network_host_err_pu=1;
    }
    else
    {
        strcpy(ping_cmd_pu,"ping ");
        ping_fun_pu=0;
        while (ping_fun_pu != 1 && ping_fun_pu != 2)
        {
            printf("*** please select one function for ");
            printf("Ping Utility ***\n\n");
            printf("  1. Reachability test\n\n");
            printf("  2. Send Echo_Request and Get Echo_\n\n");
            printf("      Response with options of routing\n\n");
            printf("      types, verbosity, datagram sizing\n\n");
            printf("\n\n Please select one ==> ");
            scanf("%d",&ping_fun_pu);
            getchar();
            printf("\n");
            if (ping_fun_pu != 1 && ping_fun_pu != 2)
            {
                printf("!!! Ping function selection is error !!!\n\n");
            }
        }
    }
}

```

```

    }
    if (ping_fun_pu == 2)
    {
        strcat(ping_cmd_pu,"-s ");
        ping_opt_err_pu=1;
        datagram_packet_size_s_pu[0]='\0';
        count_s_pu[0]='\0';
        ping_opt_pu[0]='\0';
        while (ping_opt_err_pu == 1)
        {
            printf("\n*** please select options for function 2 ***\n");
            printf(" ");
            printf("Send ECHO_REQUEST and Get ECHO_RESPONSE\n");
            printf(" ");
            printf("information from the network host \n\n");
            printf(" ");
            printf("l- Loose source route \n");
            printf(" ");
            printf("r- Bypass the normal routing tables and \n");
            printf(" ");
            printf(" send directly to host on an attached \n");
            printf(" ");
            printf(" network. \n");
            printf(" ");
            printf("R- Record route \n");
            printf(" ");
            printf("v- Verbose output \n");
            printf("\n\n Please select them ==> [lrRv] ");
            gets(ping_opt_pu);
            printf("\n");
            ping_opt_err_pu=0;
            char_index_pu=0;

            /* Check the options of the ping utility.
            If the users input error option,
            then the users need to input the options again */
            while (char_index_pu < strlen(ping_opt_pu) &&
                ping_opt_err_pu != 1)
            {
                if (ping_opt_pu[char_index_pu] != 'l' && ping_opt_pu[char_index_pu] != 'r'
                    &&
                    ping_opt_pu[char_index_pu] != 'R' && ping_opt_pu[char_index_pu] != 'v')
                {

```

```

        printf("\n!!! Ping option selection is ");
        printf("error !!!\n");
        ping_opt_err_pu=1;
    }
    char_index_pu++;
}
}
if (strlen(ping_opt_pu) > 0)
{
    strcat(ping_cmd_pu,"-");
    strcat(ping_cmd_pu,ping_opt_pu);
    strcat(ping_cmd_pu," ");
}
strcat(ping_cmd_pu,serv_host_addr_pu);
datagram_packet_size_s_pu[0]='\0';
count_s_pu[0]='\0';
get_packet_size_and_count(datagram_packet_size_s_pu,count_s_pu);
strcat(ping_cmd_pu," ");
strcat(ping_cmd_pu,datagram_packet_size_s_pu);
strcat(ping_cmd_pu," ");
strcat(ping_cmd_pu,count_s_pu);
}
else
{
    strcat(ping_cmd_pu,serv_host_addr_pu);
    timeout_seconds_pu=-1;
    timeout_seconds_s_pu[0]='\0';
    while (timeout_seconds_pu < 0)
    {
        printf("\n*** Please input the seconds of timeout (default : 20 seconds) ==> ");
        gets(timeout_seconds_s_pu);
        if (timeout_seconds_s_pu == NULL)
        {
            timeout_seconds_pu=20;
        }
        else
        {
            timeout_seconds_pu=atol(timeout_seconds_s_pu);
            printf("!!! Timeout input error !!!\n");
            printf("!!! Please input it again !!!\n");
        }
    }
    printf("\n");
}

```

```

        strcat(ping_cmd_pu," ");
        strcat(ping_cmd_pu,timeout_seconds_s_pu);
    }
    ping_cmd_pu[strlen(ping_cmd_pu)]='\0';
    printf("\n!!! Please wait for Ping utility process !!!\n\n");
    system(ping_cmd_pu);
}
}
printf("\n*** Do you want to execute the Ping utility again (Y/N) ");
continue_flag_pu=confirm_continue();
}
}

```

```

/* This procedure will get the datagram packet size and
   count number for the 'ping' unix command */
get_packetsize_and_count(datagram_packet_size_s_gpac,count_s_gpac)

```

```

char *datagram_packet_size_s_gpac;
char *count_s_gpac;

```

```

{
    int datagram_packet_size_gpac,count_gpac;

    datagram_packet_size_gpac=-1;
    count_gpac=-1;
    while (datagram_packet_size_gpac < 0)
    {
        printf("*** Please input the size of datagram ");
        printf("packet (default : 64 bytes) ==> ");
        gets(datagram_packet_size_s_gpac);
        printf("\n");
        if (datagram_packet_size_s_gpac == NULL || strlen(datagram_packet_size_s_gpac) <=
0)
        {
            datagram_packet_size_gpac=64;
            sprintf(datagram_packet_size_s_gpac,"%ld",datagram_packet_size_gpac);
        }
        else
        {
            if (strlen(datagram_packet_size_s_gpac) > 0)
            {

```

```

        if (atol(datagram_packet_size_s_gpac) == 0)
        {
            datagram_packet_size_gpac=64;
            sprintf(datagram_packet_size_s_gpac,"%ld",datagram_packet_size_gpac);
        }
        else
        {
            datagram_packet_size_gpac=atol(datagram_packet_size_s_gpac);
        }
    }
}
datagram_packet_size_s_gpac
[strlen(datagram_packet_size_s_gpac)]='\0';
while (count_gpac < 0)
{
    printf("**** Please input the count number (default : 65536 times) ==> ");
    gets(count_s_gpac);
    printf("\n");
    if (count_s_gpac == '\0' ||
        strlen(count_s_gpac) <= 0)
    {
        count_gpac=65536;
        sprintf(count_s_gpac,"%ld",
            count_gpac);
    }
    else
    {
        if (strlen(count_s_gpac) > 0)
        {
            if (atol(count_s_gpac) == 0)
            {
                count_gpac=65536;
                sprintf(count_s_gpac,"%ld",
                    count_gpac);
            }
            else
            {
                count_gpac=atol(count_s_gpac);
            }
        }
    }
}
}

```

```
count_s_gpac[strlen(count_s_gpac)]='\0';
```

```
}
```

```
/* This procedure will show the status of the network.
```

```
   The users can select the different options to request  
   the status of the network          */
```

```
network_status_utility()
```

```
{
```

```
int continue_flag_nsu,confirm_flag_nsu,netstat_batch_exit_nsu;
```

```
int netstat_fun_nsu,batch_mode_nsu;
```

```
int network_host_err_nsu,netstat_opt_err_nsu,char_index_nsu;
```

```
int seconds_interval_nsu;
```

```
char netstat_cmd_nsu[256],netstat_opt_nsu[256];
```

```
char netstat_batch_command_nsu[256];
```

```
char serv_host_addr_nsu[17],tmp_host_addr_nsu[17];
```

```
unsigned long inaddr_nsu;
```

```
struct hostent *dummy_ptr_nsu;
```

```
/* this is a dummy variable for the convert_host_name_to_address  
   procedure in this procedure.          */
```

```
continue_flag_nsu=1;
```

```
while (continue_flag_nsu == 1)
```

```
{
```

```
netstat_cmd_nsu[0]='\0';
```

```
tmp_host_addr_nsu[0]='\0';
```

```
serv_host_addr_nsu[0]='\0';
```

```
network_host_err_nsu=0;
```

```
netstat_batch_exit_nsu=0;
```

```
printf("\n\n*** Welcome to the Network Status Utility ***\n\n");
```

```
strcpy(netstat_cmd_nsu,"netstat ");
```

```
batch_mode_nsu=select_execution_mode();
```

```
if (batch_mode_nsu == 1)
```

```
{
```

```
netstat_batch_command_nsu[0]='\0';
```

```
printf("\n!!! Type <exit> or <quit> to exit the netstat utility batch mode !!!\n");
```

```
while ((strcmp(netstat_batch_command_nsu,"exit")) != 0 &&  
      (strcmp(netstat_batch_command_nsu,"quit")) != 0)
```

```
{
```

```
printf("\nnetstat command > ");
```

```
gets(netstat_batch_command_nsu);
```

```

        if ((strcmp(netstat_batch_command_nsu,"exit")) != 0 &&
            (strcmp(netstat_batch_command_nsu,"quit")) != 0)
        {
            if (strcmp(netstat_batch_command_nsu,"netstat",7) == 0)
            {
                system(netstat_batch_command_nsu);
            }
            else
            {
                printf("\n!!! Unknown command (not netstat command) !!\n");
            }
        }
    }
    printf("\n!!! Exit the netstat utility batch mode execution !!\n\n");
    netstat_batch_exit_nsu=1;
    getchar();
}
else
{
    netstat_fun_nsu=0;
    while (netstat_fun_nsu < 1 || netstat_fun_nsu > 3)
    {
        printf("*** please select one function for Show Network Status Utility ***\n\n");
        printf(" 1. Active Sockets\n");
        printf("    Display a list of active sockets for each protocol\n\n");
        printf(" 2. Network Data Structure/Rounting Table\n");
        printf("    Select one from various other network ");
        printf("data structures\n\n");
        printf(" 3. Cumulative Traffic Statistics\n");
        printf("    Display running statistics of packet traffic\n");
        printf("    on configured network interfaces\n\n");
        printf("\n Please select one ==> ");
        scanf("%d",&netstat_fun_nsu);
        getchar();
        printf("\n");
        if (netstat_fun_nsu < 1 && netstat_fun_nsu > 3)
        {
            printf("!!! Netstat function selection ");
            printf("is error !!\n");
        }
    }
    switch(netstat_fun_nsu)
    {

```

```

case 1 :
    netstat_opt_err_nsu=1;
    netstat_opt_nsu[0]='\0';
    while (netstat_opt_err_nsu == 1)
    {
        printf("\n*** please select options for function ***\n");
        printf("    Active Sockets \n");
        printf("    a- Show the state of all sockets \n");
        printf("    A- Show the address of any protocol control blocks \n");
        printf("        associated with sockets (for debugging) \n");
        printf("    n- Show network addresses as numbers \n");
        printf("\n\n Please select them ==> [aAn] ");
        gets(netstat_opt_nsu);
        printf("\n");
        netstat_opt_err_nsu=0;
        char_index_nsu=0;

        /* Check the options of the netstat utility,
           If the users input error option, then
           the users need to input the options again */
        while (char_index_nsu < strlen(netstat_opt_nsu) &&
               netstat_opt_err_nsu != 1)
        {
            if (netstat_opt_nsu[char_index_nsu] != 'a' &&
                netstat_opt_nsu[char_index_nsu] != 'A' &&
                netstat_opt_nsu[char_index_nsu] != 'n')
            {
                printf("\n!!! Netstat option selection is error !!!\n");
                netstat_opt_err_nsu=1;
            }
            char_index_nsu++;
        }
    }
    if (strlen(netstat_opt_nsu) > 0)
    {
        strcat(netstat_cmd_nsu, "-");
        strcat(netstat_cmd_nsu, netstat_opt_nsu);
        strcat(netstat_cmd_nsu, " ");
    }
    netstat_opt_err_nsu=1;
    netstat_opt_nsu[0]='\0';
    while (netstat_opt_err_nsu == 1)
    {

```



```

printf("\n*** please select address family for function 1 ***\n\n");
printf("    1. AF_INET address family\n");
printf("    2. AF_UNIX address family\n");
printf("\n\n Please select one ==> ");
gets(netstat_opt_nsu);
printf("\n");
netstat_opt_err_nsu=0;
if (strlen(netstat_opt_nsu) > 0)
{
    if (strcmp(netstat_opt_nsu,"1") != 0 && strcmp(netstat_opt_nsu,"2") != 0)
    {
        printf("\n!!! Address family selection is error !!!\n");
        netstat_opt_err_nsu=1;
    }
}
}
if (strlen(netstat_opt_nsu) > 0)
{
    strcat(netstat_cmd_nsu,"-f ");
    if (strcmp(netstat_opt_nsu,"1") == 0)
    {
        strcat(netstat_cmd_nsu,"inet");
    }
    else
    {
        if (strcmp(netstat_opt_nsu,"2") == 0)
        {
            strcat(netstat_cmd_nsu,"unix");
        }
    }
    strcat(netstat_cmd_nsu," ");
}
break;

case 2 :
    netstat_opt_nsu[0]='\0';
    printf("\n*** please select options for function 2 ***\n");
    printf("    Network Data Structure/Rounting Table\n");
    printf("\n*** You want to show network addresses as numbers (Y/N) ");
    confirm_flag_nsu=confirm_continue();
    if (confirm_flag_nsu == 1)
    {
        strcat(netstat_cmd_nsu,"-n ");
    }

```

```

    }
    netstat_opt_nsu[0]='\0';
    printf("\n*** You want to show per-protocol statistics (Y/N) ");
    confirm_flag_nsu=confirm_continue();
    if (confirm_flag_nsu == 1)
    {
        strcat(netstat_cmd_nsu,"-s ");
    }
    netstat_opt_nsu[0]='\0';
    netstat_opt_err_nsu=1;
    while (netstat_opt_err_nsu == 1)
    {
        printf("\n*** please select one option for ");
        printf("function 2 ***\n\n");
        printf("    1. Show the statistics recorded by management routines\n");
        printf("        for the network's private buffer pool\n");
        printf("    2. Show the state of auto-configured interfaces\n");
        printf("    3. Show the routing table\n");
        printf("\n\n Please select one ==> ");
        gets(netstat_opt_nsu);
        printf("\n");
        netstat_opt_err_nsu=0;
        if (strlen(netstat_opt_nsu) > 0)
        {
            if (strcmp(netstat_opt_nsu,"1") != 0 && strcmp(netstat_opt_nsu,"2") != 0 &&
                strcmp(netstat_opt_nsu,"3") != 0)
            {
                printf("\n!!! Netstat option selection is error !!!\n");
                netstat_opt_err_nsu=1;
            }
            else
            {
                if (strcmp(netstat_opt_nsu,"1") == 0)
                {
                    strcat(netstat_cmd_nsu,"-m ");
                }
                else
                {
                    if (strcmp(netstat_opt_nsu,"2") == 0)
                    {
                        strcat(netstat_cmd_nsu,"-i ");
                    }
                    else

```

```

        {
            if (strcmp(netstat_opt_nsu,"2") == 0)
            {
                strcat(netstat_cmd_nsu,"-r ");
            }
            else
            {
                printf("\n!!! Netstat option selection is error !!!\n");
            }
        }
    }
}

netstat_opt_err_nsu=1;
netstat_opt_nsu[0]='\0';
while (netstat_opt_err_nsu == 1)
{
    printf("\n*** please select address family for function 2 ***\n\n");
    printf("    1. AF_INET address family\n");
    printf("    2. AF_UNIX address family\n");
    printf("\n\n Please select one ==> ");
    gets(netstat_opt_nsu);
    printf("\n");
    netstat_opt_err_nsu=0;
    if (strlen(netstat_opt_nsu) > 0)
    {
        if (strcmp(netstat_opt_nsu,"1") != 0 && strcmp(netstat_opt_nsu,"2") != 0)
        {
            printf("\n!!! Address family selection is error !!!\n");
            netstat_opt_err_nsu=1;
        }
    }
}

if (strlen(netstat_opt_nsu) > 0)
{
    strcat(netstat_cmd_nsu,"-f ");
    if (strcmp(netstat_opt_nsu,"1") == 0)
    {
        strcat(netstat_cmd_nsu,"inet");
    }
    else
    {

```

```

        if (strcmp(netstat_opt_nsu,"2") == 0)
        {
            strcat(netstat_cmd_nsu,"unix");
        }
    }
    strcat(netstat_cmd_nsu," ");
}
break;

case 3 :
    netstat_opt_nsu[0]='\0';
    printf("\n*** please select options for function 3 ***\n");
    printf("    Cumulative Traffic Statistics\n");
    printf("\n*** You want to show network addresses as numbers (Y/N) ");
    confirm_flag_nsu=confirm_continue();
    if (confirm_flag_nsu == 1)
    {
        strcat(netstat_cmd_nsu,"-n ");
    }
    netstat_opt_nsu[0]='\0';
    printf("\n*** You want to show the specific interface information (Y/N) ");
    confirm_flag_nsu=confirm_continue();
    if (confirm_flag_nsu == 1)
    {
        netstat_opt_err_nsu=1;
        while (netstat_opt_err_nsu == 1)
        {
            netstat_opt_nsu[0]='\0';
            printf("\n*** Please input the interface name ==> ");
            gets(netstat_opt_nsu);
            printf("\n");
            if (strlen(netstat_opt_nsu) < 1)
            {
                netstat_opt_err_nsu=1;
            }
            else
            {
                netstat_opt_err_nsu=0;
                strcat(netstat_cmd_nsu,"-I ");
                strcat(netstat_cmd_nsu,netstat_opt_nsu);
            }
        }
    }
}

```

```

while (seconds_interval_nsu < 1)
{
    printf("\n*** Please input the number of second\n");
    printf("    in which to gather statistics between displays ==> ");
    scanf("%d",&seconds_interval_nsu);
    getchar();
    printf("\n");
}
break ;

default :
    printf("\n Function selection is error, please check it !!!\n");
    break;
}

netstat_opt_nsu[0]='\0';
printf("\n*** Please input the system argument (default value : /vmunix) ==> ");
gets(netstat_opt_nsu);
printf("\n");
if (strlen(netstat_opt_nsu) > 0)
{
    strcat(netstat_cmd_nsu,netstat_opt_nsu);
    strcat(netstat_cmd_nsu," ");
}
netstat_opt_nsu[0]='\0';
printf("\n*** Please input the core argument (default value : /dev/kmem) ==> ");
gets(netstat_opt_nsu);
printf("\n");
if (strlen(netstat_opt_nsu) > 0)
{
    strcat(netstat_cmd_nsu,netstat_opt_nsu);
    strcat(netstat_cmd_nsu," ");
}
netstat_cmd_nsu[strlen(netstat_cmd_nsu)]='\0';
}
if (netstat_batch_exit_nsu == 0)
{
    printf("\n\n!!! Please wait for the Network Status Utility process !!!\n\n");
    system(netstat_cmd_nsu);
}
printf("\n*** Do you want to execute the Network Status Utility again (Y/N) ");
continue_flag_nsu=confirm_continue();
}

```

```
}
```

```
/* This procedure will generate the I/O statistics report for users.  
   Users can set the different parameters to get the I/O statistics  
   report which they want. */
```

```
io_statistics_report_utility()  
{  
    int continue_flag_isru, batch_mode_isru, iostat_opt_err_isru;  
    int char_index_isru, disk_limit_flag_isru;  
    int included_disk_number_err_flag_isru, get_count_flag_isru;  
    char iostat_cmd_isru[256], iostat_batch_cmd_isru[256];  
    char iostat_opt_isru[7];  
    char included_disk_number_isru[34], interval_s_isru[34], count_s_isru[34];  
    char specified_disks_isru[256];  
    long interval_isru, count_isru;  
  
    continue_flag_isru=1;  
    while (continue_flag_isru == 1)  
    {  
        iostat_cmd_isru[0]='\0';  
        printf("\n\n*** Welcome to ");  
        printf("the I/O statistics report Utility ***\n\n");  
        batch_mode_isru=select_execution_mode();  
        if (batch_mode_isru == 1)  
        {  
            iostat_batch_cmd_isru[0]='\0';  
            printf("\n!!! Type <exit> or <quit> to exit ");  
            printf("the I/O statistics utility batch mode !!!\n");  
            while ((strcmp(iostat_batch_cmd_isru, "exit")) != 0 &&  
                (strcmp(iostat_batch_cmd_isru, "quit")) != 0)  
            {  
                printf("\niostat command > ");  
                gets(iostat_batch_cmd_isru);  
                if ((strcmp(iostat_batch_cmd_isru, "exit")) != 0 &&  
                    (strcmp(iostat_batch_cmd_isru, "quit")) != 0)  
                {  
                    if (strncmp(iostat_batch_cmd_isru, "iostat", 6) == 0)  
                    {  
                        system(iostat_batch_cmd_isru);  
                    }  
                }  
            }  
            else  
            {  
                continue_flag_isru=0;  
            }  
        }  
    }  
}
```

```

        printf("\n!!! Unknown command");
        printf("(not iostat command) !!\n");
    }
}
printf("\n!!! Exit the iostat utility batch mode execution !!\n\n");
getchar();
}
else
{
    strcpy(iostat_cmd_isru,"iostat ");
    iostat_opt_err_isru=1;
    while (iostat_opt_err_isru == 1)
    {
        printf("\n*** please select options for I/O statistics ");
        printf("report utility ***\n\n");
        printf(" ");
        printf("c- Report percentage of time the syatem has ");
        printf("spent in user mode\n");
        printf(" ");
        printf("d- For each disk, report\n");
        printf(" ");
        printf("the number of Kbytes transferred per second,\n");
        printf(" ");
        printf("the number of transfers per second,\n");
        printf(" ");
        printf("the milliseconds per average seek\n");
        printf(" ");
        printf("D- For each disk, report\n");
        printf(" ");
        printf("the reads/writes per second,\n");
        printf(" ");
        printf("the percentage of disk utilization\n");
        printf(" ");
        printf("I- Report the counts in each interval\n");
        printf(" ");
        printf("t- Report the read/write character number ");
        printf("to terminal\n");
        printf("\n\n Please select them ==> [cdDIIt] ");
        gets(iostat_opt_isru);
        printf("\n");
        iostat_opt_err_isru=0;
        char_index_isru=0;
    }
}

```

```

/* Check the options of the ping utility,
   If the users input error option,
   then the users need to input the options again */
while (char_index_isru < strlen(iostat_opt_isru) && iostat_opt_err_isru != 1)
{
    if (iostat_opt_isru[char_index_isru] != 'c' &&
        iostat_opt_isru[char_index_isru] != 'd' &&
        iostat_opt_isru[char_index_isru] != 'D' &&
        iostat_opt_isru[char_index_isru] != 'T' &&
        iostat_opt_isru[char_index_isru] != 't')
    {
        printf("\n!!! Iostat option selection is ");
        printf("error !!!\n");
        iostat_opt_err_isru=1;
    }
    char_index_isru++;
}
}
if (strlen(iostat_opt_isru) > 0)
{
    strcat(iostat_cmd_isru, "-");
    strcat(iostat_cmd_isru, iostat_opt_isru);
    strcat(iostat_cmd_isru, " ");
}
printf("*** Do you want to limit the number of disk \n");
printf("    included in the report (Y/N) ==> \n");
disk_limit_flag_isru=confirm_continue();
included_disk_number_err_flag_isru=1;
while (disk_limit_flag_isru == 1 && included_disk_number_err_flag_isru == 1)
{
    strcat(iostat_cmd_isru, "-l ");
    included_disk_number_isru[0]='\0';
    printf("*** Please input the included disk number(default : 4) ==> ");
    gets(included_disk_number_isru);
    printf("\n");
    if (included_disk_number_isru == NULL ||
        strlen(included_disk_number_isru) <= 0)
    {
        strcat(iostat_cmd_isru, "4 ");
    }
}
else
{

```



```

        if (atol(included_disk_number_isru) > 0)
        {
            strcat(iostat_cmd_isru,included_disk_number_isru);
            strcat(iostat_cmd_isru," ");
            included_disk_number_err_flag_isru=0;
        }
        else
        {
            included_disk_number_err_flag_isru=1;
            printf("!!! The included disk number input ");
            printf("error !!!\n");
            printf("!!! Please input it again !!!\n");
        }
    }
}

printf("*** Please explicitly specify the disks to be reported ==> \n");
specified_disks_isru[0]='\0';
gets(specified_disks_isru);
if (strlen(specified_disks_isru) > 0)
{
    strcat(iostat_cmd_isru,specified_disks_isru);
    strcat(iostat_cmd_isru," ");
}

interval_isru=-1;
while (interval_isru < 0)
{
    interval_s_isru[0]='\0';
    printf("*** Please set once each interval seconds ==> (default : 1 second)\n");
    gets(interval_s_isru);
    if (interval_s_isru != NULL)
    {
        if (strlen(interval_s_isru) > 0)
        {
            interval_isru=atol(interval_s_isru);
            if (interval_isru > 0)
            {
                strcat(iostat_cmd_isru,interval_s_isru);
                strcat(iostat_cmd_isru," ");
                get_count_flag_isru=1;
            }
        }
        else
        {
            printf("!!! Interval input error !!!\n");
        }
    }
}

```

```

        printf("!!! Please input it again !!!\n");
        interval_isru=-1;
    }
}
else
{
    interval_isru=0;
    get_count_flag_isru=0;
}
}
else
{
    interval_isru=0;
    get_count_flag_isru=0;
}
}
count_isru=-1;
while (count_isru < 0 && get_count_flag_isru == 1)
{
    count_s_isru[0]='\0';
    printf("*** Please set count reports");
    printf(" ==> \n");
    gets(count_s_isru);
    if (count_s_isru != NULL)
    {
        if (strlen(count_s_isru) > 0)
        {
            count_isru=atol(count_s_isru);
            if (count_isru > 0)
            {
                strcat(iostat_cmd_isru,count_s_isru);
            }
        }
        else
        {
            printf("!!! Count input error !!!\n");
            printf("!!! Please input it again !!!\n");
            count_isru=-1;
        }
    }
}
else
{
    count_isru=0;
}
}

```

```

        }
        else
        {
            count_isru=0;
        }
    }
    iostat_cmd_isru[strlen(iostat_cmd_isru)]='\0';
    system(iostat_cmd_isru);
}
printf("\n*** Do you want to execute the I/O statistics report ");
printf("utility again (Y/N) ");
continue_flag_isru=confirm_continue();
}
}

```

/* This procedure will generate the virtual memory statistics report for users. Users can set the different parameters to get the virtual memory statistics report which they want. */

```

vm_statistics_report_utility()
{
    int continue_flag_vsru, batch_mode_vsru, vmstat_opt_err_vsru;
    int get_count_flag_vsru;
    char vmstat_cmd_vsru[256], vmstat_batch_cmd_vsru[256];
    char vmstat_opt_vsru[7];
    char interval_s_vsru[34], count_s_vsru[34];
    long interval_vsru, count_vsru;

    continue_flag_vsru=1;
    while (continue_flag_vsru == 1)
    {
        vmstat_cmd_vsru[0]='\0';
        printf("\n\n*** Welcome to ");
        printf("the virtual memory statistics report Utility ***\n\n");
        batch_mode_vsru=select_execution_mode();
        if (batch_mode_vsru == 1)
        {
            vmstat_batch_cmd_vsru[0]='\0';
            printf("\n!!! Type <exit> or <quit> to exit ");
            printf("the virtual statistics utility batch mode !!!\n");
            while ((strcmp(vmstat_batch_cmd_vsru, "exit")) != 0 &&
                (strcmp(vmstat_batch_cmd_vsru, "quit")) != 0)

```

```

{
printf("\nvmstat command > ");
gets(vmstat_batch_cmd_vsru);
if ((strcmp(vmstat_batch_cmd_vsru,"exit")) != 0 &&
    (strcmp(vmstat_batch_cmd_vsru,"quit")) != 0)
{
    if (strcmp(vmstat_batch_cmd_vsru,"vmstat",6) == 0)
    {
        system(vmstat_batch_cmd_vsru);
    }
    else
    {
        printf("\n!!! Unknown command (not vmstat command) !!!\n");
    }
}
}
printf("\n!!! Exit the vmstat utility batch mode ");
printf("execution !!!\n\n");
getchar();
}
else
{
strcpy(vmstat_cmd_vsru,"vmstat ");
vmstat_opt_err_vsru=1;
while (vmstat_opt_err_vsru == 1)
{
printf("\n*** please select options for virtual memory ");
printf("statistics report utility *** \n\n");
printf(" ");
printf("f- Report on \n");
printf(" ");
printf("the number of forks and vforks");
printf("since system startup, \n");
printf(" ");
printf("the number of pages of virtual memory involved \n");
printf(" ");
printf("in each kind of fork, \n");
printf(" ");
printf("i- Report the number of interrupts per device \n");
printf(" ");
printf("s- Display the contents of the sum structure \n");
printf(" ");
printf("S- Report on swapping activity ");

```

```

printf("\n\n Please select one ==> ");
gets(vmstat_opt_vsr_u);
printf("\n");
vmstat_opt_err_vsr_u=0;
if (strlen(vmstat_opt_vsr_u) > 1)
{
    vmstat_opt_err_vsr_u=1;
}
else
{
    /* Check the options of the ping utility,
    If the users input error option,
    then the users need to input the options again */
    if (vmstat_opt_vsr_u[0] != 'f' && vmstat_opt_vsr_u[0] != 'i' &&
        vmstat_opt_vsr_u[0] != 's' && vmstat_opt_vsr_u[0] != 'S')
    {
        printf("\n!!! Vmstat option selection is error !!!\n");
        vmstat_opt_err_vsr_u=1;
    }
}
}
if (strlen(vmstat_opt_vsr_u) > 0)
{
    strcat(vmstat_cmd_vsr_u, "-");
    strcat(vmstat_cmd_vsr_u, vmstat_opt_vsr_u);
    strcat(vmstat_cmd_vsr_u, " ");
}
interval_vsr_u=-1;
while (interval_vsr_u < 0)
{
    interval_s_vsr_u='\0';
    printf("**** Please set once each interval seconds ==> \n");
    gets(interval_s_vsr_u);
    if (strlen(interval_s_vsr_u) > 0)
    {
        if (interval_s_vsr_u != NULL)
        {
            interval_vsr_u=atol(interval_s_vsr_u);
            if (interval_vsr_u > 0)
            {
                strcat(vmstat_cmd_vsr_u, interval_s_vsr_u);
                strcat(vmstat_cmd_vsr_u, " ");
                get_count_flag_vsr_u=1;
            }
        }
    }
}

```

```

    }
    else
    {
        printf("!!! Interval input error !!!\n");
        printf("!!! Please input it again !!!\n");
        interval_vsru=-1;
    }
}
else
{
    interval_vsru=0;
    get_count_flag_vsru=0;
}
}
else
{
    interval_vsru=0;
    get_count_flag_vsru=0;
}
}
count_vsru=-1;
while (count_vsru < 0 && get_count_flag_vsru == 1)
{
    count_s_vsru[0]='\0';
    printf("*** Please set count reports");
    printf(" ==> \n");
    gets(count_s_vsru);
    if (count_s_vsru != NULL)
    {
        if (strlen(count_s_vsru) > 0)
        {
            count_vsru=atol(count_s_vsru);
            if (count_vsru > 0)
            {
                strcat(vmstat_cmd_vsru,count_s_vsru);
            }
        }
        else
        {
            printf("!!! Count input error !!!\n");
            printf("!!! Please input it again !!!\n");
            count_vsru=-1;
        }
    }
}

```

```

        else
        {
            count_vsru=0;
        }
    }
    else
    {
        count_vsru=0;
    }
}
vmstat_cmd_vsru[strlen(vmstat_cmd_vsru)]='\0';
system(vmstat_cmd_vsru);
}
printf("\n*** Do you want to execute the virtual
memory statistics report utility again (Y/N) ");
continue_flag_vsru=confirm_continue();
}

}

/* This procedure will let users use the 'unix command' */
unix_command_execution()
{
    char command_uce[256];

    command_uce[0]='\0';
    printf("\n!!! Type <exit> or <quit> to exit the unix command !!!\n");
    while ((strcmp(command_uce,"exit")) != 0 &&
           (strcmp(command_uce,"quit")) != 0)
    {
        printf("\nunix command > ");
        gets(command_uce);
        if ((strcmp(command_uce,"exit")) != 0 &&
            (strcmp(command_uce,"quit")) != 0)
        {
            system(command_uce);
        }
    }
    printf("\n!!! Exit the unix command execution !!!\n\n");
    getchar();
}

```

```

/* This procedure will let users select the execution mode :
   Batch mode or Interactive mode */
select_execution_mode()
{
    int execution_mode_sem, batch_mode_sem;

    execution_mode_sem=0;
    while (execution_mode_sem != 1 && execution_mode_sem != 2)
    {
        printf("\n*** please select the execution mode ");
        printf(" ***\n\n");
        printf("    1. BATCH mode\n");
        printf("    2. INTERACTIVE mode\n");
        printf("\n\n Please select one ==> ");
        scanf("%d",&execution_mode_sem);
        getchar();
        printf("\n");
        if (execution_mode_sem == 1)
        {
            batch_mode_sem=1;
        }
        else
        {
            if (execution_mode_sem == 2)
            {
                batch_mode_sem=0;
            }
            else
            {
                printf("\n!!! Execution mode selection is error");
                printf(" !!!\n");
            }
        }
    }
    return batch_mode_sem;
}

```

```

/* Tools for File Transfer Measurement */
file_transfer_measurement_tools()

```



```

{

FILE *ftp_data_fmt;
char cur_path_fmt[255],title_fmt[81];
char ftp_data_filename_fmt[45];
int simple_num_fmt,data_without_io_fmt,cur_path_len_fmt;
long time_hour_interval_range_fmt;
int opt_fun_fmt,chdir_flag_fmt;
int exit_flag_fmt,continue_flag_fmt;

/* Get the parameters for the file transfer measurement tool */
opt_fun_fmt=0;
exit_flag_fmt=0;
while (opt_fun_fmt < 1 || opt_fun_fmt > 5 || exit_flag_fmt == 0)
{
continue_flag_fmt=0;
opt_fun_fmt=display_file_transfer_measurement_menu();
switch (opt_fun_fmt)
{
case 1 :
printf("==>> Do you really want to execute this ");
printf("function (Y/N) ? ");
continue_flag_fmt=confirm_continue();
if (continue_flag_fmt == 1)
{
file_transfer();
}
break;

case 2 :
printf("==>> Do you really want to execute this ");
printf("function (Y/N) ? ");
continue_flag_fmt=confirm_continue();
if (continue_flag_fmt == 1)
{
statistic_report();
}
break;

case 3 :
printf("==>> Do you really want to execute this ");
printf("function (Y/N) ? ");
continue_flag_fmt=confirm_continue();

```

```

        if (continue_flag_fmt == 1)
        {
            file_utility();
        }
        break;

    case 4 :
        printf("\n Do you really want to exit ? (Y/N) ");
        exit_flag_fmt=confirm_continue();
        break;

    default :
        printf("\n Function selection is error, ");
        printf("please check it !!!\n");
        break;
    }
}
printf("\n\n!!! Exit file transfer measurement tools !!!");
getchar();

}

/* This function will display the menu for user to select
   the function of the file transfer measurement */
int display_file_transfer_measurement_menu()
{
    int opt_fun_dftmm;

    printf("\n\n\n <<<<< File Transfer Measurement Tools Main Menu");
    printf(" >>>>> \n\n");
    printf("      1. File Transfer Measurement \n\n");
    printf("      2. Measurement Statistic Data Report \n\n");
    printf("      3. Data File Maintenance Utility \n\n");
    printf("      4. Exit \n\n\n");
    printf("      Please select one function (1-4) ==>> ");
    scanf("%d",&opt_fun_dftmm);
    getchar();
    printf("\n");
    return opt_fun_dftmm;
}

```

```

file_transfer()
{
    int sockfd_t_ft,sockfd_u_ft;
    char serv_host_addr_ft[256];
    char ftp_file_name_with_path_ft[256],ftp_file_name_ft[45];
    char ftp_info_name_ft[20],ftp_file_size_ft[7];
    char measurement_time_s_ft[34];
    int n_ft,protocol_ft,m_size_ft,f_size_ft;
    int serv_host_addr_err_ft,socket_err_ft;
    int msg_size_ft,measurement_time_ft,measurement_ctr_ft;
    struct sockaddr_in serv_addr_u_ft;
    int file_head_ft,stat_res_ft,cur_path_len_ft;
    long file_len_ft;
    char file_len_s_ft[34],cur_path_ft[256],chmod_string_ft[256];
    struct stat f_stat_ft;
    struct timeval read_file_time_ft;
    FILE *ftp_file_ft;
    FILE *ftp_info_ft;

    cur_path_ft[0]='\0';
    ftp_file_name_with_path_ft[0]='\0';
    ftp_file_name_ft[0]='\0';
    ftp_info_name_ft[0]='\0';
    serv_host_addr_ft[0]='\0';
    file_len_ft=0;
    file_len_s_ft[0]='\0';
    serv_host_addr_err_ft=0;
    measurement_ctr_ft=1;

    get_parameters(serv_host_addr_ft,ftp_file_name_ft,
                  ftp_file_size_ft,&serv_host_addr_err_ft,
                  &protocol_ft,&m_size_ft,
                  &measurement_time_ft);
    if (serv_host_addr_err_ft == 0)
    {
        if (getcwd(cur_path_ft,255) == NULL)
        {
            printf("\n !!! Can not get the current directory !!! \n");
        }
        else
        {
            if (protocol_ft == 1)
            {

```

```

        strcat(ftp_info_name_ft,"TCP_");
    }
    else
    {
        strcat(ftp_info_name_ft,"UDP_");
    }
    if (protocol_ft == 1)
    {
        switch(m_size_ft)
        {
            case 1 :
                strcat(ftp_info_name_ft,"1024");
                msg_size_ft=1024;
                break;

            case 2 :
                strcat(ftp_info_name_ft,"512");
                msg_size_ft=512;
                break;

            case 3 :
                strcat(ftp_info_name_ft,"256");
                msg_size_ft=256;
                break;

            default :
                break;

        }
    }
    else
    {
        switch(m_size_ft)
        {
            case 1 :
                strcat(ftp_info_name_ft,"4096");
                msg_size_ft=4096;
                break;

            case 2 :
                strcat(ftp_info_name_ft,"2048");
                msg_size_ft=2048;
                break;

```

```

        case 3 :
            strcat(ftp_info_name_ft,"1024");
            msg_size_ft=1024;
            break;

        case 4 :
            strcat(ftp_info_name_ft,"512");
            msg_size_ft=512;
            break;

        case 5 :
            strcat(ftp_info_name_ft,"256");
            msg_size_ft=256;
            break;

        case 6 :
            strcat(ftp_info_name_ft,"128");
            msg_size_ft=128;
            break;

        default :
            break;
    }
}
strcat(ftp_file_name_with_path_ft,cur_path_ft);
strcat(ftp_file_name_with_path_ft,"/ftp_file/");
strcat(ftp_info_name_ft,".");
strcat(ftp_file_name_with_path_ft,ftp_file_name_ft);
strcat(ftp_info_name_ft,ftp_file_size_ft);
ftp_file_name_with_path_ft[strlen(ftp_file_name_with_path_ft)]='\0';
ftp_info_name_ft[strlen(ftp_info_name_ft)]='\0';
file_head_ft=open(ftp_file_name_with_path_ft,O_RDONLY);
if (file_head_ft == -1)
{
    printf("\n !!! Can not open the file : %s !!!\n",ftp_file_name_with_path_ft);
    close(file_head_ft);
}
else
{
    stat_res_ft=fstat(file_head_ft,&f_stat_ft);
    if (stat_res_ft == 0)
    {

```

```

    sprintf(file_len_s_ft,"%ld",f_stat_ft.st_size);
    file_len_ft=f_stat_ft.st_size;
}
else
{
    strcpy(file_len_s_ft,"0");
}
close(file_head_ft);
ftp_file_ft=fopen(ftp_file_name_with_path_ft,"r");
if (chdir(serv_host_addr_ft) == -1)
{
    if (mkdir(serv_host_addr_ft) == -1)
    {
        printf("\n !!! %s subdirectory create error !!! \n");
    }
    else
    {
        strcpy(chmod_string_ft,"chmod 777 ");
        strcat(chmod_string_ft,serv_host_addr_ft);
        system(chmod_string_ft);
        chdir(serv_host_addr_ft);
    }
}
if ((ftp_info_ft=fopen(ftp_info_name_ft,"r")) == NULL)
{
    ftp_info_ft=fopen(ftp_info_name_ft,"w");
    fprintf(ftp_info_ft,"%s",ftp_info_name_ft);
}
else
{
    fclose(ftp_info_ft);
    ftp_info_ft=fopen(ftp_info_name_ft,"a");
}
if (protocol_ft == 1)
{
    while (measurement_time_ft > 0)
    {
        read_file_time_ft.tv_sec=0;
        read_file_time_ft.tv_usec=0;
        socket_err_ft=0;
        printf("\n ==>> Measurement times is : %d \n",measurement_ctr_ft);
        open_tcp_socket(serv_host_addr_ft,&sockfd_t_ft,&socket_err_ft);
        fseek(ftp_file_ft,0L,0);
    }
}

```

```

if (socket_err_ft == 0)
{
    tcp_cli(ftp_file_name_ft,&sockfd_t_ft,msg_size_ft,
            file_len_s_ft,ftp_file_ft,&read_file_time_ft);
    tcp_summary(&sockfd_t_ft,ftp_info_name_ft,ftp_info_ft,
                read_file_time_ft);
}
else
{
    printf("!!! TCP socket open or connection error !!!\n");
}
measurement_ctr_ft++;
measurement_time_ft--;
close(sockfd_t_ft);
}
}
else
{
    while (measurement_time_ft > 0)
    {
        read_file_time_ft.tv_sec=0;
        read_file_time_ft.tv_usec=0;
        socket_err_ft=0;
        printf("\n ==>> Measurement times is : %d\n",measurement_ctr_ft);
        open_udp_socket(serv_host_addr_ft,&sockfd_u_ft,
                        &serv_addr_u_ft,&socket_err_ft);
        fseek(ftp_file_ft,0L,0);
        if (socket_err_ft == 0)
        {
            udp_cli(ftp_file_name_ft,&sockfd_u_ft,msg_size_ft,
                    (struct sockaddr *) &serv_addr_u_ft,
                    sizeof(serv_addr_u_ft),file_len_s_ft,
                    ftp_file_ft,&read_file_time_ft);
            udp_summary(&sockfd_u_ft,ftp_info_name_ft,
                        (struct sockaddr *) &serv_addr_u_ft,
                        sizeof(serv_addr_u_ft),ftp_info_ft,
                        read_file_time_ft);
        }
    }
    else
    {
        printf("!!! UDP socket open or connection error");
        printf(" !!!\n");
    }
}

```

```

        measurement_ctr_ft++;
        measurement_time_ft--;
        close(sockfd_u_ft);
    }
}
fclose(ftp_info_ft);
fclose(ftp_file_ft);
cur_path_len_ft=strlen(cur_path_ft);
cur_path_ft[cur_path_len_ft]='\0';
chdir(cur_path_ft);
}
}
}
}

```

```

get_parameters(serv_host_addr_gp,ftp_file_name_gp,
               ftp_file_size_gp,serv_host_addr_err_gp,
               protocol_gp,m_size_gp,measurement_time_gp)

```

```

char *serv_host_addr_gp;
char *ftp_file_name_gp;
char *ftp_file_size_gp;
int *serv_host_addr_err_gp;
int *protocol_gp;
int *m_size_gp;
int *measurement_time_gp;

{
    int opt_buf_gp,try_again_gp;
    unsigned long inaddr_gp;
    struct hostent *dummy_ptr_gp;
    /* this is a dummy variable for the convert_host_name_to_address
       procedure in this procedure. */

    /* initialize the protocol parameters */
    serv_host_addr_gp[0]='\0';
    *protocol_gp=0;
    *m_size_gp=0;
    *measurement_time_gp=0;
    try_again_gp=0;

```



```

while (strlen(serv_host_addr_gp) == 0 && try_again_gp <= 3)
{
    printf("\n *** Please input the symbolic name/internet address of server ==> ");
    scanf("%s",serv_host_addr_gp);
    if ((inaddr_gp = inet_addr(serv_host_addr_gp)) == INADDR_NONE)
    {
        convert_host_name_to_address(serv_host_addr_gp,&dummy_ptr_gp);
    }
    else
    {
        convert_host_address_information(serv_host_addr_gp,&dummy_ptr_gp);
    }
    if (strlen(serv_host_addr_gp) == 0)
    {
        *serv_host_addr_err_gp=1;
    }
    else
    {
        *serv_host_addr_err_gp=0;
    }
    try_again_gp=try_again_gp+1;
}
if (*serv_host_addr_err_gp == 0)
{
    while (*protocol_gp != 1 && *protocol_gp != 2)
    {
        printf("\n *** Please select the protocol for measurement *** \n");
        printf("\n 1. TCP ");
        printf("\n 2. UDP ");
        printf("\n\n Please select one ==> ");
        scanf("%d",&opt_buf_gp);
        *protocol_gp=opt_buf_gp;
        if (*protocol_gp != 1 && *protocol_gp != 2)
        {
            printf("!!! Protocol selection is error !!!\n");
        }
    }
    if (*protocol_gp == 1)
    {
        while (*m_size_gp != 1 && *m_size_gp != 2 && *m_size_gp != 3)
        {
            printf("\n *** Please select the message buffer ");

```

```

printf("size ");
printf("for measurement *** \n\n");
printf("\n 1. 1024 bytes ");
printf("\n 2. 512 bytes ");
printf("\n 3. 256 bytes ");
printf("\n\n Please select one ==> ");
scanf("%d",&opt_buf_gp);
*m_size_gp=opt_buf_gp;
if (*m_size_gp != 1 && *m_size_gp != 2 && *m_size_gp != 3)
{
printf("!!! Message buffer size selection is error !!!\n");
}
}
}
if (*protocol_gp == 2)
{
while (*m_size_gp != 1 && *m_size_gp != 2 && *m_size_gp != 3
&& *m_size_gp != 4 && *m_size_gp != 5 && *m_size_gp != 6)
{
printf("\n *** Please select the message buffer ");
printf("size for measurement *** \n\n");
printf("\n 1. 4096 bytes ");
printf("\n 2. 2048 bytes ");
printf("\n 3. 1024 bytes ");
printf("\n 4. 512 bytes ");
printf("\n 5. 256 bytes ");
printf("\n 6. 128 bytes ");
printf("\n\n Please select one ==> ");
scanf("%d",&opt_buf_gp);
*m_size_gp=opt_buf_gp;
if (*m_size_gp != 1 && *m_size_gp != 2 && *m_size_gp != 3 &&
*m_size_gp != 4 && *m_size_gp != 5 && *m_size_gp != 6)
{
printf("!!! Message buffer size selection is error !!!\n");
}
}
}
select_ftp_file_size(ftp_file_name_gp,ftp_file_size_gp);
if (strlen(ftp_file_name_gp) == 0 || strlen(ftp_file_size_gp) == 0)
{
printf("!!! No transferred file be selected, exit file transfer function !!!\n");
*measurement_time_gp=0;
}

```

```

else
{
    while (*measurement_time_gp <= 0)
    {
        printf("\n *** Please input the measurement times that you want ==> ")
        scanf("%d",&opt_buf_gp);
        *measurement_time_gp=opt_buf_gp;
        if (*measurement_time_gp <= 0)
        {
            printf("!!! Measurement time selection is error !!!\n");
        }
    }
}
else
{
    printf("!!! Server host name(address) input error !!!\n");
    printf("!!! Please check the name or address !!!\n");
}
}

```

```

/* typedef struct hostent *hostent_ptr; */

```

```

/* Convert the host name to a dotted-decimal number
   by the gethostbyname() */
convert_host_name_to_address(serv_host_addr_chnta,host_info_chnta)

```

```

char *serv_host_addr_chnta;
struct hostent **host_info_chnta;

```

```

{
    struct in_addr *host_addr_chnta;
    struct hostent *host_ptr_chnta;

    if ((host_ptr_chnta=gethostbyname(serv_host_addr_chnta)) == NULL)
    {
        printf("!!! the host name error : %s,",serv_host_addr_chnta);
        printf(" please input the correct name. !!!\n");
        serv_host_addr_chnta[0]='\0';
    }
}

```

```

else
{
    *host_info_chnta=host_ptr_chnta;
    host_addr_chnta=((struct in_addr *) host_ptr_chnta->h_addr_list[0]);
    strcpy(serv_host_addr_chnta,inet_ntoa(host_addr_chnta));
}

}

/* Convert the host address to official name and the server
   address is correct or not by the gethostbyaddr()          */
convert_host_address_information(serv_host_addr_cati,host_info_cati)

char *serv_host_addr_cati;
struct hostent **host_info_cati;

{
    struct hostent *host_ptr_cati;
    struct sockaddr_in host_addr_cati;

    host_addr_cati.sin_port=ntohs((u_short) host_addr_cati.sin_port);
    host_addr_cati.sin_addr.s_addr=inet_addr(serv_host_addr_cati);
    if ((host_ptr_cati=gethostbyaddr(host_addr_cati.sin_addr,sizeof(host_ptr_cati),
                                     AF_INET)) == NULL)
    {
        printf("!!! the host address error : %s,",
               serv_host_addr_cati);
        printf(" please input the correct address/name !!!\n");
        serv_host_addr_cati[0]='\0';
        *host_info_cati=NULL;
    }
    else
    {
        *host_info_cati=host_ptr_cati;
    }
}

/* Select the size of the FTP file.
   This procedure will return the FTP file name which is
   correspond to the file size          */

```

```

select_ftp_file_size(ftp_file_name_sffs,ftp_file_size_sffs)

char *ftp_file_name_sffs;
char *ftp_file_size_sffs;

{
char cur_path_sffs[256],ftp_file_path_sffs[256];

cur_path_sffs[0]='\0';
ftp_file_path_sffs[0]='\0';
printf("\n!!! Please wait a second for the transferred file size ");
printf("proceeding !!!\n");
if (getcwd(cur_path_sffs,255) == NULL)
{
printf("\n !!! Can not get the current directory !!!\n");
ftp_file_name_sffs[0]='\0';
ftp_file_size_sffs[0]='\0';
}
else
{
strcat(ftp_file_path_sffs,cur_path_sffs);
strcat(ftp_file_path_sffs,"/ftp_file");
ftp_file_path_sffs[strlen(ftp_file_path_sffs)]='\0';
if (chdir(ftp_file_path_sffs) == -1)
{
ftp_file_name_sffs[0]='\0';
ftp_file_size_sffs[0]='\0';
printf("!!! Can not find the ftp_file subdirectory !!!\n");
printf("!!! Please check the ftp_file subdirectory !!!\n\n");
}
else
{
system("ls -l ftp_file.*MB > file.tmp");
system("ls -l ftp_file.*KB >> file.tmp");
file_selection(ftp_file_name_sffs,ftp_file_size_sffs,1);
if (chdir(cur_path_sffs) == -1)
{
ftp_file_name_sffs[0]='\0';
ftp_file_size_sffs[0]='\0';
printf("!!! Can not return the parent subdirectory from%s !!!\n",ftp_file_path_sffs);
}
}
}
}

```

```
}
```

```
file_selection(file_name_fs,file_size_or_file_no_fs,proc_id_fs)
```

```
char *file_name_fs;
```

```
char *file_size_or_file_no_fs;
```

```
int proc_id_fs;
```

```
{
```

```
FILE *file_tmp_fs;
```

```
char str1_fs[11],str2_fs[2],str3_fs[6],str4_fs[9];
```

```
char str5_fs[4],str6_fs[3],str7_fs[6],str8_fs[45];
```

```
char file_extension_fs[7];
```

```
int rec_ctr_fs,opt_buf_fs;
```

```
file_tmp_fs=fopen("file.tmp","r");
```

```
rec_ctr_fs=1;
```

```
opt_buf_fs=0;
```

```
while (!feof(file_tmp_fs))
```

```
{
```

```
    fscanf(file_tmp_fs,"%s %s %s %s %s %s %s %s\n",
```

```
        str1_fs,
```

```
        str2_fs,
```

```
        str3_fs,
```

```
        str4_fs,
```

```
        str5_fs,
```

```
        str6_fs,
```

```
        str7_fs,
```

```
        str8_fs);
```

```
    if (proc_id_fs == 1)
```

```
    {
```

```
        get_file_extension(str8_fs,file_extension_fs,file_size_or_file_no_fs);
```

```
        printf("\n    %d. ",rec_ctr_fs);
```

```
        printf("%s",file_extension_fs);
```

```
        printf("ytes ");
```

```
    }
```

```
    else
```

```
    {
```

```
        if (proc_id_fs == 2 || proc_id_fs == 3)
```

```
        {
```

```
            printf("\n    %d. ",rec_ctr_fs);
```

```

        printf("%s",str8_fs);
    }
}
rec_ctr_fs=rec_ctr_fs+1;
}
printf("\n\n Please select one, <%d> to exit ==> ",rec_ctr_fs);
while (opt_buf_fs == 0 || opt_buf_fs < 1 || opt_buf_fs > rec_ctr_fs)
{
    scanf("%d",&opt_buf_fs);
    getchar();
    if (opt_buf_fs < 1 || opt_buf_fs > rec_ctr_fs)
    {
        printf("!!! File selection is error !!!\n");
        printf("\n Please select one again ==> ");
    }
}
if (opt_buf_fs == rec_ctr_fs)
{
    printf("\n!!! Exit file selection, no file be selected !!!\n");
    file_name_fs[0]='\0';
    file_size_or_file_no_fs[0]='\0';
}
else
{
    rewind(file_tmp_fs);
    rec_ctr_fs=1;
    while (rec_ctr_fs < opt_buf_fs && (!feof(file_tmp_fs)))
    {
        fscanf(file_tmp_fs,"%s %s %s %s %s %s %s %s\n",
            str1_fs,
            str2_fs,
            str3_fs,
            str4_fs,
            str5_fs,
            str6_fs,
            str7_fs,
            str8_fs);
        rec_ctr_fs=rec_ctr_fs+1;
    }
    fscanf(file_tmp_fs,"%s %s %s %s %s %s %s %s\n",
        str1_fs,
        str2_fs,
        str3_fs,

```

```

        str4_fs,
        str5_fs,
        str6_fs,
        str7_fs,
        str8_fs);
strcpy(file_name_fs,str8_fs);
file_name_fs[strlen(file_name_fs)]='\0';
if (proc_id_fs == 1)
{
    get_file_extension(str8_fs,file_extension_fs,file_size_or_file_no_fs);
}
if (proc_id_fs == 3)
{
    sprintf(file_size_or_file_no_fs,"%d",opt_buf_fs);
}
}
fclose(file_tmp_fs);
system("rm file.tmp");
}

```

```

/* Get the file extension from the full file name string*/
get_file_extension(file_name_str_gfe,file_extension_gfe,file_size_gfe)

```

```

char *file_name_str_gfe;
char *file_extension_gfe;
char *file_size_gfe;

```

```

{
    int file_name_str_len_gfe;
    int char1_ctr_gfe,char2_ctr_gfe,char3_ctr_gfe;
    char char_buf_gfe;

```

```

    char1_ctr_gfe=9;
    char2_ctr_gfe=0;
    char3_ctr_gfe=0;
    char_buf_gfe='\0';
    file_name_str_len_gfe=strlen(file_name_str_gfe);
    while (char1_ctr_gfe < (file_name_str_len_gfe-2))
    {
        if (file_name_str_gfe[char1_ctr_gfe] >= '0' && file_name_str_gfe[char1_ctr_gfe] <=

```



```

'9')
{
    char_buf_gfe=file_name_str_gfe[char1_ctr_gfe];
    file_extension_gfe[char2_ctr_gfe]=char_buf_gfe;
    file_size_gfe[char3_ctr_gfe]=char_buf_gfe;
    char2_ctr_gfe=char2_ctr_gfe+1;
    char3_ctr_gfe=char3_ctr_gfe+1;
}
char1_ctr_gfe=char1_ctr_gfe+1;
}
file_extension_gfe[char2_ctr_gfe]=' ';
char2_ctr_gfe=char2_ctr_gfe+1;
char_buf_gfe=file_name_str_gfe[char1_ctr_gfe];
file_extension_gfe[char2_ctr_gfe]=char_buf_gfe;
file_size_gfe[char3_ctr_gfe]=char_buf_gfe;
char1_ctr_gfe=char1_ctr_gfe+1;
char2_ctr_gfe=char2_ctr_gfe+1;
char3_ctr_gfe=char3_ctr_gfe+1;
char_buf_gfe=file_name_str_gfe[char1_ctr_gfe];
file_extension_gfe[char2_ctr_gfe]=char_buf_gfe;
file_size_gfe[char3_ctr_gfe]=char_buf_gfe;
file_extension_gfe[char2_ctr_gfe+1]='\0';
file_size_gfe[char3_ctr_gfe+1]='\0';

}

/* This procedure will open a DATASTREAM socket for TCP protocol */
open_tcp_socket(serv_host_addr_ots,sockfd_ots,socket_err_ots)

char *serv_host_addr_ots;
int *sockfd_ots;
int *socket_err_ots;

{
    struct sockaddr_in cli_addr_ots,serv_addr_ots;

    /* Fill in the structure "serv_addr" with the address of the
       server that we want to send to. */
    bzero((char *) &serv_addr_ots, sizeof(serv_addr_ots));
    serv_addr_ots.sin_family = AF_INET;
    serv_addr_ots.sin_addr.s_addr = inet_addr(serv_host_addr_ots);
    serv_addr_ots.sin_port = htons(SERV_TCP_PORT);
}

```

```

/* Open a TCP socket (an Internet stream socket). */
if ((*sockfd_ots = socket(AF_INET,SOCK_STREAM,0)) < 0)
{
    printf("\n client: can't open stream socket\n");
    *socket_err_ots=1;
}
else
{
    /* Connect to the server */
    if (connect(*sockfd_ots, (struct sockaddr *) &serv_addr_ots,sizeof(serv_addr_ots)) < 0)
    {
        printf("\n client: can't connect to server\n");
        *socket_err_ots=1;
    }
}

}

/* File transfer procedure :
   This procedure uses the socket to send file from client
   to server and receive the response message from the
   server */

tcp_cli(ftp_file_name_tc,sockfd_tc,msg_size_tc,file_len_s_tc,ftp_file_tc,read_file_time_t
c)

char *ftp_file_name_tc;
int *sockfd_tc;
int msg_size_tc;
char *file_len_s_tc;
FILE *ftp_file_tc;
struct timeval *read_file_time_tc;

{
    int n_tc;
    char *sendline_tc,*recvline_tc;
    long file_len_tc;
    struct timeval start_rec_time_v_tc;
    struct timezone start_rec_time_z_tc;
    struct timeval end_rec_time_v_tc;

```

```

struct timezone end_rec_time_z_tc;

n_tc=0;
file_len_tc=0;
sendline_tc=(char *)malloc((msg_size_tc+1)*sizeof(char));
recvline_tc=(char *)malloc((msg_size_tc+1)*sizeof(char));
*sendline_tc='\0';
*recvline_tc='\0';
printf("\n !!! File transfer simulation is processing !!! \n");
n_tc=strlen(file_len_s_tc);
file_len_s_tc[n_tc]='\0';

/* send the file length to server */
if (write(*sockfd_tc, file_len_s_tc, n_tc) != n_tc)
{
    printf("tcp_cli: write file_len_s_t error on socket \n");
}
else
{
    if (read(*sockfd_tc, recvline_tc, n_tc) != n_tc)
    {
        printf("tcp_echo: read file_len_s_tc error on socket \n");
    }
    else
    {
        *recvline_tc='\0';
        n_tc = strlen(ftp_file_name_tc);

        /* write the file name to server */
        if (write(*sockfd_tc, ftp_file_name_tc, n_tc) != n_tc)
        {
            printf("tcp_cli: write file_name_tc error on socket \n");
        }
        else
        {
            if (read(*sockfd_tc, recvline_tc, n_tc) != n_tc)
            {
                printf("tcp_echo: read ftp_file_name_tc error on socket \n");
            }
            else
            {
                *recvline_tc='\0';
                file_len_tc=atol(file_len_s_tc);
            }
        }
    }
}

```

```

if (ftp_file_tc == NULL)
{
    printf("\n !!! Transferred file %s open error !!!\n",ftp_file_name_tc);
}
else
{
    while (file_len_tc > 0)
    {
        gettimeofday(&start_rec_time_v_tc,&start_rec_time_z_tc);
        n_tc=fread(sendline_tc, sizeof(char),msg_size_tc, ftp_file_tc);
        gettimeofday(&end_rec_time_v_tc,&end_rec_time_z_tc);
        tvsub(&end_rec_time_v_tc,&start_rec_time_v_tc);
        tvadd(read_file_time_tc,&end_rec_time_v_tc);
        file_len_tc=file_len_tc-n_tc;
        *(sendline_tc+n_tc)='\0';

        /* write the file content to server */
        if (write(*sockfd_tc, sendline_tc, n_tc) !=
            n_tc)
        {
            printf("tcp_cli: Transferred file write error ");
            printf("on socket\n");
        }

        /* Now read a message from the socket and write
           it to our standard output */
        n_tc=read(*sockfd_tc, recvline_tc, msg_size_tc);
        if (n_tc < 0)
        {
            printf("tcp_echo : Transferred file echo read error\n");
        }
        *(recvline_tc+n_tc)='\0'; /* null terminate */
    }
    if (ferror(ftp_file_tc))
    {
        printf("tcp_cli: error reading file");
    }
}
}
}
}
}
free(sendline_tc);

```

```

    free(recvline_tc);

}

/* This procedure will receive the file transfer time from server and
   save these data into file transfer information files */
tcp_summary(sockfd_ts,ftp_info_name_ts,ftp_info_ts,read_file_time_ts)

int *sockfd_ts;
char *ftp_info_name_ts;
FILE *ftp_info_ts;
struct timeval read_file_time_ts;

{
    char file_len_s_ts[34],time_hour_ts[34];
    char ftp_with_io_sec_ts[34],ftp_with_io_usec_ts[34];
    char ftp_without_io_sec_ts[34],ftp_without_io_usec_ts[34];
    int n_ts;
    long time_hour_d_ts;

    file_len_s_ts[0]='\0';
    time_hour_ts[0]='\0';
    ftp_with_io_sec_ts[0]='\0';
    ftp_with_io_usec_ts[0]='\0';
    ftp_without_io_sec_ts[0]='\0';
    ftp_without_io_usec_ts[0]='\0';
    printf("\n File transfer finished !!\n");

    /* get the hour of file transfer simulation */
    n_ts = read(*sockfd_ts, time_hour_ts, 34);
    if (n_ts < 0)
    {
        printf("tcp_smy : time_hour_ts read error\n");
    }
    else
    {
        time_hour_ts[n_ts]='\0';
        if (write(*sockfd_ts, time_hour_ts, n_ts) != n_ts)
        {
            printf("tcp_smy : time_hour_ts write error\n");
        }
    }
}

```

```

/* get the second for file transfer with disk I/O */
n_ts = read(*sockfd_ts, ftp_with_io_sec_ts, 34);
if (n_ts < 0)
{
    printf("tcp_smy : ftp_with_io_sec_ts read error\n");
}
else
{
    ftp_with_io_sec_ts[n_ts]='\0';
    if (write(*sockfd_ts, ftp_with_io_sec_ts, n_ts) != n_ts)
    {
        printf("tcp_smy : ftp_with_io_sec_ts write error\n");
    }
}

/* get the u_second for file transfer with disk I/O */
n_ts = read(*sockfd_ts, ftp_with_io_usec_ts, 34);
if (n_ts < 0)
{
    printf("tcp_smy : ftp_with_io_usec_ts read error\n");
}
else
{
    ftp_with_io_usec_ts[n_ts]='\0';
    if (write(*sockfd_ts, ftp_with_io_usec_ts, n_ts) != n_ts)
    {
        printf("tcp_smy : ftp_with_io_usec write error\n");
    }
}

/* get the second for file transfer without disk I/O */
n_ts = read(*sockfd_ts, ftp_without_io_sec_ts, 34);
if (n_ts < 0)
{
    printf("tcp_smy : ftp_without_io_sec_ts read error\n");
}
else
{
    ftp_without_io_sec_ts[n_ts]='\0';
    if (write(*sockfd_ts, ftp_without_io_sec_ts, n_ts) != n_ts)
    {
        printf("tcp_smy : ftp_without_io_sec_ts write error\n");
    }
}

```

```

    }
}

/* get the u_second for file transfer without disk I/O */
n_ts = read(*sockfd_ts, ftp_without_io_usec_ts, 34);
if (n_ts < 0)
{
    printf("tcp_smy : ftp_without_io_usec_ts read error\n");
}
else
{
    ftp_without_io_usec_ts[n_ts]='\0';
    if (write(*sockfd_ts, ftp_without_io_usec_ts, n_ts) != n_ts)
    {
        printf("tcp_smy : ftp_without_io_usec write error\n");
    }
}

/* get the ftp_file length */
n_ts = read(*sockfd_ts, file_len_s_ts, 34);
if (n_ts < 0)
{
    printf("tcp_smy : file_len_s_ts read error\n");
}
else
{
    file_len_s_ts[n_ts]='\0';
    if (write(*sockfd_ts, file_len_s_ts, n_ts) != n_ts)
    {
        printf("tcp_smy : file_len_s_ts write error\n");
    }
}

fprintf(ftp_info_ts, "\n%s %s %s %s %s %s",
        file_len_s_ts,
        time_hour_ts,
        ftp_with_io_sec_ts,
        ftp_with_io_usec_ts,
        ftp_without_io_sec_ts,
        ftp_without_io_usec_ts);
printf("\n\n*** SUMMARY ***\n\n");
printf("The transfered file length is : %s\n", file_len_s_ts);
time_hour_d_ts = atol(time_hour_ts);
printf("The file transfer time is : %s\n", ctime(&time_hour_d_ts));

```

```

printf("The sec for file transfer with disk I/O is : %s\n",ftp_with_io_sec_ts);
printf("The mirco sec for file transfer with disk I/O is :");
printf(" %s\n",ftp_with_io_usec_ts);
printf("The sec for file transfer without disk I/O is :");
printf(" %s\n",ftp_without_io_sec_ts);
printf("The mirco sec for file transfer without disk I/O is");
printf(" : %s\n",ftp_without_io_usec_ts);

}

/* This procedure will open a DATAGRAM socket for UDP protocol */
open_udp_socket(serv_host_addr_ous,sockfd_ous,serv_addr_ous.socket_err_ous)

char *serv_host_addr_ous;
int *sockfd_ous;
struct sockaddr_in *serv_addr_ous;
int *socket_err_ous;

{
    struct sockaddr_in cli_addr_ous;

    /* Fill in the structure "serv_addr" with the address of the server that we want to send to. */
    bzero((char *) serv_addr_ous, sizeof(*serv_addr_ous));
    serv_addr_ous->sin_family = AF_INET;
    serv_addr_ous->sin_addr.s_addr = inet_addr(serv_host_addr_ous);
    serv_addr_ous->sin_port = htons(SERV_UDP_PORT);

    /* Open a UDP socket (an Internet datagram socket). */
    if ((*sockfd_ous = socket(AF_INET,SOCK_DGRAM,0)) < 0)
    {
        printf("client : can't open datagram socket\n");
        *socket_err_ous=1;
    }
    else
    {

        /* Bind any local address for us */
        bzero((char *) &cli_addr_ous, sizeof(cli_addr_ous));
        cli_addr_ous.sin_family = AF_INET;
        cli_addr_ous.sin_addr.s_addr = htonl(INADDR_ANY);
        cli_addr_ous.sin_port = htons(0);
        if (bind(*sockfd_ous, (struct sockaddr *) &cli_addr_ous,sizeof(cli_addr_ous)) < 0)

```



```

        {
            printf("client : can't bind local address\n");
            *socket_err_ous=1;
        }
    }

}

udp_cli(ftp_file_name_uc,sockfd_uc,msg_size_uc,pserv_addr_uc,
        servlen_uc,file_len_s_uc,ftp_file_uc,read_file_time_uc)

char *ftp_file_name_uc;
int *sockfd_uc;
int msg_size_uc;
struct sockaddr *pserv_addr_uc;
        /* ptr to appropriate sockaddr_XX structure */
int servlen_uc; /* sizeof(*pcli_addr) */
char *file_len_s_uc;
FILE *ftp_file_uc;
struct timeval *read_file_time_uc;

{
    int n_uc;
    char *sendline_uc,*recvline_uc;
    long file_len_uc;
    struct timeval start_rec_time_v_uc;
    struct timezone start_rec_time_z_uc;
    struct timeval end_rec_time_v_uc;
    struct timezone end_rec_time_z_uc;

    n_uc=0;
    file_len_uc=0;
    sendline_uc=(char *)malloc((msg_size_uc+1)*sizeof(char));
    recvline_uc=(char *)malloc((msg_size_uc+1)*sizeof(char));
    *sendline_uc='\0';
    *recvline_uc='\0';
    printf("\n !!! File transfer simulation is processing !!!\n");
    n_uc=strlen(file_len_s_uc);

    /* send the file length to server */
    if (sendto(*sockfd_uc, file_len_s_uc, n_uc, 0,pserv_addr_uc, servlen_uc) != n_uc)
    {

```

```

    printf("udp_cli: sendto file_len_s error on socket \n");
}
else
{
    n_uc=recvfrom(*sockfd_uc, file_len_s_uc, n_uc, 0,pserv_addr_uc, &servlen_uc);
    if (n_uc < 0)
    {
        printf("udp_cli : recvfrom file_len_s error \n");
    }
    else
    {
        *(file_len_s_uc+n_uc) = '\0'; /* null terminate */
    }
    n_uc=strlen(ftp_file_name_uc);

    /* send the file name to server */
    if (sendto(*sockfd_uc, ftp_file_name_uc, n_uc, 0,pserv_addr_uc, servlen_uc) != n_uc)
    {
        printf("udp_cli: sendto ftp_file_name_u error on socket \n");
    }
    else
    {
        n_uc=recvfrom(*sockfd_uc, ftp_file_name_uc, n_uc, 0,pserv_addr_uc, &servlen_uc);
        if (n_uc < 0)
        {
            printf("udp_cli : recvfrom ftp_file_name_uc error \n");
        }
        else
        {
            *(ftp_file_name_uc+n_uc) = '\0'; /* null terminate */
        }
        file_len_uc=atoi(file_len_s_uc);
        if (ftp_file_uc == NULL)
        {
            printf("\n !!! Transferred file %s open error !!! \n",ftp_file_name_uc);
        }
        else
        {
            while (file_len_uc > 0)
            {
                gettimeofday(&start_rec_time_v_uc,&start_rec_time_z_uc);
                n_uc=fread(sendline_uc, sizeof(char),msg_size_uc, ftp_file_uc);
                gettimeofday(&end_rec_time_v_uc,&end_rec_time_z_uc);
            }
        }
    }
}

```

```

    tvsub(&end_rec_time_v_uc,&start_rec_time_v_uc);
    tvadd(read_file_time_uc,&end_rec_time_v_uc);
    file_len_uc=file_len_uc-n_uc;
    *(sendline_uc+n_uc)='\0';

    /* send the file content to server */
    if (sendto(*sockfd_uc, sendline_uc, n_uc, 0,pserv_addr_uc, servlen_uc) != n_uc)
    {
        printf("udp_cli: Transferred file sendto error on socket \n");
    }

    /* Now read a message from the socket and write it
       to our standard output */
    n_uc=recvfrom(*sockfd_uc, recvline_uc, msg_size_uc, 0,
                  pserv_addr_uc, &servlen_uc);
    if (n_uc < 0)
    {
        printf("udp_cli : Transferred file echo recvfrom error \n");
    }
    else
    {
        *(recvline_uc+n_uc) = '\0'; /* null terminate */
    }
}
if (ferror(ftp_file_uc))
{
    printf("udp_cli: error reading file");
}
}
}
free(sendline_uc);
free(recvline_uc);

}

/* This procedure will receive the FTP time from server and save
   these data into FTP information files */
udp_summary(sockfd_us,ftp_info_name_us,pserv_addr_us, servlen_us,
            ftp_info_us,read_file_time_us)

int *sockfd_us;

```

```

char *ftp_info_name_us;
struct sockaddr *pserv_addr_us;
        /* ptr to appropriate sockaddr_XX structure */
int servlen_us; /* sizeof(*pcli_addr) */
FILE *ftp_info_us;
struct timeval read_file_time_us;

{
char time_hour_us[34],file_len_s_us[34];
char ftp_with_io_sec_us[34],ftp_with_io_usec_us[34];
char ftp_without_io_sec_us[34],ftp_without_io_usec_us[34];
int n_us;
long time_hour_d_us;

file_len_s_us[0]='\0';
time_hour_us[0]='\0';
ftp_with_io_sec_us[0]='\0';
ftp_with_io_usec_us[0]='\0';
ftp_without_io_sec_us[0]='\0';
ftp_without_io_usec_us[0]='\0';
printf("File transfer finished !!\n");

/* get the hour in a day for file transfer */
n_us = recvfrom(*sockfd_us, time_hour_us, 34, 0,pserv_addr_us, &servlen_us);
if (n_us < 0)
{
printf("udp_smy : time_hour_us recvfrom error\n");
}
else
{
time_hour_us[n_us]='\0';

/* send the time_hour_us to server */
if (sendto(*sockfd_us, time_hour_us, n_us, 0,pserv_addr_us, servlen_us) != n_us)
{
printf("udp_smy : sendto time_hour error on socket\n");
}
}

/* get the second for file transfer with disk I/O */
n_us = recvfrom(*sockfd_us, ftp_with_io_sec_us, 34, 0,pserv_addr_us, &servlen_us);
if (n_us < 0)
{

```

```

    printf("udp_smy : ftp_with_io_sec_us recvfrom error\n");
}
else
{
    ftp_with_io_sec_us[n_us]='\0';

    /* send the ftp_with_io_sec_us to server */
    if (sendto(*sockfd_us, ftp_with_io_sec_us, n_us, 0,
               pserv_addr_us, servlen_us) != n_us)
    {
        printf("udp_smy : sendto ftp_with_io_sec_us error on socket\n");
    }
}

/* get the mirco second for file transfer with disk I/O */
n_us=recvfrom(*sockfd_us, ftp_with_io_usec_us, 34, 0, pserv_addr_us, &servlen_us);
if (n_us < 0)
{
    printf("udp_smy : ftp_with_io_usec_us recvfrom error\n");
}
else
{
    ftp_with_io_usec_us[n_us]='\0';

    /* send the ftp_with_io_usec_us to server */
    if (sendto(*sockfd_us, ftp_with_io_usec_us, n_us, 0,
               pserv_addr_us, servlen_us) != n_us)
    {
        printf("udp_smy : sendto ftp_with_io_usec_us error on socket\n");
    }
}

/* get the second for file transfer without disk I/O */
n_us = recvfrom(*sockfd_us, ftp_without_io_sec_us, 34, 0, pserv_addr_us, &servlen_us);
if (n_us < 0)
{
    printf("udp_smy : ftp_without_io_sec_us recvfrom error\n");
}
else
{
    ftp_without_io_sec_us[n_us]='\0';

    /* send the ftp_without_io_sec_us to server */

```

```

        if (sendto(*sockfd_us, ftp_without_io_sec_us, n_us, 0,
                    pserv_addr_us, servlen_us) != n_us)
        {
            printf("udp_smy : sendto ftp_without_io_sec_us error on socket \n");
        }
    }

    /* get the mirco second for file transfer without disk I/O */
    n_us=recvfrom(*sockfd_us, ftp_without_io_usec_us, 34, 0, pserv_addr_us, &servlen_us);
    if (n_us < 0)
    {
        printf("udp_smy : ftp_without_io_usec_us recvfrom error \n");
    }
    else
    {
        ftp_without_io_usec_us[n_us]='\0';

        /* send the ftp_without_io_sec_us to server */
        if (sendto(*sockfd_us, ftp_without_io_usec_us, n_us, 0,
                    pserv_addr_us, servlen_us) != n_us)
        {
            printf("udp_smy : sendto ftp_without_io_usec_us error on socket \n");
        }
    }

    /* get the file length for file transfer */
    n_us=recvfrom(*sockfd_us, file_len_s_us, 34, 0, pserv_addr_us, &servlen_us);
    if (n_us < 0)
    {
        printf("udp_smy : file_len_s_us recvfrom error \n");
    }
    else
    {
        file_len_s_us[n_us]='\0';

        /* send the file_len_s_us to server */
        if (sendto(*sockfd_us, file_len_s_us, n_us, 0, pserv_addr_us, servlen_us) != n_us)
        {
            printf("udp_smy : sendto file_len_s_us error on socket \n");
        }
    }
    fprintf(ftp_info_us, "\n%s %s %s %s %s %s",
            file_len_s_us,

```

```

        time_hour_us,
        ftp_with_io_sec_us,
        ftp_with_io_usec_us,
        ftp_without_io_sec_us,
        ftp_without_io_usec_us);
printf("\n\n*** SUMMARY ***\n\n");
printf("The transfered file length is : %s\n",file_len_s_us);
time_hour_d_us=atol(time_hour_us);
printf("The file transfer time is : %s\n",ctime(&time_hour_d_us));
printf("The sec for file transfer with disk I/O is : %s\n",ftp_with_io_sec_us);
printf("The micro sec for file transfer with disk I/O is : %s\n",ftp_with_io_usec_us);
printf("The sec for file transfer without disk I/O is : %s\n",ftp_without_io_sec_us);
printf("The micro sec for file transfer without disk I/O is : %s\n",ftp_without_io_usec_us);
}

```

```

/* This procedure will generate the statistic report from
   the file transfer simulation measurement data      */
statistic_report()
{
    FILE *ftp_data_sr;
    char cur_path_sr[256],title_sr[81];
    char ftp_data_filename_sr[45];
    char serv_host_addr_sr[256];
    int simple_num_sr,data_without_io_sr,cur_path_len_sr;
    long time_hour_interval_range_sr;
    int chdir_flag_sr;

    simple_num_sr=0;
    get_ftp_file_path(cur_path_sr,serv_host_addr_sr,&chdir_flag_sr);
    if (chdir_flag_sr != -1)
    {
        open_data_file(&ftp_data_sr,ftp_data_filename_sr);
        if (strlen(ftp_data_filename_sr) > 0)
        {
            set_simple_num_and_time_hour_interval_range
            (&simple_num_sr,&time_hour_interval_range_sr);
            if (ftp_data_sr == (FILE *)NULL)
            {
                chdir(cur_path_sr);
                printf("!!! Can not open the %s file transfer simulation data file !!!\n");
                printf("!!! Please check the %s file transfer simulation data file !!!\n");
            }
        }
    }
}

```

```

else
{
    check_file_format(ftp_data_sr,&data_without_io_sr,title_sr);
    while (!feof(ftp_data_sr))
    {
        stat_data(ftp_data_sr,title_sr,serv_host_addr_sr,
                  data_without_io_sr,simple_num_sr,
                  time_hour_interval_range_sr);
    }
    fclose(ftp_data_sr);
}
else
{
    printf("!!! Exit statistic data report function !!!\n\n");
}
chdir(cur_path_sr);
}
else
{
    printf("!!! Can not get the current path !!!\n");
    printf("!!! Please check the file path !!!\n\n");
}
printf("!!! Please hit the <Enter> key to continue !!!\n");
getchar();
}

struct intervals
{
    long      interval;
    float     counter;
    struct intervals *next;
};

typedef struct intervals *intervals_ptr;

/* Statistics for the file transfer collection data */
stat_data(ftp_data_sd,title_sd,serv_host_addr_sd,data_without_io_sd,
          simple_num_sd,time_hour_interval_range_sd)

FILE *ftp_data_sd;

```



```

char *title_sd;
char *serv_host_addr_sd;
int data_without_io_sd;
int simple_num_sd;
long time_hour_interval_range_sd;

{
float sum1_with_io_sd, sum2_with_io_sd, sec_with_io_sd;
float usec_with_io_sd, mean_with_io_sd, var_with_io_sd;
float sum1_without_io_sd, sum2_without_io_sd, sec_without_io_sd;
float usec_without_io_sd, mean_without_io_sd, var_without_io_sd;
long time_hour_sd, max_time_hour_sd, min_time_hour_sd;
long init_time_hour_sd, last_time_hour_sd;
long time_hour_interval_sd;
long time_hour_interval_begin_sd, time_hour_interval_end_sd;
long file_length_sd, sum_file_length_sd, avg_file_length_sd;
float max_sec_with_io_sd, max_usec_with_io_sd;
float min_sec_with_io_sd, min_usec_with_io_sd;
float max_sec_without_io_sd, max_usec_without_io_sd,
      min_sec_without_io_sd, min_usec_without_io_sd;
float rec_num_sd;
int simple_index_sd;
double s_diff_with_io_sd, s_diff_without_io_sd;
intervals_ptr i_head_with_io_sd, i_head_without_io_sd,
              i_ptr_with_io_sd, i_ptr_without_io_sd,
              i_head_time_hour_sd, i_ptr_time_hour_sd;

/*initialize the variables */
simple_index_sd=1;
sec_without_io_sd=0.0;
usec_without_io_sd=0.0;
sum1_with_io_sd=0.0;
sum2_with_io_sd=0.0;
max_sec_with_io_sd=0.0;
max_usec_with_io_sd=0.0;
min_sec_with_io_sd=999999.0;
min_usec_with_io_sd=999999.0;
sum1_without_io_sd=0.0;
sum2_without_io_sd=0.0;
max_sec_without_io_sd=0.0;
max_usec_without_io_sd=0.0;
min_sec_without_io_sd=999999.0;
min_usec_without_io_sd=999999.0;

```

```

rec_num_sd=0.0;
max_time_hour_sd=0;
min_time_hour_sd=999999999;
sum_file_length_sd=0;
avg_file_length_sd=0;
i_head_with_io_sd=NULL;
i_head_without_io_sd=NULL;
i_head_time_hour_sd=NULL;
i_ptr_with_io_sd=NULL;
i_ptr_without_io_sd=NULL;
i_ptr_time_hour_sd=NULL;

while (!feof(ftp_data_sd) && (simple_index_sd <= simple_num_sd))
{
    if (data_without_io_sd == 0)
    {
        fscanf(ftp_data_sd, "\n%ld %ld %f %f",
            &file_length_sd,
            &time_hour_sd,
            &sec_with_io_sd,
            &usec_with_io_sd);
    }
    else
    {
        fscanf(ftp_data_sd, "\n%ld %ld %f %f %f %f",
            &file_length_sd, &time_hour_sd,
            &sec_with_io_sd,
            &usec_with_io_sd,
            &sec_without_io_sd,
            &usec_without_io_sd);
    }

    /* To compute time intervals information with WITH_DISK_IO data */
    get_max_and_min_interval_time(sec_with_io_sd, usec_with_io_sd,
        &sum1_with_io_sd, &sum2_with_io_sd,
        &max_sec_with_io_sd, &max_usec_with_io_sd,
        &min_sec_with_io_sd, &min_usec_with_io_sd);
    stat_time_interval((long) sec_with_io_sd, &i_head_with_io_sd);
    if (data_without_io_sd == 1)
    {
        /* To compute intervals time information with WITHOUT_DISK_IO data */
        get_max_and_min_interval_time(sec_without_io_sd, usec_without_io_sd,

```

```

                                &sum1_without_io_sd,
                                &sum2_without_io_sd,
                                &max_sec_without_io_sd,
                                &max_usec_without_io_sd,
                                &min_sec_without_io_sd,
                                &min_usec_without_io_sd);
    stat_time_interval((long) sec_without_io_sd.&i_head_without_io_sd);
}
sum_file_length_sd=sum_file_length_sd+file_length_sd;

/* To get the max and min time_hour for
   the file transfer simulation time range */
get_max_and_min_hour_time(time_hour_sd.&min_time_hour_sd,
                           &max_time_hour_sd);

/* To get the initial hour time interval */
if (simple_index_sd == 1)
{
    init_time_hour_sd=time_hour_sd;
}

/* To get the last hour time interval */
if (simple_index_sd == simple_num_sd)
{
    last_time_hour_sd=time_hour_sd;
}

/* To compute hour time interval for file transfer simulation */
time_hour_interval_sd=(time_hour_sd-init_time_hour_sd)/
    time_hour_interval_range_sd;
stat_time_interval(time_hour_interval_sd,&i_head_time_hour_sd);

/* To compute the number of file transfer simulation records */
rec_num_sd=rec_num_sd+1.0;
simple_index_sd=simple_index_sd+1;
} /* while (!feof(ftp_data_sd)) */

mean_with_io_sd=sum1_with_io_sd/rec_num_sd;
var_with_io_sd=((rec_num_sd*sum2_with_io_sd)-
    (sum1_with_io_sd*sum1_with_io_sd))/
    (rec_num_sd*(rec_num_sd-1));
avg_file_length_sd=sum_file_length_sd/((long) rec_num_sd);
s_diff_with_io_sd=sqrt((double) var_with_io_sd);

```

```

/* output the statistics result of
file transfer simulation measurement */
printf("\n\n ***** File Transfer Simulation statistic report ***** \n\n\n");
printf("The address of server      : %s\n",serv_host_addr_sd);
printf("The file name is           : %s\n",title_sd);
printf("The file length is          : %ld bytes\n",avg_file_length_sd);
printf("The sample number of record is : %3.0f\n",rec_num_sd);
printf("The time hour interval range is : per %d ",(time_hour_interval_range_sd/60));
printf("minutes \n\n");
printf("*** File transfer simulation time range ***\n\n");
printf("File transfer start time : %s \n",ctime(&min_time_hour_sd));
printf("File transfer end time   : %s \n\n",ctime(&max_time_hour_sd));

/* output the statistics result of time_hour interval */
i_ptr_time_hour_sd=i_head_time_hour_sd;
while (i_ptr_time_hour_sd != NULL)
{
    time_hour_interval_begin_sd=init_time_hour_sd+ i_ptr_time_hour_sd->interval*
                                time_hour_interval_range_sd;
    if (i_ptr_time_hour_sd->next != NULL)
    {
        time_hour_interval_end_sd=init_time_hour_sd+
                                (i_ptr_time_hour_sd->interval+((long) 1))*
                                time_hour_interval_range_sd;
    }
    else
    {
        time_hour_interval_end_sd=last_time_hour_sd;
    }
    printf("The file transfer time hour interval begin      : %s",
           ctime(&time_hour_interval_begin_sd));
    printf("The file transfer time hour interval end          : %s",
           ctime(&time_hour_interval_end_sd));
    printf("The percent in this time hour interval is : %2.2f ",
           (i_ptr_time_hour_sd->counter/rec_num_sd)* 100.0);
    printf("\n\n");
    i_ptr_time_hour_sd=i_ptr_time_hour_sd->next;
}
free(i_head_time_hour_sd);
printf("*** File transfer with disk I/O ***\n\n");
printf("The mean value is           : %2.6f seconds\n",mean_with_io_sd);
printf("The variance is             : %f seconds\n",var_with_io_sd);

```

```

printf("The standard deviation is : %f seconds\n",s_diff_with_io_sd);
printf("The max FTP time is : %4.0f sec, %6.0f usec\n",
      max_sec_with_io_sd,max_usec_with_io_sd);
printf("The min FTP time is : %4.0f sec, %6.0f usec\n\n",
      min_sec_with_io_sd,min_usec_with_io_sd);

/* output the statistics result of time interval with io */
i_ptr_with_io_sd=i_head_with_io_sd;
while (i_ptr_with_io_sd != NULL)
{
    printf("The file transfer time range from %3.0ld seconds to %3.0ld ",
          i_ptr_with_io_sd->interval,i_ptr_with_io_sd->interval+((long) 1));
    printf("seconds is : %2.2f ",
          (i_ptr_with_io_sd->counter/rec_num_sd)* 100.0);
    printf("percent\n");
    i_ptr_with_io_sd=i_ptr_with_io_sd->next;
}
free(i_head_with_io_sd);

/* output the statistics result if the file transfer data file contains */
if (data_without_io_sd == 1)
{
    mean_without_io_sd=sum1_without_io_sd/rec_num_sd;
    var_without_io_sd=((rec_num_sd*sum2_without_io_sd)-
                      (sum1_without_io_sd*sum1_without_io_sd))/
                      rec_num_sd*(rec_num_sd-1));
    s_diff_without_io_sd=sqrt((double) var_without_io_sd);
    printf("\n*** File transfer without disk I/O ***\n\n");
    printf("The mean value is : %2.6f ",mean_without_io_sd);
    printf("seconds\n");
    printf("The variance is : %f ",var_without_io_sd);
    printf("seconds\n");
    printf("The standard deviation is : %f ",s_diff_without_io_sd);
    printf("seconds\n");
    printf("The max file transfer time is : %4.0f sec, %6.0f usec\n",
          max_sec_without_io_sd,max_usec_without_io_sd);
    printf("The min file transfer time is : %4.0f sec, %6.0f usec\n\n",
          min_sec_without_io_sd,min_usec_without_io_sd);

    /* output the statistics result of time interval without io */
    i_ptr_without_io_sd=i_head_without_io_sd;
    while (i_ptr_without_io_sd != NULL)
    {

```

```

        printf("The file transfer time range from %.3ld seconds to %.3ld ",
            i_ptr_without_io_sd->interval,
            i_ptr_without_io_sd->interval+((long) 1));
        printf("seconds is : %.2f ",(i_ptr_without_io_sd->counter/rec_num_sd)* 100.0);
        printf(" percent \n");
        i_ptr_without_io_sd=i_ptr_without_io_sd->next;
    }
    free(i_head_without_io_sd);
}
printf("\n\n *** End of file transfer simulation data statistics function *** \n\n");
}

```

```

set_simple_num_and_time_hour_interval_range(simple_num_set,
                                             time_hour_interval_range_set)

```

```

int *simple_num_set;
long *time_hour_interval_range_set;

{
    int simple_num_temp_set;
    long time_hour_interval_range_temp_set;

    simple_num_temp_set=0;
    while (simple_num_temp_set < 10)
    {

        /* Input how many file transfer samples which we want to compute */
        printf("\n*** How many file transfer samples(10~) will be used in one ");
        printf("statistics ==> \n");
        scanf("%d",&simple_num_temp_set);
        if (simple_num_temp_set < 10)
        {
            printf("!!! file transfer simple number error !!!\n");
            printf("!!! It must be greater than 10 !!!\n");
        }
    }
    *simple_num_set=simple_num_temp_set;
    time_hour_interval_range_temp_set=0;
    while (time_hour_interval_range_temp_set < 10)
    {

```

```

/* Input the time hour interval range which we want to compute */
printf("\n*** How many minutes(10~) do you want to be a ");
printf("time hour interval range ==> \n");
scanf("%ld",&time_hour_interval_range_temp_set);
if (time_hour_interval_range_temp_set < 10)
{
    printf("!!! File transfer time hour interval ranage error !!!\n");
    printf("!!! It must be greater than 10 !!!\n");
}
}
*time_hour_interval_range_set=time_hour_interval_range_temp_set*((long) 60);
}

```

```

get_ftp_file_path(cur_path_gcp,serv_host_addr_gcp,chmod_flag_gcp)

```

```

char *cur_path_gcp;
char *serv_host_addr_gcp;
int *chmod_flag_gcp;

```

```

{
    int try_again_gcp,cur_path_len_gcp,inaddr_gcp;
    struct hostent *dummy_ptr_gcp;
        /* this is a dummy variable for the
        convert_host_name_to_address
        procedure in this procedure. */

```

```

/* To keep the current file path */
*chmod_flag_gcp=0;
try_again_gcp=1;
if (getcwd(cur_path_gcp,255) == NULL)
{
    printf("\n !!! Can not get the current directory !!!\n");
}
cur_path_len_gcp=strlen(cur_path_gcp);
cur_path_gcp[cur_path_len_gcp]='\0';
printf("\n\n *** Please input the server address (name, alias) ==>> \n);
scanf("%s",serv_host_addr_gcp);
if ((inaddr_gcp = inet_addr(serv_host_addr_gcp)) == INADDR_NONE)
{
    convert_host_name_to_address(serv_host_addr_gcp,&dummy_ptr_gcp);
}

```

```

    }
    else
    {
        convert_host_address_information(serv_host_addr_gcp,&dummy_ptr_gcp);
    }
    if (strlen(serv_host_addr_gcp) > 0)
    {
        *chdir_flag_gcp=chdir(serv_host_addr_gcp);
    }
    else
    {
        *chdir_flag_gcp=-1;
    }
    while ((*chdir_flag_gcp == -1) && try_again_gcp < 4)
    {
        printf("\n !!! Can not change subdirectory to %s !!!\n",serv_host_addr_gcp);
        printf("\n\n *** Please input the server address again ==>> \n");
        scanf("%s",serv_host_addr_gcp);
        if ((inet_addr(serv_host_addr_gcp)) == INADDR_NONE)
        {
            convert_host_name_to_address(serv_host_addr_gcp,&dummy_ptr_gcp);
        }
        else
        {
            convert_host_address_information(serv_host_addr_gcp,
                                            &dummy_ptr_gcp);
        }
        if (strlen(serv_host_addr_gcp) > 0)
        {
            *chdir_flag_gcp=chdir(serv_host_addr_gcp);
            serv_host_addr_gcp[strlen(serv_host_addr_gcp)]='\0';
        }
        else
        {
            *chdir_flag_gcp=-1;
            serv_host_addr_gcp[0]='\0';
        }
        try_again_gcp=try_again_gcp+1;
    }
}

```



```

typedef FILE *file_ptr;

open_data_file(ftp_data_odf,ftp_data_filename_odf)

file_ptr *ftp_data_odf;
char *ftp_data_filename_odf;

{
    int try_again_odf;
    char dummy_file_size_odf[7];

    dummy_file_size_odf[0]='\0';
    try_again_odf=1;
    printf("!!! Please wait a second for the FTP data file listing !!!");
    system("ls -l *.* > file.tmp");
    file_selection(ftp_data_filename_odf,dummy_file_size_odf,2);
    if (strlen(ftp_data_filename_odf) > 0)
    {

        /* open the file transfer data file */
        if ((*ftp_data_odf=fopen(ftp_data_filename_odf,"r")) == NULL)
        {
            fclose(*ftp_data_odf);
            while ((*ftp_data_odf=fopen(ftp_data_filename_odf,"r")) == NULL &&
                try_again_odf < 3)
            {
                fclose(*ftp_data_odf);
                try_again_odf=try_again_odf+1;
                printf("Can not open the file %s !!!\n",ftp_data_filename_odf);
                printf("*** Please input the data file name again ==>\n");
                scanf("%s",ftp_data_filename_odf);
                getchar();
            }
        }
    }
}

check_file_format(ftp_data_cff,data_without_io_cff,title_cff)

FILE *ftp_data_cff;

```

```

int *data_without_io_cff;
char *title_cff;

{
    char record_buf[81];
    int record_buf_length,record_buf_index,space_ctr;

    /* To check the file format
       which include the without disk io data or not */
    fgets(record_buf,81,ftp_data_cff);
    fgets(record_buf,81,ftp_data_cff);
    record_buf_length=strlen(record_buf);
    record_buf_index=0;
    space_ctr=0;
    for (0;record_buf_index <= record_buf_length;record_buf_index++)
    {
        if (record_buf[record_buf_index] == ' ')
        {
            space_ctr=space_ctr+1;
        }
    }
    if (space_ctr > 4)
    {
        *data_without_io_cff=1;
    }
    else
    {
        *data_without_io_cff=0;
    }
    fseek(ftp_data_cff,0L,0);
    fscanf(ftp_data_cff,"%s\n",title_cff);
}

get_max_and_min_interval_time(sec,usec,sum1,sum2,max_sec,max_usec,
                               min_sec,min_usec)

float sec,usec;
float *sum1,*sum2,*max_sec,*max_usec,*min_sec,*min_usec;

{

```

```

/* To compute the sum of second, u_second and file length */
*sum1=*sum1+sec+(usec/1000000.0);
*sum2=*sum2+(sec+(usec/1000000.0))*(sec+(usec/1000000.0));

/* To get the max second and u_second for file transfer simulation */
if (sec > *max_sec)
{
    *max_sec=sec;
    *max_usec=usec;
}
else
{
    if (sec == *max_sec && usec > *max_usec)
    {
        *max_usec=usec;
    }
}

/* To get the min second and u_second for file transfer simulation */
if (sec < *min_sec)
{
    *min_sec=sec;
    *min_usec=usec;
}
else
{
    if (sec == *min_sec && usec < *min_usec)
    {
        *min_usec=usec;
    }
}
}

get_max_and_min_hour_time(time_hour_g,min_time_hour_g,max_time_hour_g)

long time_hour_g;
long *min_time_hour_g;
long *max_time_hour_g;

{

```

```

/* To get the max hour_time for file transfer simulation */
if (time_hour_g > *max_time_hour_g)
{
    *max_time_hour_g=time_hour_g;
}

/* To get the min hour_time for FTP */
if (time_hour_g < *min_time_hour_g)
{
    *min_time_hour_g=time_hour_g;
}

}

stat_time_interval(interval,i_head)

long interval;
intervals_ptr *i_head;

{

int new_node,end_queue;
intervals_ptr i_ptr,i_new,i_p_ptr,i_c_ptr,i_p;

/* check the interval of file transfer data */
i_ptr=*i_head;
new_node=0;
if (*i_head == NULL)
{
    i_new=(struct intervals *)malloc(sizeof(struct intervals));
    i_new->interval=interval;
    i_new->counter=1.0;
    i_new->next=NULL;
    *i_head=i_new;
    new_node=1;
}
else
{
    while (i_ptr->interval != interval && new_node == 0)
    {
        if (i_ptr->next == NULL)

```

```

{
i_new=(struct intervals *)malloc(sizeof(struct intervals));
i_new->interval=interval;
i_new->counter=1.0;
i_new->next=NULL;

/* insert new interval according to ascending */
if (i_new->interval < (*i_head)->interval)
{
i_new->next=*i_head;
*i_head=i_new;
}
else
{
if ((*i_head)->next == NULL)
{
(*i_head)->next=i_new;
}
else
{
i_c_ptr=(*i_head)->next;
i_p_ptr=*i_head;
end_queue=0;
while (i_c_ptr->interval < i_new->interval && end_queue != 1)
{
i_p_ptr=i_c_ptr;
if (i_c_ptr->next != NULL)
{
i_c_ptr=i_c_ptr->next;
}
else
{
end_queue=1;
}
}
if (end_queue != 1)
{
i_new->next=i_c_ptr;
i_p_ptr->next=i_new;
}
else
{
i_p_ptr->next=i_new;
}
}
}

```

```

        }
    }
    }
    new_node=1;
    }
    else
    {
        i_ptr=i_ptr->next;
    }
}
if (new_node == 0)
{
    i_ptr->counter=i_ptr->counter+1.0;
}
} /* end of interval check */

}

```

```

/* Subtract 2 timeval structs : out=out-in.
   Out is assumed to be >= in.      */
tvsub(out_ts,in_ts)

```

```

register struct timeval *out_ts,*in_ts;

{
    if ((out_ts->tv_usec -= in_ts->tv_usec) < 0)
    {
        out_ts->tv_sec--;
        out_ts->tv_usec += 1000000;
    }
    out_ts->tv_sec -= in_ts->tv_sec;
}

```

```

/* Add 2 timeval structs: out=out+in. */
tvadd(out_ta,in_ta)

```

```

register struct timeval *out_ta,*in_ta;

{
    if ((out_ta->tv_usec += in_ta->tv_usec) >= 1000000)

```

```

    {
        out_ta->tv_sec++;
        out_ta->tv_usec -= 1000000;
    }
    out_ta->tv_sec += in_ta->tv_sec;
}

```

/* This procedure will provide some utilities for users to manage
the file transfer data files and file transfer information file */
file_utility()

```

{
    int opt_fun_fu;
    int continue_flag_fu, exit_flag_fu;

    exit_flag_fu=0;
    while (opt_fun_fu < 1 || opt_fun_fu > 4 || exit_flag_fu == 0)
    {
        continue_flag_fu=1;
        opt_fun_fu=display_file_utility_menu();
        switch (opt_fun_fu)
        {
            case 1 :
                while (continue_flag_fu == 1)
                {
                    delete_files_and_subdirectory();
                    printf("====>> Do you want to delete another file ");
                    printf("transfer simulation server's data (Y/N) ?\n");
                    continue_flag_fu=confirm_continue();
                }
                break;

            case 2 :
                while (continue_flag_fu == 1)
                {
                    maintain_ftp_file();
                    printf("====>> Do you want to add/delete another transferred file (Y/N) ?\n");
                    continue_flag_fu=confirm_continue();
                }
                break;

            case 3 :

```

```

        printf("\n Do you really want to exit ? (Y/N) ");
        exit_flag_fu=confirm_continue();
        break;

    default :
        printf("\n Function selection is error, please check it !!! \n");
        break;
    }
}

}

/* To display the menu of file utility */
int display_file_utility_menu()
{
    int opt_fun_dfum;

    printf("\n\n      <<<<<  File Utility Menu  >>>>> \n\n");
    printf("          1. Delete one server's file transfer ");
    printf("simulation data \n");
    printf("          (Delete files & subdirectory) \n\n");
    printf("          2. Maintain FTP files \n\n");
    printf("          3. Exit \n\n\n");
    printf("          Please select one function (1-3) ==>>> ");
    scanf("%d",&opt_fun_dfum);
    getchar();
    printf("\n");
    return opt_fun_dfum;

}

/* To delete all files in the selected subdirectory
   which is named by the server's address */
delete_files_and_subdirectory()
{
    int try_again_dfs,cur_path_len_dfs;
    unsigned long inaddr_dfs;
    char serv_host_addr_dfs[255],cur_path_dfs[255];
    char rm_subdirectory_cmd_dfs[255];
    int confirm_flag_dfs;
    struct hostent *dummy_ptr_dfs;

```



```

        /* this is a dummy variable for the
        convert_host_address_information procedure
        in this procedure. */

try_again_dfs=0;
confirm_flag_dfs=0;
serv_host_addr_dfs[0]='\0';
printf(" *** Do you want to list server's address ==> (Y/N) ");
confirm_flag_dfs=confirm_continue();
if (confirm_flag_dfs == 1)
{
    system("ls -la *.*.*.* | more");
    printf("\n\n !!! Please hit any key to continue !!!\n");
}
while (strlen(serv_host_addr_dfs) == 0 && try_again_dfs <=3)
{
    printf("\n *** Please input the internet address of server ==> \n");
    scanf("%s",serv_host_addr_dfs);
    getchar();
    printf("\n");
    if ((inaddr_dfs = inet_addr(serv_host_addr_dfs)) == INADDR_NONE)
    {
        convert_host_name_to_address(serv_host_addr_dfs);
    }
    else
    {
        convert_host_address_information(serv_host_addr_dfs,&dummy_ptr_dfs);
    }
    if (strlen(serv_host_addr_dfs) > 0)
    {
        printf("!!! Do you really want to delete %s server's");
        printf(" file transfer simulation data !!!\n",serv_host_addr_dfs);
        printf("==>> (Y/N) ");
        confirm_flag_dfs=confirm_continue();
        if (confirm_flag_dfs == 1)
        {
            if (getcwd(cur_path_dfs,255) == NULL)
            {
                printf("\n !!! Can not get the current directory !!!\n");
            }
            else
            {
                chdir(serv_host_addr_dfs);
            }
        }
    }
}

```

```

        system("rm *.*MB");
        cur_path_len_dfs=strlen(cur_path_dfs);
        cur_path_dfs[cur_path_len_dfs]='\0';
        chdir(cur_path_dfs);
        strcpy(rm_subdirectory_cmd_dfs,"rmdir ");
        strcat(rm_subdirectory_cmd_dfs,serv_host_addr_dfs);
        system(rm_subdirectory_cmd_dfs);
        printf("!!! %s server's file transfer simulation data has ");
        printf("been deleted !!!\n\n",serv_host_addr_dfs);
    }
}
else
{
    printf("!!! %s server's file transfer simulation data is not ",serv_host_addr_dfs);
    printf("deleted !!!\n\n");
}
}
else
{
    printf("!!! Server host name(address) input error !!!\n");
    printf("!!! Please check the name or address !!!\n");
}
try_again_dfs=try_again_dfs+1;
}
}

```

```

/* maintain(add or delete) the transferred files */
maintain_ftp_file()
{
    FILE *ftp_file_mff;
    char cur_path_mff[256],ftp_file_path_mff[256];
    int opt_fun_mff;

    opt_fun_mff=0;
    cur_path_mff[0]='\0';
    ftp_file_path_mff[0]='\0';
    while (opt_fun_mff < 1 || opt_fun_mff > 3)
    {
        printf("\n\n\n <<<<< File Transfer - ");
        printf("File Maintenance Utility >>>>> \n\n");
        printf("          1. Add a transferred file \n\n");
    }
}

```

```

printf("          2. Delete a transferred file \n\n");
printf("          3. Exit \n\n\n");
printf("          Please select one function (1-3) ==>> ");
scanf("%d",&opt_fun_mff);
getchar();
printf("\n");
if (opt_fun_mff < 1 || opt_fun_mff > 3)
{
    printf("!!! Transferred File Maintain function selection is error !!!\n");
}
}
if (opt_fun_mff != 3)
{
    if (getcwd(cur_path_mff,255) == NULL)
    {
        printf("\n !!! Can not get the current directory !!!\n");
    }
    else
    {
        strcat(ftp_f_path_mff,cur_path_mff);
        strcat(ftp_file_path_mff,"/ftp_file");
        if (chdir(ftp_file_path_mff) == -1)
        {
            printf("\n !!! Can not change the %s subdirectory !!!\n",ftp_file_path_mff);
            printf("!!! Please check the %s subdirectory !!!\n");
        }
        else
        {
            switch(opt_fun_mff)
            {
                case 1 :
                    add_ftp_file();
                    break;

                case 2 :
                    delete_ftp_file();
                    break;

                default :
                    break;
            }
        }
        chdir(cur_path_mff);
    }
}

```

```

    }
}
else
{
    printf("\n!!! Exit  File Transfer - File Maintenance Utility !!!\n\n");
}
}

```

/* Add the FTP files to the ~/ftp_file subdirectory */

add_ftp_file()

```

{
    FILE *ftp_file_aff;
    char ftp_file_name_aff[17],file_size_s_aff[34];
    char char_aff;
    int file_size_unit_aff,file_size_aff;
    int loop1_aff,loop2_aff,confirm_flag_aff;

    file_size_unit_aff=0;
    file_size_aff=0;
    ftp_file_name_aff[0]='\0';
    char_aff='a';
    get_file_parameter(&file_size_unit_aff,&file_size_aff,ftp_file_name_aff);
    printf("\n*** Do you really want to add this transferred ");
    printf("%s file *** \n",ftp_file_name_aff);
    printf("====>> (Y/N) ");
    confirm_flag_aff=confirm_continue();
    if (confirm_flag_aff == 1)
    {
        printf("!!! %s file add is proceeding !!!\n",ftp_file_name_aff);
        ftp_file_aff=fopen(ftp_file_name_aff,"w");
        while (file_size_aff > 0)
        {
            if (file_size_unit_aff == 1)
            {
                for (loop1_aff=0;loop1_aff<1024;loop1_aff++)
                {
                    fprintf(ftp_file_aff,"%c",char_aff);
                }
            }
            else
            {

```

```

        for (loop1_aff=0;loop1_aff<1024;loop1_aff++)
        {
            for (loop2_aff=0;loop2_aff<1024;loop2_aff++)
            {
                fprintf(ftp_file_aff,"%c",char_aff);
            }
        }
        file_size_aff--;
    }
    fclose(ftp_file_aff);
    printf("!!! %s is added !!!\n",ftp_file_name_aff);
}
else
{
    printf("!!! %s is not added & exit !!!\n",ftp_file_name_aff);
}
}

```

```

/* Delete the transferred files from the ~/ftp_file subdirectory */
delete_ftp_file()
{
    char ftp_file_name_dff[17],delete_file_cmd_dff[20];
    char dummy_file_size_dff[1];
    int confirm_flag_dff;
    int file_size_unit_dff,file_size_dff;

    dummy_file_size_dff[0]='\0';
    ftp_file_name_dff[0]='\0';
    delete_file_cmd_dff[0]='\0';
    printf("\n*** File transfer data file listing ***\n\n");
    system("ls -l ftp_file.*MB > file.tmp");
    system("ls -l ftp_file.*KB >> file.tmp");
    file_selection(ftp_file_name_dff,dummy_file_size_dff,2);
    if (strlen(ftp_file_name_dff) > 0)
    {
        printf("\n*** Do you really want to delete this file ");
        printf("transfer data file : %s ==> (Y/N) ",ftp_file_name_dff);
        confirm_flag_dff=confirm_continue();
        if (confirm_flag_dff == 1)
        {
            strcpy(delete_file_cmd_dff,"rm ");

```

```

        strcat(delete_file_cmd_dff,ftp_file_name_dff);
        system(delete_file_cmd_dff);
        printf("!!! %s file transfer data file has been deleted !!!\n",ftp_file_name_dff);
    }
    else
    {
        printf("!!! %s file transfer data file is not deleted !!!\n",ftp_file_name_dff);
    }
}
else
{
    printf("!!! Exit file transfer data file deletion function !!!\n");
}
}

```

```

/* Get the file name, file size unit and file size
   for adding file or deleting file      */
get_file_parameter(file_size_unit_gfp,file_size_gfp,ftp_file_name_gfp)

```

```

int *file_size_unit_gfp;
int *file_size_gfp;
char *ftp_file_name_gfp;

```

```

{
    char file_size_s_gfp[34];
    int opt_buf_gfp;

```

```

while (*file_size_unit_gfp < 1 || *file_size_unit_gfp > 2)
{
    printf("    *** Please input the size unit ");
    printf("of the ftp file ***\n\n");
    printf("        1. Kbytes \n\n");
    printf("        2. Mbytes \n\n");
    printf(" Please select one function (1,2) ==> ");
    scanf("%d",&opt_buf_gfp);
    getchar();
    printf("\n");
    *file_size_unit_gfp=opt_buf_gfp;
    if (*file_size_unit_gfp < 1 || *file_size_unit_gfp > 2)
    {
        printf("!!! The file size unit selection is error !!!\n\n");
    }
}

```

```

    }
    while (*file_size_gfp <= 0)
    {
        if (*file_size_unit_gfp == 1)
        {
            printf("*** Many Kbytes do you want to process ==> ");
            scanf("%d",&opt_buf_gfp);
            getchar();
            *file_size_gfp=opt_buf_gfp;
            printf("\n");
        }
        else
        {
            printf("*** Many Mbytes do you want to create ==> ");
            scanf("%d",&opt_buf_gfp);
            getchar();
            *file_size_gfp=opt_buf_gfp;
            printf("\n");
        }
    }
    sprintf(file_size_s_gfp,"%d",opt_buf_gfp);
    strcat(ftp_file_name_gfp,"ftp_file.");
    strcat(ftp_file_name_gfp,file_size_s_gfp);
    if (*file_size_unit_gfp == 1)
    {
        strcat(ftp_file_name_gfp,"KB");
    }
    else
    {
        strcat(ftp_file_name_gfp,"MB");
    }
    ftp_file_name_gfp[strlen(ftp_file_name_gfp)]='\0';
}

```

```

/* This procedure will execute the network profile queries,
   users can use the network profile to query the network
   hosts' status.                                          */
network_profile_query(argc_npq,argv_npq)

```

```

int argc_npq;
char *argv_npq;

```

```

{
int opt_fun_npq,exit_flag_npq;
char cur_path_npq[256],network_profile_path_npq[256];
FILE *temp_out_npq;

opt_fun_npq=0;
exit_flag_npq=0;
while ((opt_fun_npq < 1) || (opt_fun_npq > 4) || (exit_flag_npq == 0))
{
    cur_path_npq[0]='\0';
    network_profile_path_npq[0]='\0';
    opt_fun_npq=display_profile_query_menu();
    if (opt_fun_npq < 1 || opt_fun_npq > 4)
    {
        printf("!!! network profile query function selection is error !!!\n");
    }
    else
    {
        switch(opt_fun_npq)
        {
            case 1 :
                get_network_profile_path(cur_path_npq,network_profile_path_npq,1);
                if (strlen(cur_path_npq) >= 1 && strlen(network_profile_path_npq) >= 1)
                {
                    profile_reachability_test();
                    if (chdir(cur_path_npq) == -1)
                    {
                        printf("\n!!! Can not return to %s subdirectory !!!\n\n",cur_path_npq);
                    }
                }
                else
                {
                    printf("\n !!! Please check the current ");
                    printf("directory !!!\n");
                }
                break;

            case 2 :
                profile_traffic_query(argc_npq,argv_npq);
                break;

            case 3 :

```



```

        profile_traffic_rating_query(argc_npq,argv_npq);
        break;

    case 4 :
        printf("\n Do you really want to exit ? (Y/N) ");
        exit_flag_npq=confirm_continue();
        break;

    default :
        printf("\n!!! Network profile query function selection is error !!!\n");
        break;
    }
}
}
}
}

```

```

/* This function will display the network profile query menu,
   let users select the network profile query function */

```

```

int display_profile_query_menu()
{
    int opt_fun_dpqm;

    printf("\n\n    <<<<<  Network Profile Query");
    printf("    >>>>> \n\n");
    printf("        1. Reachability Test \n\n");
    printf("        2. Traffic Statistics \n\n");
    printf("        3. Network Node Traffic Status Rating \n\n");
    printf("        4. Exit \n\n");
    printf("        Please select one function (1-4) ==>>> ");
    scanf("%d",&opt_fun_dpqm);
    getchar();
    printf("\n");
    return opt_fun_dpqm;
}

```

```

/* This procedure will read the network nodes from the network
   profile as the 'ping' unix command input data.
   The users can get the reachabilities of the network nodes
   in the network profile */

```

```

profile_reachability_test()
{
    char ping_cmd_prt[256],network_node_prt[256];
    char network_profile_name_prt[45];
    FILE *network_profile_prt;
    int confirm_flag_prt,continue_flag_prt;
    char dummy_file_no_prt[1]; /* This dummy variable for the
                                get_network_profile procedure
                                in this procedure */

    continue_flag_prt=1;
    network_profile_name_prt[0]='\0';
    while (continue_flag_prt == 1)
    {
        get_network_profile(network_profile_name_prt,dummy_file_no_prt,1,
                            &confirm_flag_prt);
        network_profile_prt=fopen(network_profile_name_prt,"r");
        if (confirm_flag_prt == 1)
        {
            printf("\n*** Reachability Report ***\n\n");
            while (!feof(network_profile_prt))
            {
                ping_cmd_prt[0]='\0';
                network_node_prt[0]='\0';
                fscanf(network_profile_prt,"%s",network_node_prt);
                if (strlen(network_node_prt) > 0)
                {
                    strcpy(ping_cmd_prt,"ping ");
                    strcat(ping_cmd_prt,network_node_prt);
                    system(ping_cmd_prt);
                }
            }
            fclose(network_profile_prt);
            printf("\n*** Do you want to query another network profile (Y/N) ");
            continue_flag_prt=confirm_continue();
        }
    }
}

```

/* This procedure will display the network profiles,
the users can select one of them as the input

```

        data file as the network profile query          */
get_network_profile(network_profile_name_gnp,file_no_gnp,proc_id_gnp,
                    confirm_flag_gnp)

char *network_profile_name_gnp;
char *file_no_gnp;
int proc_id_gnp;
int *confirm_flag_gnp;

{
char dummy_file_size_gnp[1],network_node_gnp[256];
FILE *tmp_profile_gnp;

*confirm_flag_gnp=0;
system("ls -l * > file.tmp");
printf("\n*** Network profile listing ***\n\n");
if (proc_id_gnp == 0)
{
file_selection(network_profile_name_gnp,dummy_file_size_gnp,2);
}
else
{
if (proc_id_gnp == 1)
{
file_selection(network_profile_name_gnp,file_no_gnp,3);
}
}
if (strlen(network_profile_name_gnp) > 0)
{
printf("\n*** Do you want to browse the content of this ");
printf("network profile : %s (Y/N) ",network_profile_name_gnp);
*confirm_flag_gnp=confirm_continue();
if (*confirm_flag_gnp == 1)
{
tmp_profile_gnp=fopen(network_profile_name_gnp,"r");
browse_network_profile(tmp_profile_gnp);
fclose(tmp_profile_gnp);
}
printf("*** Do you really want to use this network ");
printf("profile : %s ==> (Y/N) ",network_profile_name_gnp);
*confirm_flag_gnp=confirm_continue();
}
else

```

```

    {
        *confirm_flag_gnp=0;
    }

}

/* This procedure will display the content of the network
   profile which is specified by the users */
browse_network_profile(network_profile_bnp)

FILE *network_profile_bnp;

{
    char rec_buf_bnp[256];

    while (!feof(network_profile_bnp))
    {
        rec_buf_bnp[0]='\0';
        fgets(rec_buf_bnp,255,network_profile_bnp);
        if (strlen(rec_buf_bnp) > 0)
        {
            printf("%s",rec_buf_bnp);
        }
    }
    printf("\n");
}

/* This procedure will provide the real time network traffic query,
   long term network traffic monitoring and long term network
   traffic statistic report */
profile_traffic_query(argc_ptq,argv_ptq)

int argc_ptq;
char *argv_ptq;

{
    char cur_path_ptq[256];
    char long_term_traffic_info_file_name_ptq[45];
    char traffic_stat_path_ptq[256];
    int profile_network_node_no_ptq;

```

```

int opt_fun_ptq,exit_flag_ptq,confirm_flag_ptq;

opt_fun_ptq=0;
exit_flag_ptq=0;
while ((opt_fun_ptq < 1) || (opt_fun_ptq > 4) || (exit_flag_ptq == 0))
{
    opt_fun_ptq=display_traffic_query_menu();
    if (opt_fun_ptq < 1 || opt_fun_ptq > 4)
    {
        printf("!!! network profile traffic query function selection is error !!!\n");
    }
    else
    {
        switch(opt_fun_ptq)
        {
            case 1 :
                profile_traffic_statistics(argc_ptq,argv_ptq);
                break;

            case 2 :
                unix_shell_generation_and_execution(1);
                break;

            case 3 :
                long_term_traffic_info_file_name_ptq[0]='\0';
                profile_network_node_no_ptq=0;
                get_profile_network_node_info(&profile_network_node_no_ptq,
                                                long_term_traffic_info_file_name_ptq,1);

                cur_path_ptq[0]='\0';
                traffic_stat_path_ptq[0]='\0';
                get_network_profile_path(cur_path_ptq,traffic_stat_path_ptq,2);
                if (strlen(cur_path_ptq) >= 1 && strlen(traffic_stat_path_ptq) >=1 )
                {
                    profile_long_term_traffic_statistic_report(profile_network_node_no_ptq,
                                                                long_term_traffic_info_file_name_ptq);

                    if (chdir(cur_path_ptq) == -1)
                    {
                        printf("\n!!! Can not return to %s subdirectory !!!\n\n",
                            cur_path_ptq);
                    }
                }
            else
            {

```

```

        if (strlen(cur_path_ptq) > 0)
        {
            if (chdir(cur_path_ptq) == -1)
            {
                printf("\n!!! Can not return to %s subdirectory !!!\n\n",
                    cur_path_ptq);
            }
        }
        printf("\n !!! Please check the current directory !!!\n");
    }
    break;

case 4 :
    printf("\n Do you really want to exit ? (Y/N) ");
    exit_flag_ptq=confirm_continue();
    break;

default :
    printf("\n!!! Network profile traffic query function selection is error !!!\n");
    break;
}
}
}
}
}

```

/* This procedure will display the network traffic query menu */

```

int display_traffic_query_menu()
{
    int opt_fun_dtqm;

    printf("\n\n      ***** Network Traffic Query Mode Selection");
    printf("\n      *****\n\n");
    printf("      1. real time network traffic query.\n\n");
    printf("      2. long term network traffic monitoring.\n\n");
    printf("      3. long term network traffic statistic report\n\n");
    printf("      4. Exit\n\n");
    printf("      Please select one function (1-4) ===>> ");
    scanf("%d",&opt_fun_dtqm);
    getchar();
    printf("\n");
    return opt_fun_dtqm;
}

```

```
}
```

```
/* This procedure allow users input the parameters for the  
'long term traffic status monitoring' and the 'long term  
network node rating monitoring' in the interactive mode.  
It will generate the unix shell command file according to  
the parameters which is input by the users. The unix shell  
command file will be executed automatically in unix  
background mode */  
unix_shell_generation_and_execution(proc_id_usgae)
```

```
int proc_id_usgae;
```

```
{  
char traffic_log_file_name_usgae[45];  
char network_profile_name_usgae[45];  
char unix_shell_cmd_usgae[256];  
char mkdir_cmd_usgae[256];  
char chmod_cmd_usgae[256];  
char rm_cmd_usgae[256];  
char log_traffic_shell_usgae[256];  
char traffic_stat_path_usgae[256];  
char traffic_rating_path_usgae[256];  
char cur_path_usgae[256];  
char network_profile_path_usgae[256];  
char logging_path_usgae[256];  
char total_measueament_times_s_usgae[34];  
char per_measurement_period_s_usgae[34];  
char per_sampling_number_s_usgae[34];  
char packet_size_s_usgae[34];  
char file_no_usgae[34];  
char long_term_traffic_info_file_name_usgae[45];  
int opt_fun_usgae,exit_flag_usgae,confirm_flag_usgae;  
int create_file_flag_usgae,profile_network_node_no_usgae;  
int contigue_process_flag_usgae;  
FILE *unix_shell_file_usgae,*temp_out_usgae;
```

```
cur_path_usgae[0]='\0';  
network_profile_path_usgae[0]='\0',  
network_profile_name_usgae[0]='\0',  
traffic_stat_path_usgae[0]='\0';
```

```

get_network_profile_path(cur_path_usgae, network_profile_path_usgae, 1);
if (strlen(cur_path_usgae) >= 1 && strlen(network_profile_path_usgae) >= 1)
{
    get_network_profile(network_profile_name_usgae, file_no_usgae, 1,
        &confirm_flag_usgae);
    network_profile_name_usgae[strlen(network_profile_name_usgae)] = '\0';
    if (chdir(cur_path_usgae) == -1 || confirm_flag_usgae != 1)
    {
        if (chdir(cur_path_usgae) == -1)
        {
            printf("\n!!! Can not return to %s subdirectory !!!\n\n", cur_path_usgae);
            printf("!!! Please check it !!!\n");
        }
    }
    else
    {
        cur_path_usgae[0] = '\0';
        if (proc_id_usgae == 1)
        {
            traffic_stat_path_usgae[0] = '\0';
            get_network_profile_path(cur_path_usgae, traffic_stat_path_usgae, 2);
            strcpy(unix_shell_cmd_usgae, traffic_stat_path_usgae);
        }
        else
        {
            if (proc_id_usgae == 2)
            {
                traffic_rating_path_usgae[0] = '\0';
                get_network_profile_path(cur_path_usgae, traffic_rating_path_usgae, 3);
                strcpy(unix_shell_cmd_usgae, traffic_rating_path_usgae);
            }
        }
        if (proc_id_usgae == 1)
        {
            if (strlen(cur_path_usgae) >= 1 && strlen(traffic_stat_path_usgae) >= 1)
            {
                contigue_process_flag_usgae = 1;
            }
            else
            {
                contigue_process_flag_usgae = 0;
            }
        }
    }
}

```



```

    }
    if (proc_id_usgae == 2)
    {
        if (strlen(cur_path_usgae) >= 1 && strlen(traffic_rating_path_usgae) >= 1)
        {
            contigue_process_flag_usgae=1;
        }
        else
        {
            contigue_process_flag_usgae=0;
        }
    }
    if (contigue_process_flag_usgae == 1)
    {
        get_traffic_monitoring_information(network_profile_name_usgae,
                                           traffic_log_file_name_usgae,
                                           total_measuement_times_s_usgae,
                                           per_measurement_period_s_usgae,
                                           packet_size_s_usgae,
                                           per_sampling_number_s_usgae,
                                           &create_file_flag_usgae);

        if (proc_id_usgae == 1)
        {
            strcpy(logging_path_usgae,traffic_stat_path_usgae);
            strcat(logging_path_usgae,"/");
            strcat(logging_path_usgae,network_profile_name_usgae);
        }
        else
        {
            if (proc_id_usgae == 2)
            {
                strcpy(logging_path_usgae,traffic_rating_path_usgae);
                strcat(logging_path_usgae,"/");
                strcat(logging_path_usgae,network_profile_name_usgae);
            }
        }
        mkdir_cmd_usgae[0]='\0';
        strcpy(mkdir_cmd_usgae,"mkdir ");
        strcat(mkdir_cmd_usgae,network_profile_name_usgae);
        mkdir_cmd_usgae[strlen(mkdir_cmd_usgae)]='\0';
        system(mkdir_cmd_usgae);
        strcpy(log_traffic_shell_usgae,logging_path_usgae);
        strcat(log_traffic_shell_usgae,"/log_traffic_shell.bat");
    }

```

```

log_traffic_shell_usgae[strlen(log_traffic_shell_usgae)] =
'\0';
unix_shell_file_usgae =
fopen(log_traffic_shell_usgae, "w");
fprintf(unix_shell_file_usgae, "#! /bin/csh\n");
fprintf(unix_shell_file_usgae, "set index = 1\n");
if (create_file_flag_usgae == 1)
{
    if (proc_id_usgae == 1)
    {
        fprintf(unix_shell_file_usgae, "rm %s/", traffic_stat_path_usgae);
        fprintf(unix_shell_file_usgae, "%s\n", traffic_log_file_name_usgae);
        fprintf(unix_shell_file_usgae, "date >& %s/", traffic_stat_path_usgae);
    }
    else
    {
        if (proc_id_usgae == 2)
        {
            fprintf(unix_shell_file_usgae, "rm %s/", traffic_rating_path_usgae);
            fprintf(unix_shell_file_usgae, "%s\n", traffic_log_file_name_usgae);
            fprintf(unix_shell_file_usgae, "date >& %s/", traffic_rating_path_usgae);
        }
    }
    fprintf(unix_shell_file_usgae, "%s\n", traffic_log_file_name_usgae);
}
fprintf(unix_shell_file_usgae, "while ($index <= ");
fprintf(unix_shell_file_usgae, "%s)\n", total_measurement_times_s_usgae);
if (proc_id_usgae == 1)
{
    fprintf(unix_shell_file_usgae, "date >> %s/", traffic_stat_path_usgae);
}
else
{
    if (proc_id_usgae == 2)
    {
        fprintf(unix_shell_file_usgae, "date >> %s/", traffic_rating_path_usgae);
    }
}
fprintf(unix_shell_file_usgae, "%s\n", traffic_log_file_name_usgae);
fprintf(unix_shell_file_usgae, "%s/", cur_path_usgae);
fprintf(unix_shell_file_usgae, "ftp_cli3 %s >& %s/dummy.out <<+\n",
        logging_path_usgae, logging_path_usgae);
fprintf(unix_shell_file_usgae, "2\n");

```

```

fprintf(unix_shell_file_usgae,"Y\n");
if (proc_id_usgae == 1)
{
    fprintf(unix_shell_file_usgae,"2\n");
}
else
{
    if (proc_id_usgae == 2)
    {
        fprintf(unix_shell_file_usgae,"3\n");
    }
}
fprintf(unix_shell_file_usgae,"I\n");
fprintf(unix_shell_file_usgae,"%s\n",file_no_usgae);
fprintf(unix_shell_file_usgae,"n\n");
fprintf(unix_shell_file_usgae,"Y\n");
fprintf(unix_shell_file_usgae,"%s\n",packet_size_s_usgae);
fprintf(unix_shell_file_usgae,"%s\n",per_sampling_number_s_usgae);
if (proc_id_usgae == 1)
{
    fprintf(unix_shell_file_usgae,"n\n");
}
fprintf(unix_shell_file_usgae,"n\n");
fprintf(unix_shell_file_usgae,"4\n");
fprintf(unix_shell_file_usgae,"Y\n");
fprintf(unix_shell_file_usgae,"4\n");
fprintf(unix_shell_file_usgae,"Y\n");
fprintf(unix_shell_file_usgae,"4\n");
fprintf(unix_shell_file_usgae,"Y\n");
fprintf(unix_shell_file_usgae,"n");
fprintf(unix_shell_file_usgae,"+n");
if (proc_id_usgae == 1)
{
    fprintf(unix_shell_file_usgae,"cat %s/.stat_out.tmp >> ",logging_path_usgae);
    fprintf(unix_shell_file_usgae,"%s/",traffic_stat_path_usgae);
}
else
{
    if (proc_id_usgae == 2)
    {
        fprintf(unix_shell_file_usgae,"cat %s/.network_node_rating.tmp >> ",
            logging_path_usgae);
        fprintf(unix_shell_file_usgae,"%s/",traffic_rating_path_usgae);
    }
}

```

```

    }
}
fprintf(unix_shell_file_usgae,"%s\n",traffic_log_file_name_usgae);
fprintf(unix_shell_file_usgae,"rm %s/dummy.out\n",logging_path_usgae);
fprintf(unix_shell_file_usgae,"sleep ");
fprintf(unix_shell_file_usgae,"%s\n",per_measurement_period_s_usgae);
fprintf(unix_shell_file_usgae,"@ index++\n");
fprintf(unix_shell_file_usgae,"end\n");
rm_cmd_usgae[0]='\0';
strcpy(rm_cmd_usgae,"rm ");
strcat(rm_cmd_usgae,logging_path_usgae);
strcat(rm_cmd_usgae,"/log_traffic_shell.bat");
rm_cmd_usgae[strlen(rm_cmd_usgae)]='\0';
fprintf(unix_shell_file_usgae,"%s\n",rm_cmd_usgae);
rm_cmd_usgae[0]='\0';
strcpy(rm_cmd_usgae,"rm ");
strcat(rm_cmd_usgae,logging_path_usgae);
if (proc_id_usgae == 1)
{
    strcat(rm_cmd_usgae,"/stat_out.tmp");
}
else
{
    if (proc_id_usgae == 2)
    {
        strcat(rm_cmd_usgae,"/network_node_rating.tmp");
    }
}
rm_cmd_usgae[strlen(rm_cmd_usgae)]='\0';
fprintf(unix_shell_file_usgae,"%s\n",rm_cmd_usgae);
rm_cmd_usgae[0]='\0';
strcpy(rm_cmd_usgae,"rmdir ");
strcat(rm_cmd_usgae,logging_path_usgae);
rm_cmd_usgae[strlen(rm_cmd_usgae)]='\0';
fprintf(unix_shell_file_usgae,"%s\n",rm_cmd_usgae);
fclose(unix_shell_file_usgae);
chmod_cmd_usgae[0]='\0';
strcpy(chmod_cmd_usgae,"chmod 700 ");
strcat(chmod_cmd_usgae,log_traffic_shell_usgae);
system(chmod_cmd_usgae);
if (chdir(cur_path_usgae) == -1)
{
    printf("\n!!! Can not return to %s subdirectory !!\n\n",cur_path_usgae);
}

```

```

    }
    unix_shell_cmd_usgae[0]='\0';
    strcpy(unix_shell_cmd_usgae,log_traffic_shell_usgae);
    strcat(unix_shell_cmd_usgae," &");
    unix_shell_cmd_usgae[strlen(unix_shell_cmd_usgae)]='\0';
    system(unix_shell_cmd_usgae);
}
}
else
{
    printf("\n !!! Please check the current directory !!! \n");
}
}

```

```

/* This procedure will get the long term network traffic monitoring
   log file name according to the selections which are chosen by the
   users */

```

```

get_traffic_monitoring_information(network_profile_name_gtmi,
                                   traffic_log_file_name_gtmi,
                                   total_measurement_times_s_gtmi,
                                   per_measurement_period_s_gtmi,
                                   packet_size_s_gtmi,
                                   per_sampling_number_s_gtmi,
                                   create_file_flag_gtmi)

```

```

char *network_profile_name_gtmi;
char *traffic_log_file_name_gtmi;
char *total_measurement_times_s_gtmi;
char *per_measurement_period_s_gtmi;
char *per_sampling_number_s_gtmi;
char *packet_size_s_gtmi;
int *create_file_flag_gtmi;

{
    int opt_buf_gtmi,period_unit_buf_gtmi;
    int per_sampling_number_gtmi,packet_size_gtmi;
    int check_file_flag_gtmi,file_opt_gtmi;
    long period_unit_gtmi;
    long total_periods_gtmi,per_period;
    long total_measurement_times_gtmi,total_measurement_minutes_gtmi;
}

```

```

long measurement_period_unit_buf_gtmi,measurement_period_unit_gtmi;
long measurement_period_gtmi,per_measurement_period_gtmi;
char total_periods_s_gtmi[34];
char measurement_period_s_gtmi[34];

```

```

opt_buf_gtmi=0;
strcpy(traffic_log_file_name_gtmi,network_profile_name_gtmi);
while ((period_unit_buf_gtmi < 1) || (period_unit_buf_gtmi > 5))
{
    printf("*** Plaese select the time unit of traffic ");
    printf("monitoring ***\n\n");
    printf(" 1. minute \n");
    printf(" 2. hour \n");
    printf(" 3. day \n");
    printf(" 4. month(30 days) \n");
    printf(" 5. year \n\n");
    printf(" Please select one time unit (1-5) ");
    printf("==>> ");
    scanf("%d",&period_unit_buf_gtmi);
    getchar();
    printf("\n");
}
total_periods_gtmi=0;
while (total_periods_gtmi < 1)
{
    if ((period_unit_buf_gtmi >= 1) && (period_unit_buf_gtmi <= 5))
    {
        printf("*** How many ");
        switch(period_unit_buf_gtmi)
        {
            case 1 :
                period_unit_gtmi=60;
                printf("minutes");
                break;

            case 2 :
                period_unit_gtmi=3600;
                printf("hours");
                break;

            case 3 :
                period_unit_gtmi=86400;
                printf("days");

```

```

        break;

    case 4 :
        period_unit_gtmi=2592000;
        printf("months");
        break;

    case 5 :
        period_unit_gtmi=31536000;
        printf("years");
        break;

    default :
        printf("!!! the time unit of traffic monitoring selection is error !!!");
        break;
    }
    printf(" do you want to log the network traffic ==> ");
    scanf("%d",&total_periods_gtmi);
    getchar();
    printf("\n");
}

printf(total_periods_s_gtmi,"%ld",total_periods_gtmi);
total_measurement_minutes_gtmi=period_unit_gtmi*total_periods_gtmi;
strcat(traffic_log_file_name_gtmi,"_");
strcat(traffic_log_file_name_gtmi,total_periods_s_gtmi);
strcat(traffic_log_file_name_gtmi,"_");
switch(period_unit_buf_gtmi)
{
    case 1 :
        strcat(traffic_log_file_name_gtmi,"mi","minute");
        break;

    case 2 :
        strcat(traffic_log_file_name_gtmi,"hour");
        break;

    case 3 :
        strcat(traffic_log_file_name_gtmi,"day");
        break;

    case 4 :
        strcat(traffic_log_file_name_gtmi,"month");

```

```

        break;

    case 5 :
        strcat(traffic_log_file_name_gtmi,"year");
        break;

    default :
        break;
}
strcat(traffic_log_file_name_gtmi,"_by_per_");
while ((measurement_period_unit_buf_gtmi < 1) || (measurement_period_unit_buf_gtmi
> 3))
{
    printf("*** Please select one as the one traffic sample");
    printf("time unit ***\n\n");
    printf("  1. minute \n");
    printf("  2. hour \n");
    printf("  3. day \n\n");
    printf("  Please select one time unit (1-3) ");
    printf("====>> ");
    scanf("%d",&measurement_period_unit_buf_gtmi);
    getchar();
    printf("\n");
}
measurement_period_gtmi=0;
while (measurement_period_gtmi < 1)
{
    if ((measurement_period_unit_buf_gtmi >= 1) &&
(measurement_period_unit_buf_gtmi <= 3))
    {
        printf("*** How many ");
        switch(measurement_period_unit_buf_gtmi)
        {
            case 1 :
                measurement_period_unit_gtmi=60;
                printf("minutes");
                break;

            case 2 :
                measurement_period_unit_gtmi=3600;
                printf("hours");
                break;

```



```

        case 3 :
            measurement_period_unit_gtmi=86400;
            printf("days");
            break;

        default :
            printf("!!! the time unit of traffic ");
            printf("monitoring selection is error !!!");
            break;
    }
    printf(" do you want to collect one sample\n");
    printf(" the network traffic status ==> ");
    scanf("%d",&measurement_period_gtmi);
    getchar();
    printf("\n");
}

printf(measurement_period_s_gtmi,"%ld",measurement_period_gtmi);
per_measurement_period_gtmi=measurement_period_unit_gtmi*
    measurement_period_gtmi;
printf(per_measurement_period_s_gtmi,"%ld",per_measurement_period_gtmi);
total_measurement_times_gtmi=total_measurement_minutes_gtmi/
    per_measurement_period_gtmi;
printf(total_measurement_times_s_gtmi,"%ld",total_measurement_times_gtmi);
strcat(traffic_log_file_name_gtmi,measurement_period_s_gtmi);
strcat(traffic_log_file_name_gtmi,"_");
switch(measurement_period_unit_buf_gtmi)
{
    case 1 :
        strcat(traffic_log_file_name_gtmi,"minute");
        break;

    case 2 :
        strcat(traffic_log_file_name_gtmi,"hour");
        break;

    case 3 :
        strcat(traffic_log_file_name_gtmi,"day");
        break;

    default :
        break;
}

```

```

traffic_log_file_name_gtmi[strlen(traffic_log_file_name_gtmi)]='\0';
check_file_flag_gtmi=check_network_profile(traffic_log_file_name_gtmi);
if (check_file_flag_gtmi == 1)
{
    printf("!!! The %s traffic log file already has existed !!!\n",traffic_log_file_name_gtmi);
    *create_file_flag_gtmi=0;
    while (*create_file_flag_gtmi != 1 && *create_file_flag_gtmi != 2)
    {
        printf("*** Do you want to <1> overwrite or <2> append it ==> ");
        scanf("%d",&file_opt_gtmi);
        getchar();
        printf("\n");
        if (file_opt_gtmi != 1 && file_opt_gtmi != 2)
        {
            printf("!!! File operation mode selection is error !!!\n")
        }
        else
        {
            switch(file_opt_gtmi)
            {
                case 1 :
                    *create_file_flag_gtmi=1;
                    printf("!!! The %s traffic log will be overwritten
!!!\n",traffic_log_file_name_gtmi);
                    break;

                    case 2 :
                        *create_file_flag_gtmi=2;
                        printf("!!! The %s traffic log will be appended
!!!\n",traffic_log_file_name_gtmi);
                        break;

                        default :
                            *create_file_flag_gtmi=0;
                            break;
                        }
                    }
            }
        }
    }
else
{
    *create_file_flag_gtmi=1;
}

```

```

packet_size_gtmi=0;
while ((packet_size_gtmi < 10) || (packet_size_gtmi > 1024))
{
    printf("*** Please input the datagram packet size (10~1024)\n");
    printf("    for the per traffic status collection ==> ");
    scanf("%d",&packet_size_gtmi);
    getchar();
    printf("\n");
}
sprintf(packet_size_s_gtmi,"%d",packet_size_gtmi);
per_sampling_number_gtmi=0;
while ((per_sampling_number_gtmi < 5) ||
       (per_sampling_number_gtmi > 100))
{
    printf("*** Please input the sampling number (5~100)\n");
    printf("    in per traffic status collection ==> ");
    scanf("%d",&per_sampling_number_gtmi);
    getchar();
    printf("\n");
}
sprintf(per_sampling_number_s_gtmi,"%d",per_sampling_number_gtmi);
}

/* This procedure will get the long term traffic status file
   name, the network node number in the profile and
   delete the 'stat_out.tmp' temporary file          */
get_profile_network_node_info(profile_network_node_no_gpnni,
                             long_term_traffic_info_file_name_gpnni,
                             proc_id_gpnni)

int *profile_network_node_no_gpnni;
char *long_term_traffic_info_file_name_gpnni;
int proc_id_gpnni;

{
    char cur_path_gpnni[256];
    char network_profile_path_gpnni[256];
    char network_profile_name_gpnni[45];
    char dummy_file_size_gpnni[1];
    char network_node_buf_gpnni[256];
    int char_index_gpnni,network_node_no_gpnni;

```

```

FILE *network_profile_gpnni,*ping_stat_out_gpnni;

network_node_no_gpnni=0;
cur_path_gpnni[0]='\0';
network_profile_path_gpnni[0]='\0';
network_profile_name_gpnni[0]='\0';
network_node_buf_gpnni[0]='\0';
if (proc_id_gpnni == 1)
{
    get_network_profile_path(cur_path_gpnni,
                            network_profile_path_gpnni,2);
}
else
{
    if (proc_id_gpnni == 2)
    {
        get_network_profile_path(cur_path_gpnni,network_profile_path_gpnni,3);
    }
}
if (strlen(cur_path_gpnni) >= 1 &&strlen(network_profile_path_gpnni) >= 1)
{
    system("ls -l * > file.tmp");
    file_selection(long_term_traffic_info_file_name_gpnni,dummy_file_size_gpnni,2);
    char_index_gpnni=0;
    if (strlen(long_term_traffic_info_file_name_gpnni) > 0)
    {
        while (long_term_traffic_info_file_name_gpnni[char_index_gpnni] != '\0')
        {
            network_profile_name_gpnni[char_index_gpnni]=
            long_term_traffic_info_file_name_gpnni[char_index_gpnni];
            char_index_gpnni++;
        }
        network_profile_name_gpnni[char_index_gpnni]='\0';
        if (chdir(cur_path_gpnni) == -1)
        {
            printf("\n!!! Can not return to %s subdirectory !!!\n\n",cur_path_gpnni);
            printf("!!! Please check it !!!\n");
        }
    }
    else
    {
        cur_path_gpnni[0]='\0';
        network_profile_path_gpnni[0]='\0';
        get_network_profile_path(cur_path_gpnni,network_profile_path_gpnni,1);
    }
}

```

```

if (strlen(cur_path_gpnni) >= 1 && strlen(network_profile_path_gpnni) >= 1)
{
    if ((network_profile_gpnni=fopen(network_profile_name_gpnni,"r")) != NULL)
    {
        while (!feof(network_profile_gpnni))
        {
            fscanf(network_profile_gpnni,"%s\n",network_node_buf_gpnni);
            network_node_no_gpnni++;
        }
        *profile_network_node_no_gpnni=network_node_no_gpnni;
        fclose(network_profile_gpnni);
    }
    if (chdir(cur_path_gpnni) == -1)
    {
        printf("\n!!! Can not return to %s subdirectory !!!\n\n",
            cur_path_gpnni);
        printf("!!! Please check it !!!\n");
    }
}
else
{
    printf("\n !!! Please check the current directory !!!\n");
}
}
else
{
    long_term_traffic_info_file_name_gpnni[0]='\0';
    *profile_network_node_no_gpnni=0;
    if (chdir(cur_path_gpnni) == -1)
    {
        printf("\n!!! Can not return to %s subdirectory !!!\n\n",cur_path_gpnni);
        printf("!!! Please check it !!!\n");
    }
}
}
else
{
    if (strlen(cur_path_gpnni) >= 1)
    {
        if (chdir(cur_path_gpnni) == -1)
        {
            printf("\n!!! Can not return to %s subdirectory !!!\n\n",cur_path_gpnni);

```

```

        printf("!!! Please check it !!!\n");
    }
}
else
{
    printf("\n !!! Please check the current directory !!! \n");
}
}

}

/* This procedure will use the 'ping' unix command to get
the traffic situations to the network nodes which are
specified by the network profile.
The users can select the network profile which they
want to measure */
profile_traffic_statistics(argc_pts,argv_pts)

int argc_pts;
char *argv_pts;

{
    FILE *network_profile_pts;
    FILE *ping_stat_pts,*stat_out_pts;
    int confirm_flag_pts,continue_flag_pts,pre_process_flag_pts;
    int profile_network_node_no_pts,index_ctr_pts;
    float packet_received_percent_pts;
    char ping_cmd_pts[256],grep_cmd_pts[256],rm_cmd_pts[256];
    char network_node_pts[256];
    char ping_stat_rec_pts[256];
    char stat_out_file_name_pts[256];
    char ping_stat_file_name_pts[256];
    char full_network_profile_name_pts[256];
    char cur_path_pts[256];
    char network_profile_path_pts[256];
    char traffic_stat_path_pts[256];
    char network_profile_name_pts[45];
    char datagram_packet_size_s_pts[34],count_s_pts[34];
    char transmitted_packet_number_s_pts[34];
    char received_packet_number_s_pts[34];
    char received_packet_number_index_pts[100];
    char loss_packet_percent_s_pts[34];

```

```

char loss_packet_percent_index_pts[100];
char packet_received_percent_s_pts[34];
char packet_received_percent_buf_pts[34];
char min_avg_max_round_trip_time_index_pts[50];
char min_round_trip_time_s_pts[34];
char max_round_trip_time_s_pts[34];
char avg_round_trip_time_s_pts[34];
char dummy_file_no_pts; /* This is a dummy variable for the
                           get_network_profile procedure
                           in this procedure          */

cur_path_pts[0]='\0';
network_profile_path_pts[0]='\0';
continue_flag_pts=1;
while (continue_flag_pts == 1)
{
    get_network_profile_path(cur_path_pts,network_profile_path_pts,1);
    if (strlen(cur_path_pts) >= 1 &&
        strlen(network_profile_path_pts) >= 1)
    {
        get_network_profile(network_profile_name_pts,dummy_file_no_pts,0,
                            &confirm_flag_pts);
        if (confirm_flag_pts == 1)
        {
            strcpy(full_network_profile_name_pts,network_profile_path_pts);
            strcat(full_network_profile_name_pts,"/");
            strcat(full_network_profile_name_pts,network_profile_name_pts);
            if ((network_profile_pts=
                fopen(full_network_profile_name_pts,"r")) == NULL)
            {
                printf("!!! Please check the network profile");
                printf(" %s, Can not open it !!!",full_network_profile_name_pts);
            }
        }
        else
        {
            if (chdir(cur_path_pts) == -1)
            {
                printf("\n!!! Can not return to %s subdirectory !!!\n\n",cur_path_pts);
            }
            else
            {
                if (confirm_flag_pts == 1)
                {

```

```

        traffic_stat_path_pts[0]='\0';
        if (argc_pts > 1)
        {
            strcpy(traffic_stat_path_pts,argv_pts);
        }
        else
        {
            get_network_profile_path(cur_path_pts,traffic_stat_path_pts,2);
        }
        pre_process_flag_pts=1;
    }
}
}
else
{
    if (chdir(cur_path_pts) == -1)
    {
        printf("\n!!! Can not return to %s subdirectory !!!\n",cur_path_pts);
    }
    pre_process_flag_pts=0;
    continue_flag_pts=0;
}
}
else
{
    printf("\n !!! Please check the current directory !!!\n");
}
if (pre_process_flag_pts == 1 && strlen(traffic_stat_path_pts) >= 1)
{
    strcpy(stat_out_file_name_pts,traffic_stat_path_pts);
    strcat(stat_out_file_name_pts,"/.stat_out.tmp");
    stat_out_pts=fopen(stat_out_file_name_pts,"w");
    datagram_packet_size_s_pts[0]='\0';
    count_s_pts[0]='\0';
    get_packet_size_and_count(datagram_packet_size_s_pts,count_s_pts);
    printf("*** Do you want to list ping messages ");
    printf("for each network nodes ==> (Y/N) ");
    confirm_flag_pts=confirm_continue();
    printf("\n!!! Please wait for the traffic statistics ");
    printf("process !!!\n");
    while (!feof(network_profile_pts))
    {

```



```

ping_cmd_pts[0]='\0';
network_node_pts[0]='\0';
transmitted_packet_number_s_pts[0]='\0';
received_packet_number_s_pts[0]='\0';
packet_received_percent_s_pts[0]='\0';
min_round_trip_time_s_pts[0]='\0';
avg_round_trip_time_s_pts[0]='\0';
max_round_trip_time_s_pts[0]='\0';
fscanf(network_profile_pts,"%s",network_node_pts);
if (strlen(network_node_pts) > 0)
{
strcpy(ping_cmd_pts,"ping -s ");
strcat(ping_cmd_pts,network_node_pts);
strcat(ping_cmd_pts," ");
strcat(ping_cmd_pts,datagram_packet_size_s_pts);
strcat(ping_cmd_pts," ");
strcat(ping_cmd_pts,count_s_pts);
strcat(ping_cmd_pts," > ");
strcat(ping_cmd_pts,traffic_stat_path_pts);
strcat(ping_cmd_pts,"./ping_file.tmp");
ping_cmd_pts[strlen(ping_cmd_pts)]='\0';
system(ping_cmd_pts);
if (confirm_flag_pts == 1)
{
printf("\nnetwork host name : %s  ",network_node_pts);
printf("packet size : %s data bytes\n\n",datagram_packet_size_s_pts);
grep_cmd_pts[0]='\0';
strcpy(grep_cmd_pts,"grep icmp ");
strcat(grep_cmd_pts,traffic_stat_path_pts);
strcat(grep_cmd_pts,"./ping_file.tmp");
grep_cmd_pts[strlen(grep_cmd_pts)]='\0';
system(grep_cmd_pts);
printf("\n");
}
strcpy(grep_cmd_pts,"grep packet ");
strcat(grep_cmd_pts,traffic_stat_path_pts);
strcat(grep_cmd_pts,"./ping_file.tmp > ");
strcat(grep_cmd_pts,traffic_stat_path_pts);
strcat(grep_cmd_pts,"./ping_stat.tmp");
system(grep_cmd_pts);
grep_cmd_pts[0]='\0';
strcpy(grep_cmd_pts,"grep min/avg/max ");
strcat(grep_cmd_pts,traffic_stat_path_pts);

```

```

strcat(grep_cmd_pts, ".ping_file.tmp >> ");
strcat(grep_cmd_pts, traffic_stat_path_pts);
strcat(grep_cmd_pts, ".ping_stat.tmp");
system(grep_cmd_pts);
ping_stat_file_name_pts[0]='\0';
strcpy(ping_stat_file_name_pts, traffic_stat_path_pts);
strcat(ping_stat_file_name_pts, ".ping_stat.tmp", "r");
ping_stat_pts=fopen(ping_stat_file_name_pts, "r");
fgets(ping_stat_rec_pts, 255, ping_stat_pts);
ping_stat_rec_pts[strlen(ping_stat_rec_pts)]='\0';
if (strlen(ping_stat_rec_pts) > 1)
{
    index_ctr_pts=0;
    while (ping_stat_rec_pts[index_ctr_pts] != ' ')
    {
        transmitted_packet_number_s_pts[index_ctr_pts]=
            ping_stat_rec_pts[index_ctr_pts];
        index_ctr_pts++;
    }
    transmitted_packet_number_s_pts[index_ctr_pts]='\0';
    strcpy(received_packet_number_index_pts, strstr(ping_stat_rec_pts, "ted, "));
    index_ctr_pts=5;
    while (received_packet_number_index_pts[index_ctr_pts] >= '0' &&
        received_packet_number_index_pts[index_ctr_pts] <= '9')
    {
        received_packet_number_s_pts[index_ctr_pts-5]=
            received_packet_number_index_pts[index_ctr_pts];
        index_ctr_pts++;
    }
    received_packet_number_s_pts[index_ctr_pts-5]='\0';
    strcpy(loss_packet_percent_index_pts, strstr(ping_stat_rec_pts, "ved, "));
    index_ctr_pts=5;
    while (ping_stat_rec_pts[index_ctr_pts] != '%')
    {
        loss_packet_percent_s_pts[index_ctr_pts-5]=
            loss_packet_percent_index_pts[index_ctr_pts];
        index_ctr_pts++;
    }
    loss_packet_percent_s_pts[strlen(loss_packet_percent_s_pts)]='\0';
    packet_received_percent_pts=
        100.0-atof(loss_packet_percent_s_pts);
    packet_received_percent_s_pts[0]='\0';
    if (packet_received_percent_pts <= 70.0)

```

```

    {
        sprintf(packet_received_percent_buf_pts,"%3.2f",
            packet_received_percent_pts);
        strcpy(packet_received_percent_s_pts,"");
        strcat(packet_received_percent_s_pts,packet_received_percent_buf_pts);
        packet_received_percent_s_pts[strlen(packet_received_percent_s_pts)]='\0';
    }
else
    {
        sprintf(packet_received_percent_s_pts,"%3.2f",
            packet_received_percent_pts);
    }
ping_stat_rec_pts[0]='\0';
if (!feof(ping_stat_pts))
    {
        fgets(ping_stat_rec_pts,255,ping_stat_pts);
        ping_stat_rec_pts[strlen(ping_stat_rec_pts)]='\0';
        if (strlen(ping_stat_rec_pts) > 1)
            {
                strcpy(min_avg_max_round_trip_time_index_pts,
                    strchr(ping_stat_rec_pts,'='));
                index_ctr_pts=2;
                while (min_avg_max_round_trip_time_index_pts[index_ctr_pts] != '/')
                    {
                        min_round_trip_time_s_pts[index_ctr_pts-2]=
                            min_avg_max_round_trip_time_index_pts[index_ctr_pts];
                        index_ctr_pts++;
                    }
                index_ctr_pts++;
                min_round_trip_time_s_pts[index_ctr_pts-3]='\0';
                while (min_avg_max_round_trip_time_index_pts[index_ctr_pts] != '/')
                    {
                        avg_round_trip_time_s_pts[index_ctr_pts-3-
                            strlen(min_round_trip_time_s_pts)]=
                            min_avg_max_round_trip_time_index_pts[index_ctr_pts];
                        index_ctr_pts++;
                    }
                index_ctr_pts++;
                avg_round_trip_time_s_pts
                    [index_ctr_pts-4-strlen(min_round_trip_time_s_pts)]='\0';
                while (min_avg_max_round_trip_time_index_pts[index_ctr_pts] >= '0' &&
                    min_avg_max_round_trip_time_index_pts[index_ctr_pts] <= '9')
                    {

```

```

        max_round_trip_time_s_pts
[index_ctr_pts-4-strlen(min_round_trip_time_s_pts)-
strlen(avg_round_trip_time_s_pts)]=
min_avg_max_round_trip_time_index_pts[index_ctr_pts];
index_ctr_pts++;
    }
    max_round_trip_time_s_pts
[index_ctr_pts-4-strlen(min_round_trip_time_s_pts)-
strlen(avg_round_trip_time_s_pts)]='\0';
    }
else
    {
        strcpy(min_round_trip_time_s_pts,"-");
        strcpy(avg_round_trip_time_s_pts,"-");
        strcpy(max_round_trip_time_s_pts,"-");
    }
}
fprintf(stat_out_pts, " %10s %7s %7s %9s %9s %9s %s\n",
    transmitted_packet_number_s_pts,
    received_packet_number_s_pts,
    packet_received_percent_s_pts,
    min_round_trip_time_s_pts,
    avg_round_trip_time_s_pts,
    max_round_trip_time_s_pts,
    network_node_pts);
    }
fclose(ping_stat_pts);
rm_cmd_pts[0]='\0';
strcpy(rm_cmd_pts,"rm ");
strcat(rm_cmd_pts,traffic_stat_path_pts);
strcat(rm_cmd_pts,"/.ping_stat.tmp");
system(rm_cmd_pts);
    }
}
fclose(stat_out_pts);
display_traffic_status_statistics_report(stat_out_file_name_pts);
rm_cmd_pts[0]='\0';
strcpy(rm_cmd_pts,"rm ");
strcat(rm_cmd_pts,traffic_stat_path_pts);
strcat(rm_cmd_pts,"/.ping_file.tmp");
system(rm_cmd_pts);
fclose(network_profile_pts);
if (chdir(cur_path_pts) == -1)

```

```

        {
            printf("\n!!! Can not return to %s subdirectory !!!\n\n",cur_path_pts);
        }
    else
    {
        printf("\n*** Do you want to query another network profile (Y/N) ");
        continue_flag_pts=confirm_continue();
    }
}
}
}

```

```

/* This procedure will display the statistics output of
   the traffic status. */
display_traffic_status_statistics_report(stat_out_file_name_dtssr)

```

```

char *stat_out_file_name_dtssr;

{
    FILE *stat_out_dtssr;
    char stat_out_rec_dtssr[256];

    printf("\n          *** Traffic ");
    printf("Statistics Report ***\n\n");
    printf("transmitted received received minimum   average   ");
    printf("maximum   network node\n");
    printf("packets   packets   packets round-trip round-trip ");
    printf("round-trip\n");
    printf("number   percent percent number   number   ");
    printf("number\n");
    printf("-----  -----  -----  -----  ");
    printf("-----  -----  \n\n");
    stat_out_dtssr=fopen(stat_out_file_name_dtssr,"r");
    while (!feof(stat_out_dtssr))
    {
        stat_out_rec_dtssr[0]='\0';
        fgets(stat_out_rec_dtssr,255,stat_out_dtssr);
        if (strlen(stat_out_rec_dtssr) > 0)
        {
            printf("%s",stat_out_rec_dtssr);
        }
    }
}

```

```

    }
    fclose(stat_out_dtssr);
    printf("\n\n");
}

```

```

struct traffic_info_records
{
    char time[30];
    char received_packet_percent[7];
    char min_round_trip_time[10];
    char avg_round_trip_time[10];
    char max_round_trip_time[10];
    struct traffic_info_records *next;
};

```

```

typedef struct traffic_info_records traffic_info_rec;

```

```

struct long_term_traffic_info_records
{
    char network_node[256];
    struct traffic_info_records *traffic_info_ptr;
    struct long_term_traffic_info_records *next;
};

```

```

typedef struct long_term_traffic_info_records long_term_traffic_info_rec;

```

```

profile_long_term_traffic_statistic_report(profile_network_node_no_plttsr,
                                           long_term_traffic_info_file_name_plttsr)

```

```

int profile_network_node_no_plttsr;
char *long_term_traffic_info_file_name_plttsr;

{
    long_term_traffic_info_rec *head_node_rec_plttsr;
    long_term_traffic_info_rec *tail_node_rec_plttsr;
    long_term_traffic_info_rec *cur_node_rec_plttsr;
    long_term_traffic_info_rec *new_node_rec_plttsr;
    traffic_info_rec *new_rec_plttsr;
    traffic_info_rec *cur_rec_plttsr;
    char time_buf_plttsr[30];
}

```

```

char str_buf1_plttsr[34],str_buf2_plttsr[34];
char received_packet_percent_plttsr[7];
char min_round_trip_time_plttsr[10];
char avg_round_trip_time_plttsr[10];
char max_round_trip_time_plttsr[10];
char network_node_plttsr[256];
int loop_plttsr,loop1_plttsr,block_ctr_plttsr;
FILE *long_term_traffic_statistics;

head_node_rec_plttsr=NULL;
tail_node_rec_plttsr=NULL;
cur_node_rec_plttsr=NULL;
new_node_rec_plttsr=NULL;
if ((long_term_traffic_statistics=
    fopen(long_term_traffic_info_file_name_plttsr,"r")) != NULL)
{
    time_buf_plttsr[0]='\0';
    fgets(time_buf_plttsr,255,long_term_traffic_statistics);
    fgets(time_buf_plttsr,255,long_term_traffic_statistics);
    time_buf_plttsr[strlen(time_buf_plttsr)-1]='\0';
    for (loop_plttsr=0;loop_plttsr < profile_network_node_no_plttsr;loop_plttsr++)
    {
        fscanf(long_term_traffic_statistics,"%s %s %s %s %s %s %s\n",
            str_buf1_plttsr,str_buf2_plttsr,
            received_packet_percent_plttsr,
            min_round_trip_time_plttsr,
            avg_round_trip_time_plttsr,
            max_round_trip_time_plttsr,
            network_node_plttsr);
        new_rec_plttsr=(struct traffic_info_records *)
            malloc (sizeof(struct traffic_info_records));
        strcpy(new_rec_plttsr->time,time_buf_plttsr);
        strcpy(new_rec_plttsr->received_packet_percent,received_packet_percent_plttsr);
        strcpy(new_rec_plttsr->min_round_trip_time,min_round_trip_time_plttsr);
        strcpy(new_rec_plttsr->avg_round_trip_time,avg_round_trip_time_plttsr);
        strcpy(new_rec_plttsr->max_round_trip_time,max_round_trip_time_plttsr);
        new_rec_plttsr->next=NULL;
        new_node_rec_plttsr=(struct long_term_traffic_info_records *)
            malloc (sizeof(struct long_term_traffic_info_records));
        strcpy(new_node_rec_plttsr->network_node,network_node_plttsr);
        new_node_rec_plttsr->traffic_info_ptr=new_rec_plttsr;
        new_node_rec_plttsr->next=NULL;
        if (head_node_rec_plttsr == NULL)

```

```

    {
        head_node_rec_plttsr=new_node_rec_plttsr;
        tail_node_rec_plttsr=new_node_rec_plttsr;
    }
else
    {
        tail_node_rec_plttsr->next=new_node_rec_plttsr;
        tail_node_rec_plttsr=new_node_rec_plttsr;
    }
}
block_ctr_plttsr=1;
while (!feof(long_term_traffic_statistics))
{
    cur_node_rec_plttsr=head_node_rec_plttsr;
    fgets(time_buf_plttsr,255,long_term_traffic_statistics);
    time_buf_plttsr[strlen(time_buf_plttsr)-1]='\0';
    for (loop_plttsr=0;loop_plttsr<profile_network_node_no_plttsr;loop_plttsr++)
    {
        fscanf(long_term_traffic_statistics,"%s %s %s %s %s %s %s\n",
            str_buf1_plttsr,str_buf2_plttsr,
            received_packet_percent_plttsr,
            min_round_trip_time_plttsr,
            avg_round_trip_time_plttsr,
            max_round_trip_time_plttsr,
            network_node_plttsr);
        new_rec_plttsr=(struct traffic_info_records *)
        malloc (sizeof(struct traffic_info_records));
        strcpy(new_rec_plttsr->time,time_buf_plttsr);
        strcpy(new_rec_plttsr->received_packet_percent,received_packet_percent_plttsr);
        strcpy(new_rec_plttsr->min_round_trip_time,min_round_trip_time_plttsr);
        strcpy(new_rec_plttsr->avg_round_trip_time,avg_round_trip_time_plttsr);
        strcpy(new_rec_plttsr->max_round_trip_time,max_round_trip_time_plttsr);
        new_rec_plttsr->next=NULL;
        cur_rec_plttsr=cur_node_rec_plttsr->traffic_info_ptr;
        for (loop1_plttsr=1;loop1_plttsr < block_ctr_plttsr;loop1_plttsr++)
        {
            cur_rec_plttsr=cur_rec_plttsr->next;
        }
        cur_rec_plttsr->next=new_rec_plttsr;
        if (cur_node_rec_plttsr != tail_node_rec_plttsr)
        {
            cur_node_rec_plttsr=cur_node_rec_plttsr->next;
        }
    }
}

```



```

    }
    block_ctr_plttsr++;
}
display_long_term_statistics_report(head_node_rec_plttsr
                                   tail_node_rec_plttsr, 1);

free(head_node_rec_plttsr);
free(tail_node_rec_plttsr);
free(new_node_rec_plttsr);
free(cur_node_rec_plttsr);
}
}

display_long_term_statistics_report(head_node_rec_dlttsr, tail_node_rec_dlttsr)

long_term_traffic_info_rec *head_node_rec_dlttsr;
long_term_traffic_info_rec *tail_node_rec_dlttsr;

{
    long_term_traffic_info_rec *cur_node_rec_dlttsr;
    traffic_info_rec *cur_rec_dlttsr;

    cur_node_rec_dlttsr = head_node_rec_dlttsr;
    printf("\n      ***** Long Term Traffic Statistics Report *****\n\n");
    while (cur_node_rec_dlttsr != tail_node_rec_dlttsr)
    {
        cur_rec_dlttsr = cur_node_rec_dlttsr->traffic_info_ptr;
        printf("\n\n network node name : %s \n\n",
              cur_node_rec_dlttsr->network_node);
        printf("          received ");
        printf("minimum   average   maximum\n");
        printf("time           packet ");
        printf("round-trip round-trip round-trip\n");
        printf("          percent ");
        printf("time     time     time\n");
        printf("----- ");
        printf("----- \n");
        while (cur_rec_dlttsr->next != NULL)
        {
            printf(" %30s %8s %10s %10s %10s\n",
                  cur_rec_dlttsr->time,
                  cur_rec_dlttsr->received_packet_percent,
                  cur_rec_dlttsr->min_round_trip_time,

```

```

        cur_rec_dltsr->avg_round_trip_time,
        cur_rec_dltsr->max_round_trip_time);
    cur_rec_dltsr=cur_rec_dltsr->next;
}
printf(" %30s %8s %10s %10s %10s\n",
    cur_rec_dltsr->time,
    cur_rec_dltsr->received_packet_percent,
    cur_rec_dltsr->min_round_trip_time,
    cur_rec_dltsr->avg_round_trip_time,
    cur_rec_dltsr->max_round_trip_time);
cur_node_rec_dltsr=cur_node_rec_dltsr->next;
}
printf("\n\n network node name : %s\n\n",cur_node_rec_dltsr->network_node);
printf("                received ");
printf("minimum    average    maximum\n");
printf("time                packet ");
printf("round-trip round-trip round-trip\n");
printf("                percent ");
printf("time    time    time\n");
printf("----- ");
printf("-----\n");
cur_rec_dltsr=cur_node_rec_dltsr->traffic_info_ptr;
while (cur_rec_dltsr->next != NULL)
{
    printf(" %30s %8s %10s %10s %10s\n",
        cur_rec_dltsr->time,
        cur_rec_dltsr->received_packet_percent,
        cur_rec_dltsr->min_round_trip_time,
        cur_rec_dltsr->avg_round_trip_time,
        cur_rec_dltsr->max_round_trip_time);
    cur_rec_dltsr=cur_rec_dltsr->next;
}
printf(" %30s %8s %10s %10s %10s\n",
    cur_rec_dltsr->time,
    cur_rec_dltsr->received_packet_percent,
    cur_rec_dltsr->min_round_trip_time,
    cur_rec_dltsr->avg_round_trip_time,
    cur_rec_dltsr->max_round_trip_time);
}

```

/* This procedure will let users query the real time network

```

node traffic rating, monitor the long term network node
traffic rating information and generate the long term
traffic rating statistics report. */
profile_traffic_rating_query(argc_ptrq,argv_ptrq)

int argc_ptrq;
char *argv_ptrq;

{
int profile_network_node_no_ptrq;
int opt_fun_ptrq,exit_flag_ptrq,confirm_flag_ptrq;
char cur_path_ptrq[256],network_profile_path_ptrq[256];
char traffic_rating_path_ptrq[256];
char long_term_traffic_rating_file_name_ptrq[45];

opt_fun_ptrq=0;
exit_flag_ptrq=0;
while ((opt_fun_ptrq < 1) || (opt_fun_ptrq > 4) || (exit_flag_ptrq == 0))
{
opt_fun_ptrq=display_traffic_rating_query_menu();
if (opt_fun_ptrq < 1 || opt_fun_ptrq > 4)
{
printf("!!! network profile traffic rating query function selection is error !!!\n");
}
else
{
switch(opt_fun_ptrq)
{
case 1 :
profile_traffic_rating(argc_ptrq,argv_ptrq);
break;

case 2 :
unix_shell_generation_and_execution(2);
break;

case 3 :
long_term_traffic_rating_file_name_ptrq[0]='\0';
profile_network_node_no_ptrq=0;
get_profile_network_node_info(&profile_network_node_no_ptrq,
long_term_traffic_rating_file_name_ptrq,2);
cur_path_ptrq[0]='\0';
traffic_rating_path_ptrq[0]='\0';
}
}
}
}

```

```

get_network_profile_path(cur_path_ptrq,traffic_rating_path_ptrq,3);
if (strlen(cur_path_ptrq) >= 1 && strlen(traffic_rating_path_ptrq) >=1 )
{
    profile_long_term_traffic_rating_report
    (profile_network_node_no_ptrq,long_term_traffic_rating_file_name_ptrq);
    if (chdir(cur_path_ptrq) == -1)
    {
        printf("\n!!! Can not return to %s subdirectory !!!\n\n",cur_path_ptrq);
    }
}
else
{
    printf("\n !!! Please check the current directory !!!\n");
}
break;

case 4 :
    printf("\n Do you really want to exit ? (Y/N) ");
    exit_flag_ptrq=confirm_continue();
    break;

default :
    printf("\n!!! Network profile traffic query ");
    printf("function selection is error !!!\n");
    break;
}

}

}

}

/* This procedure will display the network traffic rating query menu */
int display_traffic_rating_query_menu()
{
    int opt_fun_dtrqm;

    printf("\n\n ***** Network Traffic Rating Query Mode Selection *****\n\n");
    printf("    1. real time network node rating traffic query \n\n");
    printf("    2. long term network node rating traffic monitoring \n\n");
    printf("    3. long term network node traffic rating statistic report\n\n");
    printf("    4. Exit \n\n");
    printf("    Please select one function (1-4) ==>>> ");

```

```

scanf("%d",&opt_fun_dtrqm);
getchar();
printf("\n");
return opt_fun_dtrqm;

```

```

}

```

```

struct network_node_rating_record
{
    char network_node_name[256];
    float round_trip_time_mean_value;
    float round_trip_time_variance;
    float round_trip_time_std_deviation;
    float packet_received_percent;
    struct network_node_rating_record *next;
};

```

```

typedef struct network_node_rating_record network_node_rating_rec;

```

```

/* This procedure will show the network nodes' traffic
   status rating */
profile_traffic_rating(argc_ptr,argv_ptr)

```

```

int argc_ptr;
char *argv_ptr;

```

```

{
    char ping_cmd_ptr[256],rm_cmd_ptr[256];
    char network_node_ptr[256],rating_out_file_name_ptr[256];
    char full_network_profile_name_ptr[256];
    char ping_data_rec_ptr[256];
    char traffic_rating_path_ptr[256];
    char cur_path_ptr[256],network_profile_path_ptr[256];
    char datagram_packet_size_s_ptr[34],count_s_ptr[34];
    char round_trip_time_index_ptr[25],round_trip_time_s_ptr[34];
    char loss_packet_percent_index_ptr[25];
    char loss_packet_percent_s_ptr[5];
    char traffic_status_file_name_ptr[45];
    char network_profile_name_ptr[45];
    FILE *traffic_status_ptr,*rating_out_ptr,*network_profile_ptr;
    int ping_data_rec_ctr_ptr,index_ctr_ptr;

```

```

int confirm_flag_ptr,continue_flag_ptr,pre_process_flag_ptr;
float round_trip_time_ptr;
double sum_round_trip_time_ptr,sqr_sum_round_trip_time_ptr;
struct stat traffic_status_file_stat_ptr;
int check_traffic_status_file_ptr;
int check_traffic_status_stat_res_ptr;
network_node_rating_rec *head_rec_ptr,*new_rec_ptr;
char dummy_file_no_ptr[1]; /* This is a dummy variable for the
                           get_network_profile procedure
                           in this procedure */

cur_path_ptr[0]='\0';
network_profile_path_ptr[0]='\0';
continue_flag_ptr=1;
pre_process_flag_ptr=0;
while (continue_flag_ptr == 1)
{
    get_network_profile_path(cur_path_ptr,network_profile_path_ptr,1);
    if (strlen(cur_path_ptr) >= 1 && strlen(network_profile_path_ptr) >= 1)
    {
        printf("\n*** Network profile listing ***\n\n");
        get_network_profile(network_profile_name_ptr,dummy_file_no_ptr,0,
                           &confirm_flag_ptr);
        if (confirm_flag_ptr == 1)
        {
            strcpy(full_network_profile_name_ptr,network_profile_path_ptr);
            strcat(full_network_profile_name_ptr,"/");
            strcat(full_network_profile_name_ptr,network_profile_name_ptr);
            if ((network_profile_ptr=
                fopen(full_network_profile_name_ptr,"r")) == NULL)
            {
                printf("!!! Please check the network profile");
                printf(" %s, Can not open it !!!",full_network_profile_name_ptr);
            }
        }
        else
        {
            if (chdir(cur_path_ptr) == -1)
            {
                printf("\n!!! Can not return to %s subdirectory !!\n\n",cur_path_ptr);
            }
        }
        else
        {
            if (confirm_flag_ptr == 1)
            {

```

```

        traffic_rating_path_ptr[0]='\0';
        if (argc_ptr > 1)
        {
            strcpy(traffic_rating_path_ptr,argv_ptr);
        }
        else
        {
            get_network_profile_path(cur_path_ptr,traffic_rating_path_ptr,3);
        }
        pre_process_flag_ptr=1;
    }
}
}
else
{
    if (chdir(cur_path_ptr) == -1)
    {
        printf("\n!!! Can not return to %s subdirectory !!!\n\n",cur_path_ptr);
    }
    pre_process_flag_ptr=0;
    continue_flag_ptr=0;
}
}
else
{
    printf("\n!!! Please check the current directory !!!\n");
}
if (pre_process_flag_ptr == 1 && strlen(traffic_rating_path_ptr) >= 1)
{
    network_profile_name_ptr[0]='\0';
    head_rec_ptr=NULL;
    strcpy(rating_out_file_name_out_ptr,traffic_rating_path_ptr);
    strcat(rating_out_file_name_out_ptr,"/network_node_rating.tmp");
    rating_out_ptr=fopen(rating_out_file_name_out_ptr,"w");
    datagram_packet_size_s_ptr[0]='\0';
    count_s_ptr[0]='\0';
    get_packet_size_and_count(datagram_packet_size_s_ptr,count_s_ptr);
    printf("\n!!! Please wait for the network nodes status rating process !!!\n");
    while (!feof(network_profile_ptr))
    {
        ping_cmd_ptr[0]='\0';
        network_node_ptr[0]='\0';
    }
}

```

```

fscanf(network_profile_ptr,"%s",network_node_ptr);
if (strlen(network_node_ptr) > 0)
{
    strcpy(ping_cmd_ptr,"ping -s ");
    strcat(ping_cmd_ptr,network_node_ptr);
    strcat(ping_cmd_ptr," ");
    strcat(ping_cmd_ptr,datagram_packet_size_s_ptr);
    strcat(ping_cmd_ptr," ");
    strcat(ping_cmd_ptr,count_s_ptr);
    strcat(ping_cmd_ptr," > ");
    strcat(ping_cmd_ptr,traffic_rating_path_ptr);
    strcat(ping_cmd_ptr,"/");
    strcat(ping_cmd_ptr,network_node_ptr);
    strcat(ping_cmd_ptr,".tmp");
    system(ping_cmd_ptr);
}
}
rewind(network_profile_ptr);
while (!feof(network_profile_ptr))
{
    traffic_status_file_name_ptr[0]='\0';
    network_node_ptr[0]='\0';
    fscanf(network_profile_ptr,"%s",network_node_ptr);
    if (strlen(network_node_ptr) > 0)
    {
        strcpy(traffic_status_file_name_ptr,traffic_rating_path_ptr);
        strcat(traffic_status_file_name_ptr,"/");
        strcat(traffic_status_file_name_ptr,network_node_ptr);
        strcat(traffic_status_file_name_ptr,".tmp");
        traffic_status_file_name_ptr[strlen(traffic_status_file_name_ptr)]='\0';
        check_traffic_status_file_ptr=open(traffic_status_file_name_ptr,O_RDONLY);
        check_traffic_status_stat_res_ptr=
        fstat(check_traffic_status_file_ptr,&traffic_status_file_stat_ptr);
        if (check_traffic_status_stat_res_ptr == 0 &&
            traffic_status_file_stat_ptr.st_size > 0)
        {
            close(check_traffic_status_file_ptr);
            traffic_status_ptr=fopen(traffic_status_file_name_ptr,"r");
            fgets(ping_data_rec_ptr,255,traffic_status_ptr);
            fgets(ping_data_rec_ptr,255,traffic_status_ptr);
            ping_data_rec_ctr_ptr=0;
            sum_round_trip_time_ptr=0.0;
            sqr_sum_round_trip_time_ptr=0.0;

```



```

if (strstr(ping_data_rec_ptr,"icmp_seq=") != NULL)
{
    rewind(traffic_status_ptr);
    fgets(ping_data_rec_ptr,255,traffic_status_ptr);
    while (strlen(ping_data_rec_ptr) > 1)
    {
        ping_data_rec_ptr[0]='\0';
        fgets(ping_data_rec_ptr,255,traffic_status_ptr);
        if (strlen(ping_data_rec_ptr) > 1)
        {
            round_trip_time_index_ptr[0]='\0';
            strcpy(round_trip_time_index_ptr,strchr(ping_data_rec_ptr,'='));
            index_ctr_ptr=1;
            round_trip_time_s_ptr[0]='\0';
            while (round_trip_time_index_ptr[index_ctr_ptr] >= '0' &&
                    round_trip_time_index_ptr[index_ctr_ptr] <= '9')
            {
                round_trip_time_s_ptr
                [index_ctr_ptr-1]=round_trip_time_index_ptr[index_ctr_ptr];
                index_ctr_ptr++;
            }
            round_trip_time_s_ptr
            [index_ctr_ptr-1]='\0';
            round_trip_time_ptr=
            atof(round_trip_time_s_ptr);
            sum_round_trip_time_ptr=
            sum_round_trip_time_ptr +
            round_trip_time_ptr;
            sqr_sum_round_trip_time_ptr=sqr_sum_round_trip_time_ptr +
            (round_trip_time_ptr * round_trip_time_ptr);
            ping_data_rec_ctr_ptr++;
        }
    }
}

new_rec_ptr=(struct network_node_rating_record *)
malloc (sizeof(struct network_node_rating_record));
new_rec_ptr->next=NULL;
strcpy(new_rec_ptr->network_node_name,network_node_ptr);
new_rec_ptr->round_trip_time_mean_value=
sum_round_trip_time_ptr/(atof(count_s_ptr));
new_rec_ptr->round_trip_time_variance=
((atof(count_s_ptr)*sqr_sum_round_trip_time_ptr)-
(sum_round_trip_time_ptr*sum_round_trip_time_ptr))/

```

```

(atof(count_s_ptr)*(atof(count_s_ptr)-1.0));
new_rec_ptr->round_trip_time_std_deviation=
sqrt((double) new_rec_ptr->round_trip_time_variance);
ping_data_rec_ptr[0]='\0';
fgets(ping_data_rec_ptr,255,traffic_status_ptr);
fgets(ping_data_rec_ptr,255,traffic_status_ptr);
if (strlen(ping_data_rec_ptr) > 0)
{
    loss_packet_percent_index_ptr[0]='\0';
    strcpy(loss_packet_percent_index_ptr,strstr(ping_data_rec_ptr,"ved. "));
    index_ctr_ptr=5;
    while (loss_packet_percent_index_ptr[index_ctr_ptr] >= '0' &&
           loss_packet_percent_index_ptr[index_ctr_ptr] <= '9')
    {
        loss_packet_percent_s_ptr[index_ctr_ptr-5]=
        loss_packet_percent_index_ptr[index_ctr_ptr];
        index_ctr_ptr++;
    }
    loss_packet_percent_s_ptr[index_ctr_ptr-5]='\0';
    new_rec_ptr->packet_received_percent=100.0-
    atof(loss_packet_percent_s_ptr);
}
fclose(traffic_status_ptr);
ping_cmd_ptr[0]='\0';
}
else
{
    close(check_traffic_status_file_ptr);
}
rm_cmd_ptr[0]='\0';
strcpy(rm_cmd_ptr,"rm ");
strcat(rm_cmd_ptr,traffic_rating_path_ptr);
strcat(rm_cmd_ptr,"/");
strcat(rm_cmd_ptr,network_node_ptr);
strcat(rm_cmd_ptr,".tmp");
rm_cmd_ptr[strlen(rm_cmd_ptr)]='\0';
system(rm_cmd_ptr);
network_node_traffic_status_rating(&head_rec_ptr,new_rec_ptr);
}
}
display_traffic_status_rating(head_rec_ptr,traffic_rating_path_ptr,
                             datagram_packet_size_s_ptr,count_s_ptr);
fclose(network_profile_ptr);

```

```

    free(head_rec_ptr);
    free(new_rec_ptr);
    if (chdir(cur_path_ptr) == -1)
    {
        printf("\n!!! Can not return to %s subdirectory !!!\n\n",cur_path_ptr);
    }
    else
    {
        printf("\n*** Do you want to use another network profile \n");
        printf("    to get another network traffic rating (Y/N) ");
        continue_flag_ptr=confirm_continue();
    }
}
}
}

```

```

typedef struct network_node_rating_record *network_node_rating_rec_ptr;

```

```

/* This procedure will rate the traffic status of network nodes.

```

```

    It will rate the network nodes' traffic status according to
    the following conditions :

```

1. the packet received rate.
2. the variance and standard deviation of the traffic status. */

```

network_node_traffic_status_rating(head_rec_nntsr,new_rec_nntsr)

```

```

network_node_rating_rec_ptr *head_rec_nntsr;
network_node_rating_rec_ptr new_rec_nntsr;

```

```

{
    network_node_rating_rec_ptr ins_rec_nntsr;
    network_node_rating_rec_ptr prev_ins_rec_nntsr;

```

```

    ins_rec_nntsr=NULL;
    prev_ins_rec_nntsr=NULL;
    if (*head_rec_nntsr == NULL)
    {
        *head_rec_nntsr=new_rec_nntsr;
    }

```

```

    else
    {
        if (new_rec_nntsr->packet_received_percent >
            (*head_rec_nntsr)->packet_received_percent)

```

```

    {
        new_rec_nntsr->next=*head_rec_nntsr;
        *head_rec_nntsr=new_rec_nntsr;
    }
else
    {
        if ((*head_rec_nntsr)->next == NULL)
        {
            if (new_rec_nntsr->packet_received_percent ==
                (*head_rec_nntsr)->packet_received_percent)
            {
                if (new_rec_nntsr->round_trip_time_variance >=
                    (*head_rec_nntsr)->round_trip_time_variance)
                {
                    (*head_rec_nntsr)->next=new_rec_nntsr;
                }
            }
            else
            {
                new_rec_nntsr->next=*head_rec_nntsr;
                *head_rec_nntsr=new_rec_nntsr;
            }
        }
        else
        {
            (*head_rec_nntsr)->next=new_rec_nntsr;
        }
    }
else
    {
        if ((new_rec_nntsr->packet_received_percent ==
            (*head_rec_nntsr)->packet_received_percent) &&
            (new_rec_nntsr->round_trip_time_variance <
            (*head_rec_nntsr)->round_trip_time_variance))
        {
            new_rec_nntsr->next=*head_rec_nntsr;
            *head_rec_nntsr=new_rec_nntsr;
        }
        else
        {
            ins_rec_nntsr=(*head_rec_nntsr)->next;
            prev_ins_rec_nntsr=*head_rec_nntsr;
            while (new_rec_nntsr->packet_received_percent <
                ins_rec_nntsr->packet_received_percent &&

```

```

        ins_rec_nntsr->next != NULL)
    {
        ins_rec_nntsr=ins_rec_nntsr->next;
        prev_ins_rec_nntsr=prev_ins_rec_nntsr->next;
    }
while ((new_rec_nntsr->packet_received_percent ==
        ins_rec_nntsr->packet_received_percent) &&
        (new_rec_nntsr->round_trip_time_variance >=
        ins_rec_nntsr->round_trip_time_variance) &&
        ins_rec_nntsr->next != NULL)
    {
        ins_rec_nntsr=ins_rec_nntsr->next;
        prev_ins_rec_nntsr=prev_ins_rec_nntsr->next;
    }
if (ins_rec_nntsr->packet_received_percent ==
    new_rec_nntsr->packet_received_percent)
    {
        if (new_rec_nntsr->round_trip_time_variance >=
            ins_rec_nntsr->round_trip_time_variance)
            {
                if (ins_rec_nntsr->next != NULL)
                {
                    new_rec_nntsr=ins_rec_nntsr->next;
                }
                ins_rec_nntsr->next=new_rec_nntsr;
            }
        else
        {
            new_rec_nntsr->next=ins_rec_nntsr;
            prev_ins_rec_nntsr->next=new_rec_nntsr;
        }
    }
else
    {
        if (ins_rec_nntsr->packet_received_percent >
            new_rec_nntsr->packet_received_percent)
            {
                if (ins_rec_nntsr->next != NULL)
                {
                    new_rec_nntsr=ins_rec_nntsr->next;
                }
                ins_rec_nntsr->next=new_rec_nntsr;
            }
    }

```



```

strcat(full_network_node_rating_file_name_dtsr, ".network_node_rating.tmp");
network_node_rating=fopen(full_network_node_rating_file_name_dtsr, "w");
printf("\n ***** Network Node Traffic Status Rating");
printf(" Report *****\n\n");
printf("packet size : %7s data bytes    ", datagram_packet_size_s_dtsr);
printf("sample number : %6s times\n\n", count_s_dtsr);
printf("      received round-trip round-trip    ");
printf(" round-trip network\n");
printf(" rating packet time      time      ");
printf(" time\n");
printf("      percent mean-value variance      ");
printf(" std-deviation node\n");
printf(" -----");
printf(" ----- \n\n");
rec_ctr_dtsr=1;
while (head_rec_dtsr->next != NULL)
{
    packet_received_percent_s_dtsr[0]='\0';
    round_trip_time_mean_value_s_dtsr[0]='\0';
    round_trip_time_variance_s_dtsr[0]='\0';
    round_trip_time_std_deviation_s_dtsr[0]='\0';
    if (head_rec_dtsr->packet_received_percent <= 70.0)
    {
        strcpy(packet_received_percent_s_dtsr, "");
        sprintf(packet_received_percent_buf_s_dtsr, "%3.2f",
            head_rec_dtsr->packet_received_percent);
        strcat(packet_received_percent_s_dtsr, packet_received_percent_buf_s_dtsr);
        packet_received_percent_s_dtsr[strlen(packet_received_percent_s_dtsr)]='\0';
    }
    else
    {
        sprintf(packet_received_percent_s_dtsr, "%3.2f",
            head_rec_dtsr->packet_received_percent);
    }
    sprintf(round_trip_time_mean_value_s_dtsr, "%7.3f",
        head_rec_dtsr->round_trip_time_mean_value);
    sprintf(round_trip_time_variance_s_dtsr, "%14.3f",
        head_rec_dtsr->round_trip_time_variance);
    sprintf(round_trip_time_std_deviation_s_dtsr, "%7.3f",
        head_rec_dtsr->round_trip_time_std_deviation);
    if (rec_ctr_dtsr > 1)
    {
        if (head_rec_dtsr->packet_received_percent !=

```

```

        packet_received_percent_buf_dtsr ||
        head_rec_dtsr->round_trip_time_variance !=
        round_trip_time_variance_buf_dtsr)
    {
        rating_index_dtsr++;
    }
}
sprintf(rating_s_dtsr,"%6d",rating_index_dtsr);
printf(" %6s %6s %11s %18s %13s %s\n",
    rating_s_dtsr,
    packet_received_percent_s_dtsr,
    round_trip_time_mean_value_s_dtsr,
    round_trip_time_variance_s_dtsr,
    round_trip_time_std_deviation_s_dtsr,
    head_rec_dtsr->network_node_name);
fprintf(network_node_rating," %6s %6s %11s %18s %13s %s\n",
    rating_s_dtsr,
    packet_received_percent_s_dtsr,
    round_trip_time_mean_value_s_dtsr,
    round_trip_time_variance_s_dtsr,
    round_trip_time_std_deviation_s_dtsr,
    head_rec_dtsr->network_node_name);
packet_received_percent_buf_dtsr=head_rec_dtsr->packet_received_percent;
round_trip_time_variance_buf_dtsr=head_rec_dtsr->round_trip_time_variance;
head_rec_dtsr=head_rec_dtsr->next;
rec_ctr_dtsr++;
}
packet_received_percent_s_dtsr[0]='\0';
round_trip_time_mean_value_s_dtsr[0]='\0';
round_trip_time_variance_s_dtsr[0]='\0';
round_trip_time_std_deviation_s_dtsr[0]='\0';
if (head_rec_dtsr->packet_received_percent <= 70.0)
{
    strcpy(packet_received_percent_s_dtsr,"");
    sprintf(packet_received_percent_buf_s_dtsr,"%3.2f",
        head_rec_dtsr->packet_received_percent);
    strcat(packet_received_percent_s_dtsr,packet_received_percent_buf_s_dtsr);
    packet_received_percent_s_dtsr[strlen(packet_received_percent_s_dtsr)]='\0';
}
else
{
    sprintf(packet_received_percent_s_dtsr,"%3.2f",
        head_rec_dtsr->packet_received_percent);
}

```



```

    }
    sprintf(round_trip_time_mean_value_s_dtsr,"%7.3f",
            head_rec_dtsr->round_trip_time_mean_value);
    sprintf(round_trip_time_variance_s_dtsr,"%14.3f",
            head_rec_dtsr->round_trip_time_variance);
    sprintf(round_trip_time_std_deviation_s_dtsr,"%7.3f",
            head_rec_dtsr->round_trip_time_std_deviation);
    if (rec_ctr_dtsr > 1)
    {
        if (head_rec_dtsr->packet_received_percent != packet_received_percent_buf_dtsr ||
            head_rec_dtsr->round_trip_time_variance != round_trip_time_variance_buf_dtsr)
        {
            rating_index_dtsr++;
        }
    }
    sprintf(rating_s_dtsr,"%6d",rating_index_dtsr);
    printf(" %6s %6s %11s %18s %13s %s\n",
        rating_s_dtsr,
        packet_received_percent_s_dtsr,
        round_trip_time_mean_value_s_dtsr,
        round_trip_time_variance_s_dtsr,
        round_trip_time_std_deviation_s_dtsr,
        head_rec_dtsr->network_node_name);
    fprintf(network_node_rating,"%6s %6s %11s %18s %13s %s\n",
        rating_s_dtsr,
        packet_received_percent_s_dtsr,
        round_trip_time_mean_value_s_dtsr,
        round_trip_time_variance_s_dtsr,
        round_trip_time_std_deviation_s_dtsr,
        head_rec_dtsr->network_node_name);
    fclose(network_node_rating);
}

```

```

struct traffic_rating_records
{
    char time[30];
    char rating[6];
    char received_packet_percent[7];
    char round_trip_time_mean_value[12];
    char round_trip_time_variance[19];
    char round_trip_time_std_deviation[12];
}

```

```

    struct traffic_rating_records *next;
};

typedef struct traffic_rating_records
    traffic_rating_rec;

struct long_term_traffic_rating_records
{
    char network_node[256];
    struct traffic_rating_records *traffic_rating_ptr;
    struct long_term_traffic_rating_records *next;
};

typedef struct long_term_traffic_rating_records long_term_traffic_rating_rec;

/* This procedure will generate the 'long term traffic rating report' */
profile_long_term_traffic_rating_report(profile_network_node_no_plttr,
                                        long_term_traffic_rating_file_name_plttr)

int profile_network_node_no_plttr;
char *long_term_traffic_rating_file_name_plttr;

{
    long_term_traffic_rating_rec *head_node_rec_plttr;
    long_term_traffic_rating_rec *tail_node_rec_plttr;
    long_term_traffic_rating_rec *cur_node_rec_plttr;
    long_term_traffic_rating_rec *new_node_rec_plttr;
    traffic_rating_rec *new_rec_plttr;
    traffic_rating_rec *cur_rec_plttr;
    char time_buf_plttr[30];
    char rating_plttr[6];
    char received_packet_percent_plttr[7];
    char round_trip_time_mean_value_plttr[12];
    char round_trip_time_variance_plttr[19];
    char round_trip_time_std_deviation_plttr[12];
    char network_node_plttr[256];
    int loop_plttr, loop1_plttr, block_ctr_plttr;
    FILE *long_term_traffic_rating_plttr;

    head_node_rec_plttr=NULL;
    tail_node_rec_plttr=NULL;
    cur_node_rec_plttr=NULL;
    new_node_rec_plttr=NULL;

```

```

if ((long_term_traffic_rating_plttr=
fopen(long_term_traffic_rating_file_name_plttr,"r"))
!= NULL)
{
time_buf_plttr[0]='\0';
fgets(time_buf_plttr,255,long_term_traffic_rating_plttr);
fgets(time_buf_plttr,255,long_term_traffic_rating_plttr);
time_buf_plttr[strlen(time_buf_plttr)-1]='\0';
for (loop_plttr=0;
loop_plttr < profile_network_node_no_plttr;
loop_plttr++)
{
fscanf(long_term_traffic_rating_plttr,"%s %s %s %s %s %s\n",
rating_plttr,
received_packet_percent_plttr,
round_trip_time_mean_value_plttr,
round_trip_time_variance_plttr,
round_trip_time_std_deviation_plttr,
network_node_plttr);
new_rec_plttr=(struct traffic_rating_records *)
malloc (sizeof(struct traffic_rating_records));
strcpy(new_rec_plttr->time,time_buf_plttr);
strcpy(new_rec_plttr->rating,rating_plttr);
strcpy(new_rec_plttr->received_packet_percent,received_packet_percent_plttr);
strcpy(new_rec_plttr->round_trip_time_mean_value,
round_trip_time_mean_value_plttr);
strcpy(new_rec_plttr->round_trip_time_variance,round_trip_time_variance_plttr);
strcpy(new_rec_plttr->round_trip_time_std_deviation,
round_trip_time_std_deviation_plttr);
new_rec_plttr->next=NULL;
new_node_rec_plttr=(struct long_term_traffic_rating_records *)
malloc (sizeof(struct long_term_traffic_rating_records));
strcpy(new_node_rec_plttr->network_node,network_node_plttr);
new_node_rec_plttr->traffic_rating_ptr=new_rec_plttr;
new_node_rec_plttr->next=NULL;
if (head_node_rec_plttr == NULL)
{
head_node_rec_plttr=new_node_rec_plttr;
tail_node_rec_plttr=new_node_rec_plttr;
}
else
{
tail_node_rec_plttr->next=new_node_rec_plttr;

```

```

        tail_node_rec_plttr=new_node_rec_plttr;
    }
}
block_ctr_plttr=1;
while (!feof(long_term_traffic_rating_plttr))
{
    fgets(time_buf_plttr,255,long_term_traffic_rating_plttr);
    time_buf_plttr[strlen(time_buf_plttr)-1]='\0';
    for (loop_plttr=0;loop_plttr < profile_network_node_no_plttr;loop_plttr++)
    {
        fscanf(long_term_traffic_rating_plttr,"%s %s %s %s %s %s\n",
            rating_plttr,
            received_packet_percent_plttr,
            round_trip_time_mean_value_plttr,
            round_trip_time_variance_plttr,
            round_trip_time_std_deviation_plttr,
            network_node_plttr);
        new_rec_plttr=(struct traffic_rating_records *)
        malloc (sizeof(struct traffic_rating_records));
        strcpy(new_rec_plttr->time,time_buf_plttr);
        strcpy(new_rec_plttr->rating,rating_plttr);
        strcpy(new_rec_plttr->received_packet_percent,
            received_packet_percent_plttr);
        strcpy(new_rec_plttr->round_trip_time_mean_value,
            round_trip_time_mean_value_plttr);
        strcpy(new_rec_plttr->round_trip_time_variance,
            round_trip_time_variance_plttr);
        strcpy(new_rec_plttr->round_trip_time_std_deviation,
            round_trip_time_std_deviation_plttr);
        new_rec_plttr->next=NULL;
        cur_node_rec_plttr=head_node_rec_plttr;
        while (strcmp(cur_node_rec_plttr->network_node,network_node_plttr) != 0 &&
            cur_node_rec_plttr != tail_node_rec_plttr)
        {
            cur_node_rec_plttr=cur_node_rec_plttr->next;
        }
        cur_rec_plttr=cur_node_rec_plttr->traffic_rating_ptr;
        for (loop1_plttr=1;loop1_plttr < block_ctr_plttr;loop1_plttr++)
        {
            cur_rec_plttr=cur_rec_plttr->next;
        }
        cur_rec_plttr->next=new_rec_plttr;
    }
}

```

```

        block_ctr_plttrr++;
    }
    display_long_term_rating_report(head_node_rec_plttrr,tail_node_rec_plttrr);
    free(head_node_rec_plttrr);
    free(tail_node_rec_plttrr);
    free(new_node_rec_plttrr);
    free(cur_node_rec_plttrr);
}
}

display_long_term_rating_report(head_node_rec_dlttr,tail_node_rec_dlttr)

long_term_traffic_rating_rec *head_node_rec_dlttr;
long_term_traffic_rating_rec *tail_node_rec_dlttr;

{
    long_term_traffic_rating_rec *cur_node_rec_dlttr;
    traffic_rating_rec *cur_rec_dlttr;

    cur_node_rec_dlttr=head_node_rec_dlttr;
    printf("\n      ***** Long Term Network Node Traffic Status Rating");
    printf(" Report *****\n\n");
    while (cur_node_rec_dlttr != tail_node_rec_dlttr)
    {
        cur_rec_dlttr=cur_node_rec_dlttr->traffic_rating_ptr;
        printf("\n\n network node name : %s\n\n",cur_node_rec_dlttr->network_node);
        printf("                      received ");
        printf(" round-trip round-trip round-trip\n");
        printf(" time          rating packet ");
        printf(" time      time      time\n");
        printf("                      percent ");
        printf(" mean-value variance      std-deviation\n");
        printf(" -----");
        printf(" ----- \n");
        while (cur_rec_dlttr->next != NULL)
        {
            printf(" %30s %6s %8s %11s %18s %11s\n",
                cur_rec_dlttr->time,
                cur_rec_dlttr->rating,
                cur_rec_dlttr->received_packet_percent,
                cur_rec_dlttr->round_trip_time_mean_value,
                cur_rec_dlttr->round_trip_time_variance,

```

```

        cur_rec_dltr->round_trip_time_std_deviation);
        cur_rec_dltr=cur_rec_dltr->next;
    }
    printf(" %30s %6s %8s %11s %18s %11s\n",
        cur_rec_dltr->time,
        cur_rec_dltr->rating,
        cur_rec_dltr->received_packet_percent,
        cur_rec_dltr->round_trip_time_mean_value,
        cur_rec_dltr->round_trip_time_variance,
        cur_rec_dltr->round_trip_time_std_deviation);
        cur_node_rec_dltr=cur_node_rec_dltr->next;
    }
    printf("\n\n network node name : %s\n\n",cur_node_rec_dltr->network_node);
    printf("                                received ");
    printf(" round-trip round-trip round-trip\n");
    printf(" time rating packet ");
    printf(" time time time\n");
    printf(" percent ");
    printf(" mean-value variance std-deviation\n");
    printf(" -----");
    printf(" -----");
    cur_rec_dltr=cur_node_rec_dltr->traffic_rating_ptr;
    while (cur_rec_dltr->next != NULL)
    {
        printf(" %30s %6s %8s %11s %18s %11s\n",
            cur_rec_dltr->time,
            cur_rec_dltr->rating,
            cur_rec_dltr->received_packet_percent,
            cur_rec_dltr->round_trip_time_mean_value,
            cur_rec_dltr->round_trip_time_variance,
            cur_rec_dltr->round_trip_time_std_deviation);
        cur_rec_dltr=cur_rec_dltr->next;
    }
    printf(" %30s %6s %8s %11s %18s %11s\n",
        cur_rec_dltr->time,
        cur_rec_dltr->rating,
        cur_rec_dltr->received_packet_percent,
        cur_rec_dltr->round_trip_time_mean_value,
        cur_rec_dltr->round_trip_time_variance,
        cur_rec_dltr->round_trip_time_std_deviation);
}

```

```

/* This procedure will let users select the maintenance function
   of network profiles */
network_profile_maintain()
{
    int opt_fun_npm,exit_flag_npm;
    char cur_path_npm[256],network_profile_path_npm[256];

    opt_fun_npm=0;
    exit_flag_npm=0;
    while ((opt_fun_npm < 1) || (opt_fun_npm > 4) || (exit_flag_npm == 0))
    {
        cur_path_npm[0]='\0';
        network_profile_path_npm[0]='\0';
        opt_fun_npm=display_profile_maintenance_menu();
        if (opt_fun_npm < 1 || opt_fun_npm > 4)
        {
            printf("!!! network profile maintenance function selection is error !!!\n");
        }
        else
        {
            switch(opt_fun_npm)
            {
                case 1 :
                    get_network_profile_path(cur_path_npm,network_profile_path_npm,1);
                    if (strlen(cur_path_npm) >= 1 && strlen(network_profile_path_npm) >= 1)
                    {
                        add_network_profile();
                        if (chdir(cur_path_npm) == -1)
                        {
                            printf("\n!!! Can not return to %s subdirectory !!!\n\n",cur_path_npm);
                        }
                    }
                    else
                    {
                        printf("\n !!! Please check the current directory !!!\n");
                    }
                    break;

                case 2 :
                    get_network_profile_path(cur_path_npm,
                                            network_profile_path_npm,1);
                    if (strlen(cur_path_npm) >= 1 && strlen(network_profile_path_npm) >= 1)

```

```

        {
            delete_network_profile();
            if (chdir(cur_path_npm) == -1)
            {
                printf("\n!!! Can not return to subdirectory !!!%s ",cur_path_npm);
            }
        }
    else
    {
        printf("\n !!! Please check the current directory !!!\n");
    }
    break;

case 3 :
    get_network_profile_path(cur_path_npm,network_profile_path_npm,1);
    if (strlen(cur_path_npm) >= 1 && strlen(network_profile_path_npm) >= 1)
    {
        update_network_profile();
        if (chdir(cur_path_npm) == -1)
        {
            printf("\n!!! Can not return to %s subdirectory !!!\n\n",cur_path_npm);
        }
    }
    else
    {
        printf("\n !!! Please check the current directory !!!\n");
    }
    break;

case 4 :
    printf("\n Do you really want to exit ? (Y/N) ");
    exit_flag_npm=confirm_continue();
    break;

default :
    printf("\n!!! Network profile maintenamce function selection is error !!!\n");
    break;
}
}
}

printf("\n\n!!!! Exit the Network Profile Maintenance ");
printf("Utility !!!!\n\n");
getchar();

```



```
)
```

```
/* This function will display the profile maintenance menu,  
   let users select the profile maintenance function */
```

```
int display_profile_maintenance_menu()  
{  
    int opt_fun_dpmm;  
  
    printf("\n\n    <<<<<  Network Profile Maintain");  
    printf("    >>>>> \n\n");  
    printf("        1. Add a network profile \n\n");  
    printf("        2. Delete a network profile \n\n");  
    printf("        3. Update a network profile \n\n");  
    printf("        4. Exit \n\n\n");  
    printf("        Please select one function (1-4) ==>> ");  
    scanf("%d",&opt_fun_dpmm);  
    getchar();  
    printf("\n");  
    return opt_fun_dpmm;  
}
```

```
/* This procedure will return the current path and the network  
   profile path */
```

```
get_network_profile_path(cur_path_gnpp,network_profile_path_gnpp,proc_id_gnpp)
```

```
char *cur_path_gnpp;  
char *network_profile_path_gnpp;  
int proc_id_gnpp;  
  
{  
    int create_profile_subdirectory_gnpp;  
    char mkdir_cmd_gnpp[256];  
  
    create_profile_subdirectory_gnpp=0;  
    if (getcwd(cur_path_gnpp,255) == NULL)  
    {  
        printf("\n !!!  Can not get the current directory !!! \n");  
    }  
    else
```

```

{
network_profile_path_gnpp[0]='\0';
strcpy(network_profile_path_gnpp,cur_path_gnpp);
switch(proc_id_gnpp)
{
case 1 :
    strcat(network_profile_path_gnpp,"/profile");
    break;

case 2 :
    strcat(network_profile_path_gnpp,"/traffic_statistics");
    break;

case 3 :
    strcat(network_profile_path_gnpp,"/traffic_rating");
    break;

default :
    printf("!!! The subdirectory is not changed !!!\n");
    break;
}
network_profile_path_gnpp[strlen(network_profile_path_gnpp)]='\0';
if (chdir(network_profile_path_gnpp) == -1)
{
printf("\n !!! Can not change the %s subdirectory !!!\n",network_profile_path_gnpp);
printf("**** Do you want to create the ");
switch(proc_id_gnpp)
{
case 1 :
    printf("'profile' ");
    break;

case 2 :
    printf("'traffic_statistics' ");
    break;

case 3 :
    printf("'traffic_rating' ");
    break;

default :
    break;
}
}

```

```

printf("subdirectory ==> (Y/N) ");
create_profile_subdirectory_gnpp=confirm_continue();
if (create_profile_subdirectory_gnpp == 1)
{
    mkdir_cmd_gnpp[0]='\0';
    strcpy(mkdir_cmd_gnpp,"mkdir ");
    strcat(mkdir_cmd_gnpp,network_profile_path_gnpp);
    mkdir_cmd_gnpp[strlen(mkdir_cmd_gnpp)]='\0';
    if (system(mkdir_cmd_gnpp) == 256)
    {
        printf("!!! Please check the %s subdirectory !!!\n",network_profile_path_gnpp);
    }
    else
    {
        printf("!!! Please check the %s subdirectory !!!\n",network_profile_path_gnpp);
    }
}
}
}
}
}

```

```

/* this procedure will let users add a network profile */
add_network_profile()

```

```

{
    char network_profile_name_anp[45];
    char network_node_anp[256],tmp_node_anp[256];
    int profile_check_flag_anp,confirm_flag_anp;
    int continue_flag_anp,check_special_char_flag_anp;
    long inaddr_anp;
    FILE *network_profile_anp;

    continue_flag_anp=1;
    profile_check_flag_anp=0;
    network_profile_name_anp[0]='\0';
    network_node_anp[0]='\0';
    tmp_node_anp[0]='\0';
    while (continue_flag_anp == 1)
    {
        check_special_char_flag_anp=1;
        while (check_special_char_flag_anp == 1)
        {

```

```

printf("\n*** Please input a file name for a new network profile ==> \n ");
scanf("%s",network_profile_name_anp);
getchar();
printf("\n!!! Check profile name, please wait !!!\n");
if (strchr(network_profile_name_anp,'_') != NULL)
{
    check_special_char_flag_anp=1;
    printf("!!! New network profile name contains ");
    printf("the '_' special character !!!\n");
    printf("!!! please input another network profile name !!!");
}
else
{
    check_special_char_flag_anp=0;
}
}
profile_check_flag_anp=check_network_profile(network_profile_name_anp);
confirm_flag_anp=0;
switch (profile_check_flag_anp)
{
    case 0 :
        printf("\n*** Do you want to create this network ");
        printf("profile : %s (Y/N) ",network_profile_name_anp);
        confirm_flag_anp=confirm_continue();
        if (confirm_flag_anp == 1)
        {
            printf("!!! %s network profile add is proceeding
!!!\n\n",network_profile_name_anp);
            network_profile_anp=fopen(network_profile_name_anp,"w");
            printf("*** Please input the symbolic name/IP address of a network node
***\n\n");
            printf("!!! Please type <exit> or <quit> to ");
            printf("end network node input !!!\n\n");
            network_node_anp[0]='\0';
            while (strcmp(network_node_anp,"exit") != 0 &&
                    strcmp(network_node_anp,"quit") != 0)
            {
                printf("add profile record > ");
                scanf("%s",network_node_anp);
                getchar();
                printf("\n");
                if (strcmp(network_node_anp,"exit") != 0 &&
                    strcmp(network_node_anp,"quit") != 0)

```

```

        {
            strcpy(tmp_node_anp, network_node_anp);
            check_network_node(tmp_node_anp, network_profile_name_anp,
                               &confirm_flag_anp);
            if (confirm_flag_anp == 1)
            {
                fprintf(network_profile_anp, "%s\n", network_node_anp);
                printf("\n!!! %s is written into ", network_node_anp);
                printf("the %s file", network_profile_name_anp);
                printf(" !!!\n");
            }
            else
            {
                printf("\n!!! %s won't be written ", network_node_anp);
                printf("into the %s file", network_profile_name_anp);
                printf(" !!!\n");
            }
        }
    }
    fclose(network_profile_anp);
    break;

case 1 :
    printf("!!! Network profile : %s has existed !!!", network_profile_name_anp);
    break;

case 2 :
    printf("!!! Network profile : %s is error, ", network_profile_name_anp);
    printf("Please check it !!!\n");
    break;

default :
    break;
}
printf("\n*** Do you want to add another network profile (Y/N) ");
continue_flag_anp = confirm_continue();
}

}

/* This procedure will let users delete the network profile

```

```

        which is selected by the users                */
delete_network_profile()
{
    char network_profile_name_dnp[45],dummy_file_size_dnp[1];
    char delete_file_cmd_dnp[256];
    FILE *del_network_profile_dnp;
    int confirm_flag_dnp,continue_flag_dnp;

    network_profile_name_dnp[0]='\0';
    dummy_file_size_dnp[0]='\0';
    delete_file_cmd_dnp[0]='\0';
    confirm_flag_dnp=0;
    continue_flag_dnp=1;
    while (continue_flag_dnp == 1)
    {
        system("ls -l * > file.tmp");
        printf("\n*** Network profile listing ***\n\n");
        file_selection(network_profile_name_dnp,dummy_file_size_dnp,2);
        if (strlen(network_profile_name_dnp) == 0)
        {
            printf("\n!!! Exit network profile delete function !!!\n");
            continue_flag_dnp=0;
        }
        else
        {
            printf("\n*** Do you want to browse the %s network ",network_profile_name_dnp);
            printf("profile\n");
            printf("    to make sure it is correct one ==> (Y/N) ");
            confirm_flag_dnp=confirm_continue();
            if (confirm_flag_dnp == 1)
            {
                del_network_profile_dnp=fopen(network_profile_name_dnp,"r");
                browse_network_profile(del_network_profile_dnp);
                fclose(del_network_profile_dnp);
            }
            printf("\n*** Do you really want to delete this network ");
            printf("profile : %s ==> (Y/N) ",network_profile_name_dnp);
            confirm_flag_dnp=confirm_continue();
            if (confirm_flag_dnp == 1)
            {
                strcpy(delete_file_cmd_dnp,"rm ");
                strcat(delete_file_cmd_dnp,network_profile_name_dnp);
                system(delete_file_cmd_dnp);
            }
        }
    }
}

```

```

        printf("!!! %s network profile has been deleted !!!\n",network_profile_name_dnp);
    }
    else
    {
        printf("!!! %s network profile is not deleted !!!\n",network_profile_name_dnp);
    }
    printf("\n*** Do you want to delete another network profile (Y/N) ");
    continue_flag_dnp=confirm_continue();
}
}
}

```

```

update_network_profile()
{
    char network_profile_name_unp[45],dummy_file_size_unp[1];
    char update_file_cmd_unp[256];
    char network_node_unp[256],tmp_node_unp[256];
    int network_node_ctr_unp,loop_unp,opt_fun_unp;
    int confirm_flag_unp,exit_flag_unp,continue_flag_unp;
    FILE *update_network_profile_unp;

    network_profile_name_unp[0]='\0';
    dummy_file_size_unp[0]='\0';
    network_node_unp[0]='\0';
    tmp_node_unp[0]='\0';
    confirm_flag_unp=0;
    exit_flag_unp=0;
    continue_flag_unp=1;
    while (continue_flag_unp == 1)
    {
        printf("\n*** Network profile listing ***\n\n");
        system("ls -l * > file.tmp");
        file_selection(network_profile_name_unp,dummy_file_size_unp,2);
        if (strlen(network_profile_name_unp) > 0)
        {
            printf("\n*** Do you really want to update this network ");
            printf("profile : %s ==> (Y/N) ",network_profile_name_unp);
            confirm_flag_unp=confirm_continue();
            if (confirm_flag_unp == 1)
            {
                if ((update_network_profile_unp=

```

```

    fopen(network_profile_name_unp,"r")) == NULL)
{
    printf("!!! Can not open the network profile : %s",network_profile_name_unp);
    printf(" !!\n");
}
else
{
    while ((opt_fun_unp < 1) || (opt_fun_unp > 4) || (exit_flag_unp == 0))
    {
        printf("\n*** Update function ***\n\n");
        printf("    1. Add a network node\n");
        printf("    2. Delete a network node\n");
        printf("    3. Update a network node\n");
        printf("    4. Exit\n\n");
        printf("    Please select one function (1-4) ==>> ");
        scanf("%d",&opt_fun_unp);
        getchar();
        printf("\n");
        switch(opt_fun_unp)
        {
            case 1 :
                update_network_profile_unp=fopen(network_profile_name_unp,"a");
                continue_flag_unp=1;
                while (continue_flag_unp == 1)
                {
                    printf("\n*** Please input symbolic printf("name/IP address of a new ");
                    printf("network node ==> ");
                    scanf("%s",network_node_unp);
                    getchar();
                    strcpy(tmp_node_unp,network_node_unp);
                    check_network_node
                    (tmp_node_unp,network_profile_name_unp,
                     &confirm_flag_unp);
                    if (confirm_flag_unp == 1)
                    {
                        fprintf(update_network_profile_unp,
                                "%s\n",network_node_unp);
                        printf("!!! %s is written into ",network_node_unp);
                        printf("the %s file !!!",network_profile_name_unp);
                    }
                }
            else
            {
                printf("!!! %s won't be written ",network_node_unp);

```



```

        printf("into the %s file !!!\n",network_profile_name_unp);
    }
    printf("\n*** Do you want to add another network node (Y/N) ");
    continue_flag_unp=confirm_continue();
}
fclose(update_network_profile_unp);
break;

case 2 :
    delete_or_update_network_node(network_profile_name_unp,1);
    break;

case 3 :
    delete_or_update_network_node(network_profile_name_unp,2);
    break;

case 4 :
    printf("\n Do you really want to exit ? (Y/N) ");
    exit_flag_unp=confirm_continue();
    break;

default :
    printf("\n Function selection is error, please check it !!!\n");
    break;
}
}
}
else
{
    printf("!!! %s network profile is not updated !!!\n",network_profile_name_unp);
}
printf("\n*** Do you want to update another network profile (Y/N) ");
continue_flag_unp=confirm_continue();
}
else
{
    printf("\n!!! Exit network profile update function");
    printf("\n !!!\n");
    continue_flag_unp=0;
}
}

```

```
}
```

```
/* This function will check the network profiles in the ~/profile
   subdirectory. If the profile has existed in the subdirectory,
   then this function will return 1 to the caller procedure.
   Otherwise, it will return 0 to the caller procedure. */
int check_network_profile(network_profile_name_cnp)
```

```
char *network_profile_name_cnp;
```

```
{
char check_cmd_cnp[256];
int check_file_cnp,profile_stat_res_cnp;
struct stat profile_stat_cnp;

strcpy(check_cmd_cnp,"ls -la * | grep ");
strcat(check_cmd_cnp,network_profile_name_cnp);
strcat(check_cmd_cnp," > check_profile.tmp");
check_cmd_cnp[strlen(check_cmd_cnp)]='\0';
system(check_cmd_cnp);
check_file_cnp=open("check_profile.tmp",O_RDONLY);
if (check_file_cnp == -1)
{
printf("\n !!! Can not open the file : check_profile.tmp !!!\n");
close(check_file_cnp);
}
else
{
profile_stat_res_cnp=fstat(check_file_cnp, &profile_stat_cnp);
if (profile_stat_res_cnp == 0)
{
if (profile_stat_cnp.st_size == 0)
{
system("rm check_profile.tmp");
close(check_file_cnp);
return 0;
}
else
{
system("rm check_profile.tmp");
close(check_file_cnp);
return 1;
}
}
}
```

```

    }
}
else
{
    system("rm check_profile.tmp");
    close(check_file_cnp);
    return 2;
}
}

}

/* This procedure will check the network node name/IP address
   and ask users to confirm the new network node add
   when the name/IP address may be error */
check_network_node(network_node_cnn, network_profile_name_cnn,
                   confirm_flag_cnn)

char *network_node_cnn;
char *network_profile_name_cnn;
int *confirm_flag_cnn;

{
    long inaddr_cnn;
    struct hostent *dummy_ptr_cnn;
    /* this is a dummy variable for the convert_host_name_to_address
       procedure and convert_host_address_information procedure
       in this procedure. */

    if ((inaddr_cnn = inet_addr(network_node_cnn)) == INADDR_NONE)
    {
        convert_host_name_to_address(network_node_cnn, &dummy_ptr_cnn);
    }
    else
    {
        convert_host_address_information(network_node_cnn, &dummy_ptr_cnn);
    }
    if (strlen(network_node_cnn) == 0)
    {
        printf("!!! Network host name/address may not exist !!!\n\n");
        printf("*** Do you want to add it into ");
        printf("the %s network profile ==> (Y/N) ", network_profile_name_cnn);
    }
}

```

```

        *confirm_flag_cnn=confirm_continue();
    }
    else
    {
        *confirm_flag_cnn=1;
    }
}

struct network_node_record
{
    char          data[256];
    struct network_node_record *next;
};

typedef struct network_node_record network_node_rec;

/* This procedure will let users delete a record or
   update a record from the network profile */
delete_or_update_network_node(network_profile_name_dounn,proc_id_dounn)

char *network_profile_name_dounn;
int proc_id_dounn;

{
    char update_network_node_dounn[256],tmp_node_dounn[256];
    char new_rec_buf_dounn[256];
    int confirm_flag_dounn,continue_flag_dounn;
    int network_node_ctr_dounn,loop_dounn,write_ctr_dounn;
    int update_network_node_no_dounn,update_rec_flag_dounn;
    FILE *update_network_profile_dounn;
    network_node_rec *new_rec_ptr_dounn,*tmp_rec_ptr_dounn;
    network_node_rec *head_ptr_dounn,*tail_ptr_dounn;
    network_node_rec *upt_rec_ptr_dounn;

    confirm_flag_dounn=0;
    continue_flag_dounn=1;
    while (continue_flag_dounn == 1)
    {
        new_rec_ptr_dounn=NULL;
        head_ptr_dounn=NULL;
        tail_ptr_dounn=NULL;
        upt_rec_ptr_dounn=NULL;

```

```

update_network_profile_downn=
fopen(network_profile_name_downn,"r");
printf("*** list the network nodes in the %s ***\n\n",network_profile_name_downn);
network_node_ctr_downn=1;
while (!feof(update_network_profile_downn))
{
    new_rec_buf_downn[0]='\0';
    fgets(new_rec_buf_downn,255,
        update_network_profile_downn);
    if (strlen(new_rec_buf_downn) > 0)
    {
        new_rec_ptr_downn=(struct network_node_record *) malloc
            (sizeof(struct network_node_record));
        new_rec_buf_downn[strlen(new_rec_buf_downn)]='\0';
        strcpy(new_rec_ptr_downn->data,new_rec_buf_downn);
        new_rec_ptr_downn->next=NULL;
        if (head_ptr_downn == NULL)
        {
            head_ptr_downn=new_rec_ptr_downn;
        }
        else
        {
            if (head_ptr_downn->next == NULL)
            {
                head_ptr_downn->next=new_rec_ptr_downn;
            }
            else
            {
                tail_ptr_downn->next=new_rec_ptr_downn;
            }
        }
        tail_ptr_downn=new_rec_ptr_downn;
        printf("    %d. ",network_node_ctr_downn);
        printf("%s",new_rec_ptr_downn->data);
        network_node_ctr_downn++;
    }
}
fclose(update_network_profile_downn);
printf("\n\n");
printf("    Please select one network node,<%d> to exit ==>
",network_node_ctr_downn);
update_network_node_no_downn=0;
while (update_network_node_no_downn > network_node_ctr_downn

```

```

        || update_network_node_no_dounn < 1
        || network_node_ctr_dounn == 0)
    {
        scanf("%d",&update_network_node_no_dounn);
        getchar();
        if ((update_network_node_no_dounn > network_node_ctr_dounn)
            || (update_network_node_no_dounn < 1))
        {
            printf("!!! Network node selection is error !!!\n");
            printf("\n Please select one again ==> ");
        }
    }
    if (update_network_node_no_dounn < network_node_ctr_dounn)
    {
        if (proc_id_dounn == 2)
        {
            update_network_node_dounn[0]='\0';
            printf("*** Please input the new network node to update ==> ");
            scanf("%s",update_network_node_dounn);
            getchar();
            printf("\n");
            strcpy(tmp_node_dounn,update_network_node_dounn);
            check_network_node(tmp_node_dounn,network_profile_name_dounn,
                              &confirm_flag_dounn);
            if (confirm_flag_dounn == 1)
            {
                printf("!!! %s is written into ",update_network_node_dounn);
                printf("the %s file !!!\n",network_profile_name_dounn);
                update_rec_flag_dounn=1;
            }
            else
            {
                printf("!!! %s won't be written into ",update_network_node_dounn);
                printf("the %s file !!!",
                    network_profile_name_dounn);update_rec_flag_dounn=0;
            }
            update_network_node_dounn[strlen(update_network_node_dounn)]='\0';
        }
        if (update_network_node_no_dounn == 1)
        {
            if (proc_id_dounn == 1)
            {
                head_ptr_dounn=head_ptr_dounn->next;
            }
        }
    }

```

```

    }
else
{
    if (proc_id_dounn == 2)
    {
        head_ptr_dounn->data[0]='\0';
        strcpy(head_ptr_dounn->data,update_network_node_dounn);
        head_ptr_dounn->data[strlen(head_ptr_dounn->data)]
        ='\0';
    }
}
}
else
{
    upt_rec_ptr_dounn=head_ptr_dounn;
    for (loop_dounn=2; loop_dounn<update_network_node_no_dounn; loop_dounn++)
    {
        upt_rec_ptr_dounn=upt_rec_ptr_dounn->next;
    }
    if (proc_id_dounn == 1)
    {
        if (upt_rec_ptr_dounn->next == tail_ptr_dounn)
        {
            upt_rec_ptr_dounn->next == NULL;
            tail_ptr_dounn=upt_rec_ptr_dounn;
        }
        else
        {
            tmp_rec_ptr_dounn=(upt_rec_ptr_dounn->next);
            upt_rec_ptr_dounn->next=(tmp_rec_ptr_dounn->next);
        }
    }
    else
    {
        if (proc_id_dounn == 2 && update_rec_flag_dounn == 1)
        {
            (upt_rec_ptr_dounn->next)->data[0]='\0';
            strcpy((upt_rec_ptr_dounn->next)->data,update_network_node_dounn);
            (upt_rec_ptr_dounn->next)->data
            [strlen((upt_rec_ptr_dounn->next)->data)]='\0';
        }
    }
}
}

```

```

update_network_profile_dounn=fopen(network_profile_name_dounn,"w");
write_ctr_dounn=1;
while (head_ptr_dounn != tail_ptr_dounn)
{
    if (write_ctr_dounn == update_network_node_no_dounn && proc_id_dounn == 2)
    {
        fprintf(update_network_profile_dounn,"%s\n",head_ptr_dounn->data);
    }
    else
    {
        fprintf(update_network_profile_dounn,"%s",head_ptr_dounn->data);
    }
    head_ptr_dounn=head_ptr_dounn->next;
    write_ctr_dounn++;
}
fprintf(update_network_profile_dounn,"%s",tail_ptr_dounn->data);
fclose(update_network_profile_dounn);
}
else
{
    if (update_network_node_no_dounn == network_node_ctr_dounn)
    {
        printf("\n!!! Exit, no network node be selected !!!\n");
    }
}
}
free(head_ptr_dounn);
free(tail_ptr_dounn);
free(upt_rec_ptr_dounn);
free(new_rec_ptr_dounn);
if (proc_id_dounn == 1)
{
    printf("\n*** Do you want to delete another ");
    printf("network node (Y/N) ");
}
else
{
    if (proc_id_dounn == 2)
    {
        printf("\n*** Do you want to update another network node (Y/N) ");
    }
}
continue_flag_dounn=confirm_continue();
}

```


)

APPENDIX D - SOURCE CODE OF FILE TRANSFER SIMULATION SERVER PROGRAM USING UNIX DOMAIN STREAM PROTOCOL(TCP)

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include </usr/sys/netinet/in.h>
#include <arpa/inet.h>
#include <sys/time.h>

#define SERV_TCP_PORT 7003
#define MAXMSG 1024

char *pname;

/* Example of server using TCP protocol */

main(argc, argv)
int argc;
char *argv[];

{
    int sockfd, newsockfd, clien, childpid;
    struct sockaddr_in serv_addr, cli_addr;

    pname = argv[0];

    /* Open a TCP socket (an Internet stream socket). */
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("server: can't open datagram socket \n");
    }

    /* Bind our local address so that the client can send to us */
    bzero((char *) &serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_addr.sin_port = htons(SERV_TCP_PORT);
```

```

if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
{
    printf("server : can't bind local address\n");
}
listen(sockfd, 5);

for ( ; ; )
{
    /* Wait for a connection from a client process. */
    newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
    if (newsockfd < 0)
    {
        printf("server: accept error");
    }
    str_echo(newsockfd);
    close(newsockfd);
}

/* This procedure will receive FTP data from client and send
   a response message to client. During file transferring, it
   will compute the FTP time data. Finishing file transferring,
   it will send these information to the client */
str_echo(sockfd)

int sockfd;

{
    int n, clilen;
    long file_l, file_l_r;
    char mesg[MAXMESG+1];
    FILE *out_file;
    char file_len[34], file_name[45];
    struct timeval start_time_v, end_time_v, ftp_with_io_v, ftp_without_io_v;
    struct timeval disk_io_time_v, t_flag1_v, t_flag2_v, t_flag3_v, t_flag4_v;
    struct timezone start_time_z, end_time_z, ftp_with_io_z, ftp_without_io_z;
    struct timezone disk_io_time_z, t_flag1_z, t_flag2_z, t_flag3_z, t_flag4_z;

```

```

char ftp_with_io_sec[34],ftp_with_io_usec[34];
char ftp_without_io_sec[34],ftp_without_io_usec[34];
char disk_io_time_sec[34],disk_io_time_usec[34];
char time_hour[34];
long start_time,end_time;
int recno;

```

```

recno=0;
n=0;
file_l=0;
file_l_r=0;
file_len[0]='\0';
ftp_with_io_v.tv_sec=0;
ftp_with_io_v.tv_usec=0;
ftp_without_io_v.tv_sec=0;
ftp_without_io_v.tv_usec=0;
disk_io_time_v.tv_sec=0;
disk_io_time_v.tv_usec=0;
while (file_l < 1)
{
    n=read(sockfd, file_len, 34);
    if (n < 0)
    {
        printf("str_echo : file length read error\n");
    }
    else if (write(sockfd, file_len, n) != n)
    {
        printf("str_echo : file length write error\n");
    }
    file_l=atol(file_len);
}

```

```

/* read the file name from client */
n=read(sockfd, file_name, 45);
if (n < 0)
{
    printf("str_echo : file name read error\n");
}
else
{
    if (write(sockfd, file_name, n) != n)
    {
        printf("str_echo : file name write error\n");
    }
}

```

```

    }
    file_name[n] = '\0';
    if ((out_file = fopen(file_name,"w")) == NULL)
    {
        printf("Can not open the %s file \n",file_name);
    }
    else
    {
        time(&start_time);
        printf("Connect time is : %ld %s \n",start_time,ctime(&start_time));
        gettimeofday(&start_time_v,&start_time_z);
        printf("Start time : sec %ld, usec %ld \n",
            start_time_v.tv_sec,start_time_v.tv_usec);
        while (file_l > 0)
        {
            gettimeofday(&t_flag1_v,&t_flag1_z);
            n=read(sockfd, mesg, MAXMSG);
            if (n <= 0)
            {
                printf("str_echo : read error \n");
            }
            else
            {
                {
                    mesg[n] = '\0';
                    gettimeofday(&t_flag2_v,&t_flag2_z);
                    fwrite(mesg, sizeof(char), n, out_file);
                    gettimeofday(&t_flag3_v,&t_flag3_z);
                }
                if (write(sockfd, mesg, n) != n)
                {
                    printf("str_echo : write error \n");
                }
                gettimeofday(&t_flag4_v,&t_flag4_z);
                file_l=file_l-n;
                file_l_r=file_l_r+n;
                tvsub(&t_flag4_v,&t_flag3_v);
                tvsub(&t_flag3_v,&t_flag2_v);
                tvsub(&t_flag2_v,&t_flag1_v);
                tvadd(&disk_io_time_v,&t_flag3_v);
                tvadd(&ftp_with_io_v,&t_flag2_v);
                tvadd(&ftp_with_io_v,&t_flag3_v);
                tvadd(&ftp_with_io_v,&t_flag4_v);
                tvadd(&ftp_without_io_v,&t_flag2_v);
            }
        }
    }
}

```

```

    tvadd(&ftp_without_io_v,&t_flag4_v);
}
gettimeofday(&end_time_v,&end_time_z);
printf("End time : sec %ld, usec %ld\n",
    end_time_v.tv_sec,end_time_v.tv_usec);
fclose(out_file);
time(&end_time);
printf("Disconnect time is : %ld %s\n",end_time,ctime(&end_time));
sprintf(time_hour,"%ld",end_time);
sprintf(ftp_with_io_sec,"%ld",ftp_with_io_v.tv_sec);
sprintf(ftp_with_io_usec,"%ld",ftp_with_io_v.tv_usec);
sprintf(ftp_without_io_sec,"%ld",ftp_without_io_v.tv_sec);
sprintf(ftp_without_io_usec,"%ld",ftp_without_io_v.tv_usec);
sprintf(disk_io_time_sec,"%ld",disk_io_time_v.tv_sec);
sprintf(disk_io_time_usec,"%ld",disk_io_time_v.tv_usec);
sprintf(file_len,"%ld",file_l_r);
printf("File transfer time with disk I/O is %s sec %s mirco sec\n",
    ftp_with_io_sec,ftp_with_io_usec);
printf("File transfer time without disk I/O is %s sec %s mirco sec\n",
    ftp_without_io_sec,ftp_without_io_usec);
printf("Disk I/O time is %s sec %s mirco sec\n",
    disk_io_time_sec,disk_io_time_usec);
n=strlen(time_hour);
if (write(sockfd, time_hour, n) != n)
{
    printf("str_echo : time_hour write error\n");
}
else if (read(sockfd, time_hour, n) != n)
{
    printf("str_echo : time_hour read error\n");
}
n=strlen(ftp_with_io_sec);
if (write(sockfd, ftp_with_io_sec, n) != n)
{
    printf("str_echo : ftp_with_io_sec write error\n");
}
else if (read(sockfd, ftp_with_io_sec, n) != n)
{
    printf("str_echo : ftp_with_io_sec read error\n");
}
n=strlen(ftp_with_io_usec);
if (write(sockfd, ftp_with_io_usec, n) != n)
{

```

```

        printf("str_echo : ftp_with_io_usec sendto error\n");
    }
    else if (read(sockfd, ftp_with_io_usec, n) != n)
    {
        printf("str_echo : ftp_with_io_usec read error\n");
    }
    n=strlen(ftp_without_io_sec);
    if (write(sockfd, ftp_without_io_sec, n) != n)
    {
        printf("str_echo : ftp_without_io_sec write error\n");
    }
    else if (read(sockfd, ftp_without_io_sec, n) != n)
    {
        printf("str_echo : ftp_without_io_sec read error\n");
    }
    n=strlen(ftp_without_io_usec);
    if (write(sockfd, ftp_without_io_usec, n) != n)
    {
        printf("str_echo : ftp_without_io_usec sendto error\n");
    }
    else if (read(sockfd, ftp_without_io_usec, n) != n)
    {
        printf("str_echo : ftp_without_io_usec read error\n");
    }
    n=strlen(file_len);
    if (write(sockfd, file_len, n) != n)
    {
        printf("str_echo : file_len sendto error\n");
    }
    else if (read(sockfd, file_len, n) != n)
    {
        printf("str_echo : file_len read error\n");
    }
}
}
}

```

```

/* Subtract 2 timeval structs: out=out-in.
   Out is assumed to be >= in.      */

```

```

tvsub(out,in)

```

```
register struct timeval *out,*in;
```

```
{  
  if ((out->tv_usec -= in->tv_usec) < 0)  
  {  
    out->tv_sec--;  
    out->tv_usec += 1000000;  
  }  
  out->tv_sec -= in->tv_sec;  
}
```

```
/* Add 2 timeval structs: out=out+in. */
```

```
tvadd(out,in)
```

```
register struct timeval *out,*in;
```

```
{  
  if ((out->tv_usec += in->tv_usec) >= 1000000)  
  {  
    out->tv_sec++;  
    out->tv_usec -= 1000000;  
  }  
  out->tv_sec += in->tv_sec;  
}
```


APPENDIX E - SOURCE CODE OF FILE TRANSFER SIMULATION SERVER PROGRAM USING UNIX DOMAIN DATAGRAM PROTOCOL(UDP)

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include </usr/sys/netinet/in.h>
#include <arpa/inet.h>
#include <sys/time.h>

#define SERV_UDP_PORT 7102
#define MAXMSG 4096

char *pname;

/* Example of server using UDP protocol */

main(argc, argv)
int argc;
char *argv[];

{
    int sockfd;
    struct sockaddr_in serv_addr, cli_addr;

    pname = argv[0];

    /* Open a UDP socket (an Internet datagram socket). */

    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
    {
        printf("server: can't open datagram socket \n");
    }
    else
    {

        /* Bind our local address so that the client
           can send to us */
        bzero((char *) &serv_addr, sizeof(serv_addr));
        serv_addr.sin_family = AF_INET;
```

```

serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
serv_addr.sin_port = htons(SERV_UDP_PORT);
if (bind(sockfd, (struct sockaddr *) &serv_addr,
    sizeof(serv_addr)) < 0)
{
    printf("server : can't bind local address \n");
}
else
{
    printf("finish UDP socket open \n");
    for ( ; )
    {
        dg_echo(sockfd, (struct sockaddr *) &cli_addr,
            sizeof(cli_addr));
    }
}

/* dg_echo */

```

```

dg_echo(sockfd, pcli_addr, maxclilen)

```

```

int      sockfd;
struct sockaddr *pcli_addr; /* ptr to appropriate sockaddr_XX structure */
int      maxclilen; /* sizeof(*pcli_addr) */

{
    int n, clilen;
    long file_l, file_l_r;
    char mesg[MAXMSG];
    FILE *out_file;
    char file_len[34], file_name[45];
    struct timeval start_time_v, end_time_v, ftp_with_io_v, ftp_without_io_v;
    struct timeval disk_io_time_v, t_flag1_v, t_flag2_v, t_flag3_v, t_flag4_v;
    struct timezone start_time_z, end_time_z, ftp_with_io_z, ftp_without_io_z;
    struct timezone disk_io_time_z, t_flag1_z, t_flag2_z, t_flag3_z, t_flag4_z;
    char time_hour[34];
    char ftp_with_io_sec[34], ftp_with_io_usec[34];
    char ftp_without_io_sec[34], ftp_without_io_usec[34];
}

```

```

char disk_io_time_sec[34],disk_io_time_usec[34];
long start_time,end_time;

n=0;
file_l=0;
file_l_r=0;
file_len[0]='\0';
file_name[0]='\0';
ftp_with_io_v.tv_sec=0;
ftp_with_io_v.tv_usec=0;
ftp_without_io_v.tv_sec=0;
ftp_without_io_v.tv_usec=0;
disk_io_time_v.tv_sec=0;
disk_io_time_v.tv_usec=0;

clilen=maxclilen;
while (file_l < 1)
{
    /* receive the file length from client */
    n = recvfrom(sockfd, file_len, 34, 0, pcli_addr, &clilen);
    if (n < 0)
    {
        printf("dg_echo : file length recefrom error\n");
    }
    else
    {
        if (sendto(sockfd, file_len, n, 0,
            pcli_addr, clilen) != n)
        {
            printf("dg_echo : file_len sendto error\n");
        }
        file_len[n]='\0';
        file_l=atol(file_len);
        file_len[0]='\0';
    }
}

/* receive the file name from client */
clilen=maxclilen;
n = recvfrom(sockfd, file_name, 45, 0, pcli_addr, &clilen);
if (n < 0)
{
    printf("dg_echo : file name recvfrom error\n");
}

```

```

    }
else
{
    if (sendto(sockfd, file_name, n, 0,
        pcli_addr, clilen) != n)
    {
        printf("dg_echo : file_name sendto error\n");
    }
    file_name[n] = '\0';
    if ((out_file = fopen(file_name, "w")) == NULL)
    {
        printf("Can not open the %s file\n", file_name);
    }
    else
    {
        time(&start_time);
        printf("Connect time is : %ld %s\n",
            start_time, ctime(&start_time));
        gettimeofday(&start_time_v, &start_time_z);
        printf("Start time : sec : %ld, usec : %ld\n",
            start_time_v.tv_sec, start_time_v.tv_usec);
        while (file_l > 0)
        {
            mesg[0] = '\0';
            clilen = maxclilen;
            gettimeofday(&t_flag1_v, &t_flag1_z);
            n = recvfrom(sockfd, mesg, MAXMSG, 0,
                pcli_addr, &clilen);
            if (n <= 0)
            {
                printf("dg_echo : recvfrom error\n");
            }
            else
            {
                mesg[n] = '\0';
                gettimeofday(&t_flag2_v, &t_flag2_z);
                fwrite(mesg, sizeof(char), n, out_file);
                gettimeofday(&t_flag3_v, &t_flag3_z);
                if (sendto(sockfd, mesg, n, 0,
                    pcli_addr, clilen) != n)
                {
                    printf("dg_echo : sendto error\n");
                }
            }
        }
    }
}

```

```

        gettimeofday(&t_flag4_v,&t_flag4_z);
        file_l=file_l-n;
        file_l_r=file_l_r+n;
        tvsub(&t_flag4_v,&t_flag3_v);
        tvsub(&t_flag3_v,&t_flag2_v);
        tvsub(&t_flag2_v,&t_flag1_v);
        tvadd(&disk_io_time_v,&t_flag3_v);
        tvadd(&ftp_with_io_v,&t_flag2_v);
        tvadd(&ftp_with_io_v,&t_flag3_v);
        tvadd(&ftp_with_io_v,&t_flag4_v);
        tvadd(&ftp_without_io_v,&t_flag2_v);
        tvadd(&ftp_without_io_v,&t_flag4_v);
    }
}

gettimeofday(&end_time_v,&end_time_z);
printf("End time : sec %ld, usec %ld\n",end_time_v.tv_sec,end_time_v.tv_usec);
fclose(out_file);
time(&end_time);
printf("Disconnect time is : %ld %s\n",end_time,ctime(&end_time));
sprintf(time_hour,"%ld",end_time);
sprintf(ftp_with_io_sec,"%ld",ftp_with_io_v.tv_sec);
sprintf(ftp_with_io_usec,"%ld",ftp_with_io_v.tv_usec);
sprintf(ftp_without_io_sec,"%ld",ftp_without_io_v.tv_sec);
sprintf(ftp_without_io_usec,"%ld",ftp_without_io_v.tv_usec);
sprintf(disk_io_time_sec,"%ld",disk_io_time_v.tv_sec);
sprintf(disk_io_time_usec,"%ld",disk_io_time_v.tv_usec);
sprintf(file_len,"%ld",file_l_r);
printf("File transfer time with disk I/O is ");
printf("%s sec %s micro sec\n",ftp_with_io_sec,ftp_with_io_usec);
printf("File transfer time without disk I/O is ");
printf("%s sec %s micro sec\n",ftp_without_io_sec,ftp_without_io_usec);
printf("Disk I/O time is %s sec %s micro sec\n",disk_io_time_sec,disk_io_time_usec);
n=strlen(time_hour);
if (sendto(sockfd, time_hour, n, 0, cli_addr, clilen) != n)
{
    printf("dg_echo : time_hour sendto error\n");
}
else
{
    n = recvfrom(sockfd, time_hour, n, 0, pcli_addr, &clilen);
    if (n <= 0)
    {
        printf("dg_echo : time_hour recvfrom error\n");
    }
}

```

```

    }
}
n=strlen(ftp_with_io_sec);
if (sendto(sockfd, ftp_with_io_sec, n, 0, pcli_addr, clilen) != n)
{
    printf("dg_echo : ftp_with_io_sec sendto error\n");
}
else
{
    n = recvfrom(sockfd, ftp_with_io_sec, n, 0, pcli_addr, &clilen);
    if (n <= 0)
    {
        printf("dg_echo : ftp_with_io_sec recvfrom error\n");
    }
}
n=strlen(ftp_with_io_usec);
if (sendto(sockfd, ftp_with_io_usec, n, 0, pcli_addr, clilen) != n)
{
    printf("dg_echo : ftp_with_io_usec sendto error\n");
}
else
{
    n = recvfrom(sockfd, ftp_with_io_usec, n, 0, pcli_addr, &clilen);
    if (n <= 0)
    {
        printf("dg_echo : ftp_with_io_usec recvfrom error\n");
    }
}
n=strlen(ftp_without_io_sec);
if (sendto(sockfd, ftp_without_io_sec, n, 0, pcli_addr, clilen) != n)
{
    printf("dg_echo : ftp_without_io_sec sendto error\n");
}
else
{
    n = recvfrom(sockfd, ftp_without_io_sec, n, 0, pcli_addr, &clilen);
    if (n <= 0)
    {
        printf("dg_echo : ftp_without_io_sec recvfrom error\n");
    }
}
n=strlen(ftp_without_io_usec);
if (sendto(sockfd, ftp_without_io_usec, n, 0, pcli_addr, clilen) != n)

```

```

    {
        printf("dg_echo : ftp_without_io_usec sendto error\n");
    }
    else
    {
        n = recvfrom(sockfd, ftp_without_io_usec, n, 0, pcli_addr, &clilen);
        if (n <= 0)
        {
            printf("dg_echo : ftp_without_io_usec recvfrom error\n");
        }
    }
    n=strlen(file_len);
    if (sendto(sockfd, file_len, n, 0, pcli_addr, clilen) != n)
    {
        printf("dg_echo : file_len sendto error\n");
    }
    else
    {
        n = recvfrom(sockfd, file_len, n, 0, pcli_addr, &clilen);
        if (n <= 0)
        {
            printf("dg_echo : file_len recvfrom error\n");
        }
    }
}
}
}

```

/* Subtract 2 timeval structs: out=out-in.
 Out is assumed to be >= in. */

tvsub(out,in)

register struct timeval *out,*in;

```

{
    if ((out->tv_usec -= in->tv_usec) < 0)
    {
        out->tv_sec--;
        out->tv_usec += 1000000;
    }
}

```

```

    out->tv_sec -= in->tv_sec;
}

/* Add 2 timeval structs: out=out+in. */
tvadd(out,in)

register struct timeval *out,*in;

{
    if ((out->tv_usec += in->tv_usec) >= 1000000)
    {
        out->tv_sec++;
        out->tv_usec -= 1000000;
    }
    out->tv_sec += in->tv_sec;
}

```


LIST OF REFERENCES

- [Ref 1] Mike Loukides, The Whole Internet User's Guide & Catalog, CA, O'Reilly & Associates, Inc., 1992 September.
- [Ref 2] Stine, R.H.ed., FYI on a network management tool catalog: Tools for monitoring and debugging TCP/IP internets and interconnected devices., 1990 April.
- [Ref 3] Charles L. Hedrick., Introduction to the Internet protocol., Rutgers university, 1987.
- [Ref 4] Charles L. Hedrick., Introduction to Administration of an Internet-based local network protocol., Rutgers university, 1988.
- [Ref 5] W. Richard Stevens., Unix network programming., NJ, Prentice Hall., 1990, pp. 258-341.
- [Ref 6] W. Richard Stevens., Unix network programming., NJ, Prentice Hall., 1990, pp. 261-262.
- [Ref 7] W. Richard Stevens., Unix network programming., NJ, Prentice Hall., 1990, pp. 445-464.
- [Ref 8] Douglas E. Comer. and David L. Stevens., Interneting with TCP/IP vol II : design,implementation,and Internals., Prentice Hall., 1991, pp. 123-142.

INITIAL DISTRIBUTION LIST

Defense Technical Information Center Cameron Station Alexandria, VA 22034-6145	2
Dudley Knox Library Code 52 Naval Postgraduate School Monterey, CA 93943	2
Chairman, Code CS Computer Science Department Naval Postgraduate School Monterey, CA 93943	1
Professor Chin-Hwa Lee, Code EC/Le Department of Electtical and Computer Engineering Naval Postgraduate School Monterey, CA 93943	1
Professor Man-Tak Shing, Code CS/Sh Computer Science Department Naval Postgraduate School Monterey, CA 93943	1
Professor Chyan Yang Institute of Information Management National Chiao Tung University HsiuChu, Taiwan, Republic of China	1
Computer Education Center National Defense Management College Chung Ho, Taipei, Taiwan, Republic of China	2
Library National Defense Management College Chung Ho, Taipei, Taiwan, Republic of China	1

Capt. Fu, Chen-Hua
National Defense Management College
Chung Ho, Taipei, Taiwan, Republic of China

4