19

SMT: A System Monitoring Tool for DCE

Brutch P. C., Karmarkar A. S., Gurijala A., Walzel K., Marti W., and Pooch U. W.
Dept. of Computer Science
Texas A&M University
College Station, Texas 77843–3112, USA.
Phone (409) 845–5534. Fax (409) 847–8578.
email: {paulb, anishk, anilg, willis, pooch}@cs.tamu.edu

Abstract

Understanding interactions among various components in a distributed system is very important for system administrators and application developers. The System Monitoring Tool (SMT)[1] provides the ability to view OSF/DCE traffic at multiple levels of detail, which makes this tool useful to application writers as well as people debugging protocol. SMT allows simultaneous monitoring of multiple network segments and provides protocol aware decoding of Remote Procedure Call (RPC) activity. SMT consists of two components: a remote monitoring agent to capture packets, and a host workstation software program which analyzes and groups the captured packets according to their RPC transaction. The RPC can be viewed at multiple levels of granularity, from raw packets to complete RPC transactions. The tool is an applied example of hierarchical specification of communication protocols.

Keywords

Network Monitoring, Distributed Computing Environment, Remote Procedure Call, Client Server Model, Remote Network Monitoring Agent.

1 INTRODUCTION

Advances in computer network technology have made the implementation of distributed systems both feasible and desirable. A distributed system is defined as, 'a combination of hardware, software, and network components in which the software components execute on two or more processors and communicate via a network' (Lockhart, 1994). Various standards have been developed to enhance the implementation of distributed systems. One of the most popular of these standards is the Open Software Foundation's (OSF) Distributed Computing Environment (DCE).

DCE can be described as, 'a set of software tools and services that make it much easier to develop and operate distributed computer applications' (Lockhart 1994). DCE is basically a set of integrated services, which enable developers to create distributed applications. DCE is based on the client/server model of computing, where by a client application requests service

1. This project was supported by IBM Advanced Workstation Division. Austin Texas under contract number 32525–42640–CS

A. Schill et al. (eds.), *Distributed Platforms*

from a remote server. The server application interacts with the DCE environment and provides services to the remote client.

The remote procedure call (RPC) is the most important facility used to implement the client/server model. The RPC provides the client with the ability to call a procedure on a remote server, as if the procedure were local to the client. In addition, RPC provides network transparency for portability and interoperability.

2 RATIONAL FOR DEVELOPING SMT

The interaction among DCE services and distributed applications is complex. Therefore in order to understand these interactions, DCE managers and application developers may need to study the network traffic. This can be accomplished by capturing packets on the network and then analyzing them. A number of tools have already been developed to provide this function, but they do not provide much help in analyzing DCE RPC.

These existing tools are limited to monitoring a single packet and do not show the relationship between various packets comprising an RPC. CITI at Michigan did some work in this regard (Howe, 1993). Their *Sniffer* captures DCE RPC packets and analyzes them. However, it has some drawbacks. One is that the DCE Interface Definition Language (IDL) needs to be modified in order to analyze packets by *Sniffer*. Other disadvantages are that *Sniffer* has to be run on each host wherever capturing and analyzing is required, and the relationship between packets is not displayed. These drawbacks are overcome by SMT.

The System Monitoring Tool design offers two key advantages: it allows for the simultaneous monitoring of multiple network segments, and provides protocol–aware decoding and display of RPC activity. The other tools do not show the relationship between packets and do not provide a context for packet decodes. The SMT allows RPC traffic to be viewed at different levels of granularity – starting from the transaction level, to the exchange primitive level, and finally to the packet level, where one could see the actual bytes in a packet. SMT requires no modification to any of the DCE components. Only one host workstation running SMT is sufficient. However, an Remote Network Monitoring (RMON) agent (see section 7 below) is required on each of the network segments which are to be monitored.

3 DEVELOPMENT GOALS OF THE SMT PROJECT

The goals and focus of the SMT project were as follows (SMT, 1994):

- Develop a 'standalone' software tool, independent of the running status of DCE on the network which provides network managers, system managers, and programmers with the ability to examine the state of RPC protocol transactions within a DCE cell network from a single workstation host located on any subnet.
- Develop a set of functions in the implementation of this tool which may be used as 'building blocks' for incorporation of SMT functionality into other tools and other specific network/system management schemes.

4 INTERPROCESS COMMUNICATION IN DCE

DCE provides a client/server computing model, in which servers provide services that are requested by clients. This model is implemented through RPC, in which a client invokes an RPC to request a service from a remote server. The DCE RPC runtime takes care of the underlying communication between the client and server. At runtime, the RPC needs to provide three basic

services: binding the client and server; providing the facilities to establish communication over the network; and transporting call data between the client and server (converting the data format if necessary).

The RPC runtime supports both connection–oriented – Transmission Control Protocol (TCP) and connectionless protocol – User Datagram Protocol (UDP). Communication between the client and server is performed by RPC protocol data units (PDUs) and each RPC involves the exchange of more than one PDU. Connection–oriented (CO) and connectionless (CL) protocols have different types of PDUs and their structure and encoding is described in Chapter 12 of the AES/DC –RPC. 'An RPC PDU contains up to three parts: a PDU header that contains protocol control information; a PDU body that contains data; and an authentication verifier that contains data specific to an authentication protocol' (AES/DC–RPC, 1993). For the purpose of the SMT project, we are only interested in the PDU header and not in the data information.

4.1 RPC Transaction Definition

The interaction between the client and server can be viewed at three different levels. We call the highest level a *transaction.* A *transaction* is defined as a single complete remote procedure call between a client and server by means of the RPC protocol. One remote procedure call transaction may result in the exchange of many PDUs between the client and server. All the RPC PDUs exchanged between the client and server during the interaction belong to that transaction. The lowest level is *packet* level. At this level, the user can see the packets transmitted between the client and server due to an RPC call. In connectionless mode, PDUs are single packets. While in connection–oriented mode, PDUs are part of a TCP data stream. Multiple transactions may occur during a single TCP connection. We treat each transaction as a separate instance. The RMON agent captures only those packets which have an RPC PDU header. Since each RPC transaction may result in more than one PDU, we have created an intermediate level by grouping certain types of PDUs. This level is called the *Exchange Primitive* (XP) level. Figure 1 shows an example of this three level hierarchy.

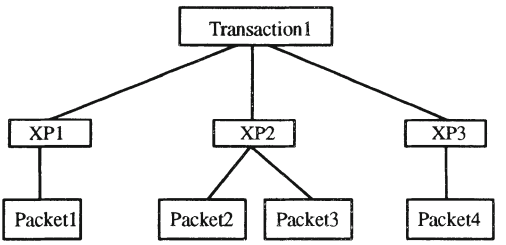

Figure 1 The Three Level Hierarchy for a Transaction.

4.2 Exchange Primitive Definition

Exchange Primitives are different phases of an RPC transaction during the client/server interaction. We have developed a set of Exchange Primitives, whose concept is borrowed from the AES DC–RPC service primitives. Some of the exchange primitives apply to both connectionless and connection–oriented mode such as: Invoke and Response. Others such as Association only apply to connection–oriented mode and Ping, which applies only to connectionless mode.

According to the (AES/DC –RPC, 1993), "the basic operations, which represent the interaction between the service user and the service provider, are specified as service primitives." The following is a list of the service primitives and their definitions (AES/DC –RPC, 1993):

- **Invoke** – used to invoke an RPC. This is a service user initiated service primitive.
- **Result** – used to return the input/output parameters at the end of a normal execution of the invoked RPC. This is a server–user initiated service primitive.
- **Cancel** – used to cancel an outstanding RPC.
- **Error** – used by the server manager routine to indicate an error in response to a previous Invoke indication.

The set of Exchange Primitives contains not only the above set of Service Primitives but also the following Exchange Primitives which are derived from the (AES/DC –RPC, 1993) definition of RPC Packet Types:

- **Ack** – indicates a client has received a response to an at–most–once request, and the server has been told to cease sending the response and discard the RESPONSE PDU.
- **Association** – contains either: a BIND and BIND_ACK, BIND and BIND_NAK, or ALTER_CTXT and ALTER_CTXT_RESP PDUs. It is associated with the actual setting up of a connection in a CO RPC transaction.
- **Association*** – contains either: a BIND and BIND_ACK, BIND and BIND_NAK, or ALTER_CTXT and ALTER_CTXT_RESP. It is associated with the second or more occurrence of an INVOKE and RESPONSE PDU for the CO RPC transaction.
- **Exception** – used in a CO RPC to indicate that the transaction contains either a FAULT, CANCEL, or ORPHAN PDU.
- **Ping** – used to indicate that a server has sent a WORKING PDU in reply to a PING PDU, and that the server is processing the client's call.
- **Response** – basically the same as the Result service primitive.
- **Shutdown** – used to indicate the termination of the connection, and the freeing of related resources for the RPC.
- **Unassigned_Fack** – used to indicate that a packet was captured, whose source can not be identified as either a client or server. This would indicate that a FACK packet was captured before any packet containing it's client/server information.

4.3 Decoding DCE RPC Packets

DCE nodes may exist on any of the common types of networks – token ring, Ethernet, FDDI, and so forth. This must be considered when decoding DCE packets, as well as the network and transport layer protocols the packet is following. An example RPC DCE packet on an Ethernet network is shown in Figure 2. SMT currently decodes DCE RPC packets using TCP and UDP with Internet Protocols (IP) over Ethernet (IEEE 802.3) and Token Ring (802.5).

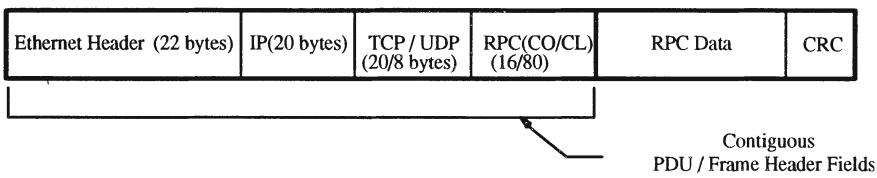

Figure 2 Protocol Headers of RPC Ethernet Packets.

4.4 Decoding Connectionless RPC Packets

Each RPC PDU consists of several fields which are given in AES/DC–RPC. A connectionless RPC PDU can be identified as one having the value of four in the RPC Version field. To decode a connectionless RPC packet, there are two fields used to identify the PDU as belonging to a particular RPC. These fields are the *Activity UUID* and the *Sequence Number.*

The header encoding for connectionless RPC PDUs is defined by the AES/DC–RPC as follows:

> The activity identifier is a Universal Unique Identifier (UUID) that uniquely identifies the client activity that is making a remote procedure call. The server can use the activity UUID as a communications key between it and the client. The sequence number is a 32–bit nonnegative integer which identifies the RPC that an activity is making. Each RPC invoked by an activity has a unique sequence number that is assigned when the call is initiated. All connectionless RPC PDUs sent on behalf of that particular call have the same sequence number, whether the PDUs are from the client to server or from the server to the client. When an activity initiates a new RPC, it increases the sequence number, so that each subsequent call has a larger sequence number. Together, the activity UUID and the sequence number uniquely identify an RPC. (AES/DC–RPC, 1993)

4.5 Decoding Connection–Oriented RPC Packets

A value of five in the RPC Version field identifies a PDU as a connection–oriented RPC PDU. Connection–oriented RPC packets do not have an Activity UUID to uniquely identify the PDU as belonging to a particular RPC. Therefore, a combination of source and destination host address, source and destination port number, and the PDU type are used to uniquely identify a PDU as belonging to an connection–oriented RPC.

5 SMT SYSTEM OVERVIEW

SMT consists of a software program running on a host workstation, and one or more RMONs agents connected to network segments containing DCE machines. These RMON agents must conform to the RMON Management Information (MIB) Internet standard, and may be either dedicated hardware, or programs running on workstations or PCs. For more information see Request for Comment (RFC) 1271, Remote Network Monitoring (RMON) MIB for Ethernet and RFC 1213, Management Information Base for Network Management of TCP/IP– Based Internets. The SMT software runs on the host workstation, and controls the RMONs via Simple Network Management Protocol (SNMP).

The SMT components in operation allow for the independent monitoring of RPC protocol transactions on the monitored network segments in order to verify DCE inter–machine communications and proper network software and hardware operations. Besides doing basic decoding of individual packets, SMT also identifies all packets related to a transaction and which exchange primitives (e.g., INVOKE, RESPONSE, CANCEL) have occurred. The SMT analysis process looks at the connection–oriented or connectionless RPC protocol header, which is encapsulated in TCP or UDP PDU's over IP.

SMT is designed to provide verification of proper network and DCE functioning, as well as to assist in software development and debugging. An example DCE network configuration consisting of DCE client and server machines, network segments, bridges, and the SMT components is shown in Figure 3.

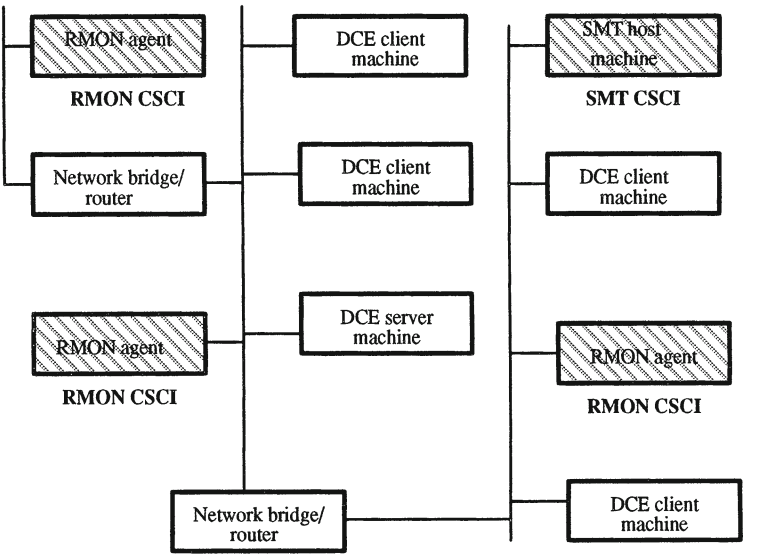

Figure 3 SMT Component Network Locations.

6 SMT COMPONENTS AND NAMING CONVENTION

The SMT is composed of two major components: the System Monitoring Tool software component and the RMON Agent hardware component. This paper follows the naming convention presented in (Roetzheim, 1991). Each such component is called a Computer System Configuration Item (CSCI). The SMT software component is further broken down into functional units called Computer Software Components (CSC). Each CSC consists of Computer Software Units (CSU), the smallest unit identified in (Roetzheim, 1991). A characteristic feature of a CSU is that it corresponds to a unit of compilation (in terms of software).

7 SMT RMON AGENT (CSCI 1)

The SMT RMON agent is a network–connected hardware component which supports MIB according to SNMP standards. The RMON agent may be a dedicated instrument, or a PC or workstation running appropriate software. Typically, one RMON agent is connected to each network segment. We have used two types of RMON agents – an Network Application Technology (NAT) Ethermeter and a PC running OS/2 RMON software. The OS/2 software used was Beholder – The Next Generation (BTNG), which is an RMON–compliant monitoring program. This OS/2 application turns a PC with a dedicated network card into a remote monitor for use with SMT. For more information on BTNG see (Otton, 1993).

The SMT CSCI software operating on the SMT host workstation controls the remote monitoring capabilities of the RMON agent(s). SMT sends SNMP messages to each RMON agent to collect and store RPC packets (or fragments of packets, particularly initial headers) and forward these packets (or packet fragments) to SMT upon request. This can be done over either the general network, or a dedicated communications link to SMT. Typically, the SMT software will poll the RMON agent(s) periodically to collect only those packets captured since the last poll.

The main functions of an RMON agent are to inform the SMT host workstation about the type of interface on which it is connected (such as Ethernet), and to capture and forward packets to the SMT host workstation. Only filter and packet capture groups are used for this purpose. The filter group of the RMON agent is set to capture only DCE packets. Basically, the agent captures only those packets which match all of the following fields:

1. The field identifying the IP protocol.
2. The field identifying the connection type (TCP or UDP)
3. The field identifying the DCE RPC.

8 SMT PROGRAM (CSCI 2)

The SMT program runs on a host workstation system as a user program under the control of the SMT Graphical User Interface (GUI). The SMT initializes as a single user program on the host workstation by a single command. No modification is needed to the host workstation operating system in order to support the SMT. The SMT program does not need to be run as the root user.

The functions of the SMT may be broken into four main areas. Specific CSC (Computer Software Components) implementing each general function are listed below:

1. Graphical User Interface (GUI) CSC: This CSC is the visible part of the SMT to the user. It provides a convenient interface to the user for accessing the components of SMT. It is also the main controller of the SMT modules and provides Help on SMT features.
2. RMON Manager CSC: This CSC provides communication to and from the remote monitoring agents (RMON CSCI). It also provides the functions to collect and log the stored packet header data from the RMON agents.
3. Packet Analysis CSC: This CSC analyzes the RPC packet headers captured using the RMON Manager. It also provides for display, printing, post analysis filtering, and gathers statistics on the analyzed transactions.
4. Utilities CSC: This CSC provides miscellaneous utilities supported by SMT. Some utilities provided are: file format conversion, UUID–to–string mapping,etc.

The above CSC's are independent modules with clearly defined interfaces. The GUI CSC controls the operation and interaction of the above CSC's to provide the overall functionality of the SMT to the user.

8.1 Graphical User Interface CSC

The GUI component provides the ability to run the SMT as a stand–alone program and centralizes user interface and display functions of the SMT. This component provides a graphical user interface based upon the X Window System, Version 11, Release 5, using the OSF MOTIF widget set and libraries and written according to OSF User Interface AES standards.

The GUI provides the menus and dialogue boxes which allow the user to control GUI functions, access SMT functions and utilities, and to communicate with the RMON Manager. The GUI also provides the display regions in which operational, error, and status messages of all SMT components are displayed. A help facility is also provided by the GUI for the user.

8.2 RMON Manager CSC

RMON Manager CSC is designed as an independent module. The RMON Manager CSC could be driven either by a graphical or text based user interface depending on the user's requirements. The RMON Manager can manage a pre–specified number of RMON Agents using concurrently

running threads which are created dynamically. The RMON Manager CSC manages the activities related to RMON Agents: configuring the agents, initiating the capturing of the packets, fetching captured packets, and stopping the capture of packets. The RMON Manager uses SNMP messages to communicate with the RMON agent(s).

The GUI CSC controls (triggers) all the activities of the RMON Manager. The interaction between GUI and RMON Manager is asynchronous. The RMON Manager allows for the simultaneous capture of RPC packets flowing in different network segments (using different RMON Agents). The capturing of packets can be done in different modes, i.e., one can specify the various attributes of the packets to be captured. The captured PDU headers and associated timestamp information are stored in respective log files for each RMON agent.

The RMON Manager CSC performs the following functions:

1. Receive commands from the Graphical User Interface (GUI) CSC to configure an RMON agent, and to start and stop the packet capture process for an RMON agent.
2. Periodically poll RMON agent(s) by using SNMP messages, when packet capture is active. Request and receive PDUs from the RMON agent(s).
3. Store captured RPC PDU headers and timestamps in a log file to allow for later analysis by the Packet Analysis CSC.

8.3 Packet Analysis CSC

The Packet Analysis CSC is comprised of five Computer Software Units (CSUs): Select Logs, Analysis, Transaction Display, Print, Post Analysis Filter, and Statistics. The Packet Analysis CSC provides the following support:

Allows the user to select multiple logs which are to be concurrently analyzed.

Analyzes the selected log file(s) by decoding the packets and logically grouping them such that the packets are a part of a transaction primitive and a transaction.

Provides an interactive screen window to display the transactions, exchange primitives, packets, and hex packet decodes of the analyzed log files.

Provides the ability to print the analyzed transactions in both Transaction Time Order (packets displayed in time order relative to their transaction) and Packet Time Order (relative time order according to timestamps).

Allows the user to filter out a subset of the transactions by specifying various filtering attributes.

8.3.1 Analysis CSU

The purpose of the analysis CSU is to logically group an arbitrary collection of RPC packets into transactions. A transaction is defined as a single complete procedure call between a client and server by means of the RPC protocol. One remote procedure call transaction may result in the exchange of many network packets between two systems. All the RPC PDUs exchanged between the client and server during that interaction belong to the transaction.

Both connection–oriented (CO) and connectionless (CL) RPC calls have well–defined RPC protocol states. Grouping the RPC PDUs according to the state of the RPC transaction allows for a higher level semantic view of ongoing transactions between two distributed programs in DCE.

A three–level tree is constructed by the Analysis CSU in order to collect packets in the hierarchy as shown in Figures 4 and 5. The first level of the tree holds the different transactions and the second level holds the exchange primitives that comprise the transaction. The set of exchange primitives consists of those phases that would be of interest for a network analyst trying to debug or study DCE traffic. The third level holds the actual PDUs which comprise the ex-

change primitives. Figure 4 shows a typical collection of packets belonging to a transaction in a connectionless RPC protocol. Figure 5 shows a typical collection of packets belonging to a transaction in connection–oriented RPC protocol.

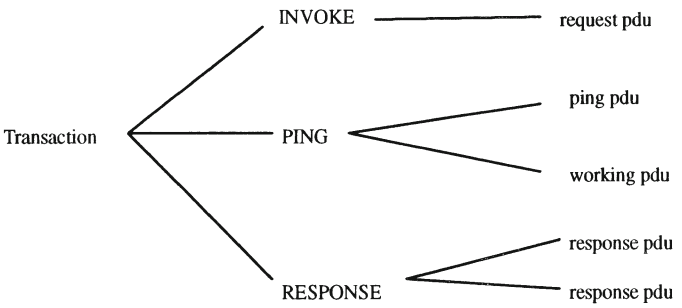

Figure 4 Example of a Transaction Hierarchy for CL.

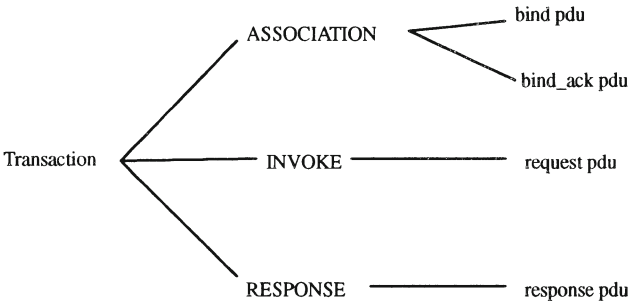

Figure 5 Example of a Transaction Hierarchy for CO.

8.3.1.1 Packet Synchronization

The Analysis CSU has to analyze packets together from multiple log files. In order to do this, packets from different log files have to be ordered with respect to time. This is accomplished by a packet synchronization function which examines the timestamps of packets sequentially within a single log file, but concurrently across files. Every time a packet is picked for analysis; the time stamps of the packets from each log file are considered, and the packet with the smallest time stamp is selected. The selected packet is analyzed and is then removed from the list of available packets for analysis. Th selection process continues until there are no more packets available for analysis.

8.3.2 Transaction Display CSU

The Transaction Display CSU provides four windows to display the analyzed transactions:

(1) Transactions window

(2) Exchange Primitives Window

(3) Packets Window

(4) Hexadecimal Packet Data Window

The user can view the information captured in the log files on four levels, each corresponding to one of these four windows. The highest level is the transaction level, and the lowest level is the hexadecimal packet data window. Figure 6 shows the transaction display window.

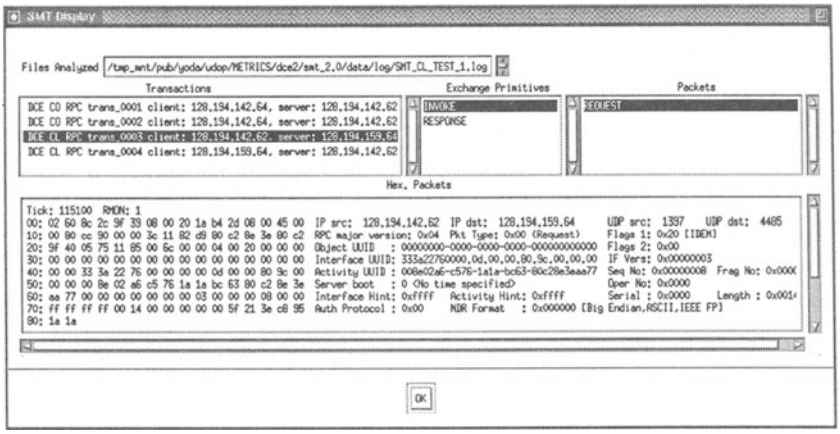

Figure 6 The Transaction Display Window.

8.3.3 Post Analysis Filter CSU

The post analysis filter allows the user to filter out a subset of the transactions by specifying various filtering attributes. The post analysis filter access menu contains functions which allow user to do the following: create a filter by entering the attribute values on which the transactions will be filtered, invoke the filter, and specify the mode of display (all transactions or only filtered transactions). Currently there is only one filter available which is a Service filter. This filter is DCE service–oriented because it is intended to search RPCs for a specified service

The filtering is performed using an expression comprised of a list of tuples OR'd together. Each tuple consists of several fields. The fields inside a tuple are logically AND'ed. The User Interface provides the functions necessary to get this information from the user in order to build the expression. Each tuple corresponds to one popup configuration window. In the popup window, all of the possible fields are listed. The user fills in the specific fields they want to filter on.

8.3.4 Statistics CSU

The Statistics CSU maintains the number of connectionless DCE RPC packets, connection–oriented DCE RPC packets, and non–DCE packets in the analyzed log files. For connectionless and connection–oriented DCE RPC, a tally is maintained for each PDU type.

The Statistical display, showing the number and type of packets in the log files is generated during the analysis phase. It is available for display immediately after the selected log files have been analyzed.

8.4 Utilities CSC

The Utilities CSC provides six utilities for the user. These utilities allow the user to edit the configuration file, edit the in–memory UUID–Map which is maintained in the configuration file, convert log files captured by other utilities such as Iptrace to the SMT format, browse a directory and view a selected log file, search the in–memory UUID–map for ASCII string names or UUIDs, and add UUID mappings to the configuration file from selected '.IDL' files.

The UUID Mapping feature allows the user to substitute names for the hexadecimal UUIDs, which are 32 digits. By doing so, the translation fields in the Transaction Display's hexadecimal packet window output is made more readable. For instance, the Object UUID could be read as `UUID_one` instead of the hexadecimal `0x0a1a3b2a39867123a67238c8b9b2312a`. This mapping from UUIDs to names is maintained in a linked list during program execution, but remains in the configuration file *.smtrc* when SMT is not running. The UUID mapping is read in from the configuration file when SMT is invoked.

8.4.1 UUID Mapping Utilities CSU

The UUID mapping feature of SMT allows the user to associate names with UUIDs. These mappings make it easier for the user to read the individual packet display. The user can enter an ASCII string and search the UUID mapping for a corresponding UUID, if it exists. Similarly, the user can enter a UUID and search the UUID map for a corresponding name. The UUID to Config utility in SMT, provides the user with the ability to automatically search .IDL files, and then add the UUID–Interface name mapping to the configuration file, as well as any function name–number mappings that may appear in the file. These mappings are stored in the *.smtrc* configuration file.

8.4.2 Format Conversion Utility CSU

The Format Conversion utility allows the Analysis CSC of SMT to analyze packets which are captured not only by SMT, but also by other utilities such as Iptrace on AIX and Etherfind on SunOS operating systems. This utility identifies and reads the files logged by these other packet capturing programs and converts them into a format which can be read and understood by the Analysis CSC. SMT currently supports Iptrace and Etherfind file conversion, but this can be extended to other formats.

9 CURRENT STATUS

The current version satisfies all of the basic design goals for SMT. It runs on the IBM RS/6000 and AIX 3.2 platform and Sparc SunOS 4.1.3. It is able to communicate with different types of RMON agents on different MAC protocols (Ethernet IEEE 802.3 and Token Ring IEEE 802.5) located on different network segments. It captures and analyzes both connection–oriented and connectionless RPC PDUs, and displays them in Transaction Time Order and Packet Time Order formats.

SMT provides for multiple RMON agent monitoring by using threads, one thread for each RMON agent that is monitored. All of the components of SMT are contained within one process, so communication can be done via global variables.

Currently, packets are ordered based on a timestamp for the log file and a time tick which is associated with each packet by the RMON agent. The log file's timestamp is determined by the host workstation's date and time at the start of the capture process. The packet's time tick is determined by the difference between the RMON agent's restart time and the number of time ticks which have expired when the packet was captured by the RMON agent. When an RMON

agent is configured in capture mode, it's time is restarted to zero. After the agent's restart time has been set to zero, a time tick is incremented every millisecond. Since the RMON agents are not time synchronized in the network, we can not order the captured packets in an "absolute" time ordering. We can, however, order the packets in a relative time ordering within a few seconds.

Since more than one RMON agent may capture the same packets as they are sent across multiple networks, we must resolve the problem of the creation of multiple transactions for a single remote procedure call. We have decided to make the RMON agent identifier a key for identifying transactions. So although the Analysis CSU may create duplicate transactions for a single RPC occurrence, each transaction can be uniquely identified by its RMON agent identifier. The Analysis CSU will not create duplicate transactions for a single RMON agent.

10 FUTURE WORK

The current version of SMT performs an analysis only after all of the packets have been captured. A more valuable capability would be to allow for the concurrent capture, analysis, and display in "real time". The current control structure was based on threads to allow this functionality to be added. The current version of SMT provides only an approximate time synchronization among multiple RMON data streams. In addition, we would like to implement several additional performance measurements, including end–to–end timing. Both require techniques to properly compare the times from separate systems and possibly different resolutions.

SMT is a practical application of on going work to allow hierarchical specification of communication protocols (from individual message formats to sequences of exchanged messages). As more is known about specifying higher level actions (e.g. sequences of RPCs), this functionality will be incorporated into SMT.

Acknowledgment: We would like to acknowledge the following individuals who made a significant contribution to the development of the System Monitoring Tool: Ganesh Jayadevan, Ramesh Narayan, Charlie Richardson, Craig Smith, Neal Krawetz, Vance Swaggerty, Ganesha Beedubail, Sridhar Muppidi, and Sunil Santha. The authors would like to express their appreciation to Mr. Scott Page at IBM Austin Texas, for his valuable comments throughout this project.

We also gratefully acknowledge the suggestions of the anonymous referees.

REFERENCE

AES/DC–RPC. (1993) AES/Distributed Computing–Remote Procedure Call, *Revision A – Review Draft*, Open Software Foundation, 11 Cambridge Center, Cambridge, MA 02142.

Howe, James. (1993) An Environment for 'Sniffing' DCE–RPC Traffic, *CITI Tech Report 93–4*, Center for Information Technology Integration, The University of Michigan.

Lockhart, H. (1994) *OSF DCE: Guide to Developing Distributed Applications*, McGraw–Hill Inc., New York.

Otton, H. A. (1993) *The Beholder Cookbook*, Delf University of Technology, Netherlands

Roetzheim, William R. (1991) *Developing Software to Government Standards*, Prentice Hall, Chapter 9.

SMT. (1994) *System Design Document for DCE SMT*, Texas A&M University.

BIOGRAPHY

Paul C. Brutch is pursuing his Master of Computer Science degree at Texas A&M University. Previously he served as the Director of Computer Operations at the Air Force Material Command's System Acquisition School. His research interests include distributed systems, security, and database systems.

Anish Karmarkar is currently pursuing his Ph.D. in Computer Science at Texas A&M University. He received his Masters in Electrical Engineering at Texas Tech University, Lubbock, Texas in 1993. His research interests are distributed systems, operating systems, fault tolerant communication support and replication in distributed systems.

Anil Gurijala is currently pursuing his Ph.D. in Computer Science at Texas A&M University. He received his Masters in Electrical Engineering at Indian Institute of Technology, Delhi, India in 1991. His research interests are fault tolerant distributed system.

Kyle Walzel is currently working as a software developer for Texas Instruments in the Systems Group. He received his Master of Computer Science degree from Texas A&M in 1995.

Willis F. Marti is the Director of Computing Services and a senior lecturer in the Computer Science Department at Texas A&M University. He is completing his research in distributed systems for his doctorate. Mr. Marti has significant operational experience in computer networks at TRW, Sytek and Martin Marietta. He earned the M.S. in Computer Engineering from Stanford University on an NSF Fellowship and the B.S. from the United States Military Academy.

Dr. Udo W. Pooch, P.E., received his Ph.D in Theoretical Physics from the University of Notre Dame in 1969 and is the E–Systems Professor of Computer Science at Texas A&M University. Dr. Pooch is a very active researcher, supervising projects in distributed systems, fault–tolerant distributed environments, network security, and network simulation.