PROMIS: A Reliable Real-Time Network Management Tool for Wide Area Networks

Eduardo Magaña, Javier Aracil, Jesús Villadangos
Dpto. Automática y Computación
Universidad Pública de Navarra
Campus Arrosadia s/n
31006 Pamplona, Spain
{eduardo.magana, javier.aracil, jesusv}@upna.es

Abstract

This paper presents the PROMIS tool, a new network management tool based on a distributed architecture of traffic probes. PROMIS allows authorized users to monitor traffic from any WAN segment using a WWW console or a Tcl/Tk graphic interface. In contrast to current network monitoring platforms based in SNMP (Simple Network Management Protocol) or RMON (Remote network MONitor), PROMIS is based on an optimized software architecture that uses a reliable transport protocol (TCP) to provide lossless transmision of monitoring information, thus allowing for real-time monitoring irrespective of network load.

1. Introduction

Nowadays, we are witnessing a huge demand for client-server applications that require a guaranteed Quality of Service (QOS) from the network. In order to provide such QOS it is necessary to monitor network traffic so that preventive actions can take place before network congestion is detected. This implies the use of network monitoring tools that allow network managers to have a global view of the whole network from a single, centralized location.

Such network monitoring tools should provide a good granularity in traffic measurements in order to obtain an accurate picture of the network in real-time. Other desirable features are alarm programming capabilities and automatic detection of network failures.

Furthermore, traffic capture capabilities are also convenient in order to analyze network performance during a period of time. A packet-level capture is adequate to perform trace-driven simulations, that prove useful for network dimensioning purposes. On the other hand, traffic filters for both real-time and non real-time measurements should be available in order to have a detailed view of protocols, applications or specific hosts.

Recent measurements of LAN and WAN traffic show that Internet traffic presents self-similar features [4, 8]. Namely, the observed traffic presents the scaling property: no matter the time scale that we consider the normalized packet counting process looks the same in a distributional sense. Traffic burstiness is present at any time scale in contrast to the Poisson process which tends to smooth out with increasing time scales.

Such high variability in network traffic makes SNMP/RMON management platforms vulnerable since both alarms and data can be lost in congestion periods. Such potential data loss is due to the unreliable nature of the transport protocol (UDP) used by SNMP/RMON and the polling scheme from manager to agent. Neither requests nor alarms are confirmed by SNMP, whose design criteria responds for non real-time purposes. Congestion periods are interesting from a network management perspective since they determine network performance. Therefore, accuracy in traffic monitoring is needed specially during network congestion stages. In a 100 Mbps network a burst congestion period of 1 second implies the loss of about 100 Mb of data. Recent studies [2] show that the HTML page mean size is around 50 KB, therefore 250 HTML pages may be lost. This is a quite simplistic example which illustrates that the more network capacity, the more resolution in traffic measurement is needed.

This paper presents the architecture and functions of a new network monitoring tool, PROMIS, that offers a true real time traffic parameters monitoring. PROMIS gives the typical functionalities of a SNMP/RMON

1089-6503/98 $10.00 © 1998 IEEE

management system, but a higher reliability is provided by means of using a reliable transport protocol and an optimized software architecture for traffic management.

The rest of this paper is structured as follows: section 2 presents the state of art in traffic monitoring tools and section 3 presents PROMIS architecture. Section 4 is devoted to an in-depth description of PROMIS probe and console. We evaluate PROMIS using stress tests in section 5. Finally, section 6 presents the conclusions that can be drawn from this study.

2. State of art

Current network monitoring tools can be grouped as follows:

- Protocol analyzers or sniffers: sniffers are normally implemented on a portable PC platform so that the network operator can monitor traffic on a given link. Sniffers provide good granularity and limited traffic capture capabilities but they do not give a global view of the network.
- Management tools based in RMON: such tools provide a centralized console and RMON probes from which traffic statistics are collected. We will show later that such platforms are not adequate for real-time traffic monitoring.

Protocol analyzers are hardware/software dedicated systems that allow traffic monitoring of network segments. However, the analysis scope is limited to a single collision domain. Thus, a global view of the network is lost. Furthermore, they do not allow for continuous network control nor for alarm generation or significant traffic captures. Therefore they are not suited for network management purposes but to solve isolated problems. Proactive network maintenance relies on the use of network management tools that provide an early warning of trouble by notifying managers of deviations from normal behaviour patterns. This objective cannot be achieved with the use of protocol analyzers.

The SNMP/RMON management platforms (*HP Openview, SunNet Manager, IBM Netview and Cabletron Spectrum*) provide a global view of all WAN segments. Either the network active elements (bridges, routers and switches) or dedicated hardware probes have SNMP (Simple Network Management Protocol) [1] and RMON (Remote network MONitor) [9] probes in charge of gathering traffic information.

The SNMP protocol is based on a polling scheme from manager to agent. In order to monitor a given network parameter the manager sets the adequate values in the agent control tables. On the other hand, the agent can deliver information to the manager by means of a trap. Even though MIB-II [5] parameters are too generic (for example, accumulative bytes or packets), the RMON standard defines a MIB that supports significant information regarding traffic management: segment statistics, historic data, alarms, filtering capabilities per host, traffic between machines, top traffic machines, events, filters and packets captures.

The SNMP/RMON standards present the following disadvantages regarding traffic monitoring:

- Polling is CPU intensive in the manager since it has to keep track of the different polling intervals for different parameters. Furthermore, in long round-trip delay environments polling is unpractical and limits real-time features.
- Since UDP is used as transport protocol, neither alarm transmissions from agent to manager nor requests from manager to agent are guaranteed.
- In highly loaded environments real time traffic monitoring is limited because manager requests and agent replies can be lost. Note that a congested situation is most interesting from a network management perspective. In such situation valuable management information can be lost.
- Remote monitoring is not possible through the Internet since a request/reply scheme with unreliable transport is unrealistic.
- Capture size is normally limited by probe or network active element RAM memory.

Such disadvantages have been studied previously in the literature [6, 7]. PROMIS tries to overcome the above mentioned limitations by using a reliable transport protocol and an optimized software architecture to minimize information loss. Thus, it enhances SNMP/RMON functionality by adding reliability and real-time monitoring capabilities.

3. PROMIS Architecture

PROMIS components are console and probe. Several consoles can be active simultaneously on different machines. Probes are placed in each network segment to be monitored and they serve real-time traffic information as a WWW server would do. Therefore they provide a truly distributed environment so that traffic information can be accesed reliably from any host in

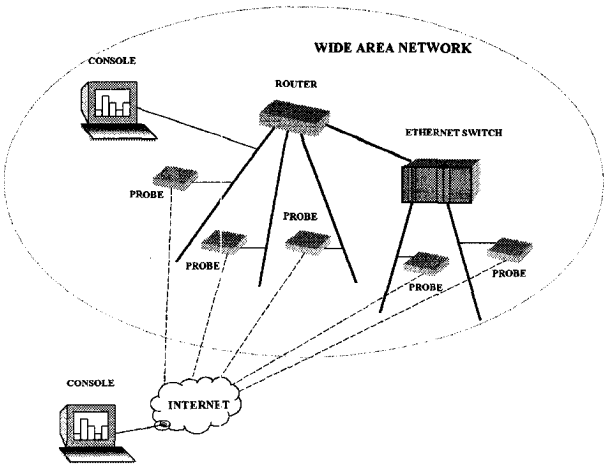

Figure 1. Network management scenario using PROMIS

the Internet. Figure 1 presents a network management scenario using PROMIS.

Traffic collection and processing is performed in the probe in order to guarantee that a minimum amount of information traffic to the console is transmitted through the network. PROMIS incorporates the following features:

- A variety of network performance parameters are monitored in real-time, ranging from global statistics (bytes/sec, packets/sec and mean utilization) to statistics filtered by machine, protocol, service and application.
- Alarms can be generated from any probe to the console.
- An expert system for automatic detection of network failures is provided, based on ICMP and a set of rules programmed by the network manager.
- Traffic capture capabilities are also provided.
- PROMIS probe incorporates an SNMP manager for communication with network active elements.
- PROMIS learns network topology in a continuous fashion through consultation of router tables using SNMP.

PROMIS probes use SNMP to access router tables in order to obtain the IP and MAC address of the segment hosts. A database is kept with all the necessary information to send real-time statistics and perform traffic captures on demand. PROMIS options can be outlined as follows:

- *Monitor mode*: In monitor mode the console receives traffic parameters in real-time and problem symptoms detected by the expert system. Such traffic parameters include bytes per second, packets per second and percentage over total observed traffic of the following services: FTP, Telnet, SMTP, WWW, POP2, POP3 and TALK. Futhermore, the same statistics are available for the following protocols: TCP, UDP, ICMP, ARP, AppleTalk(2 versions), IPX and NetBios. Any other protocol on top of IP can also be monitored.
- *Capture mode*: This mode allows to program a capture of the required traffic parameter specifying the desired time interval. Capture data is stored in probe hard disk (1.5 GB) so that the console is not forced to perform polling as in SNMP/RMON systems.
- *Alarms*: Threshold-based alarms can be programmed in each probe. Available alarms are global network utilization exceeding a threshold, host network usage percentage exceeding a threshold and broadcast storms detection.

Real-time parameters, traffic captures and alarms may have global scope (WAN segment) or filters. Predefined filters allow for a detailed monitoring of a specific service, application or host. Filters are listed in table 1.

machine	IP or MAC address
protocol	TCP, UDP, ICMP, ARP AppleTalk, IPX, NetBios
service	FTP, Telnet, SMTP, WWW POP2, POP3, TALK

Table 1. PROMIS filters

On the other hand, filters can be programmed manually using the IP protocol number or the TCP/UDP port.

4. PROMIS components: probe and console

PROMIS probes should have good storage capabilities and processing power while keeping cost at a minimum. We use PCs as probes since they also provide a great flexibility for data capture. Such PCs run PROMIS software concurrently with other applications, so that a dedicated use of the PC as a PROMIS probe is not necessary.

In order to achieve greater flexibility we design multiplatform consoles, namely WWW or Tcl/Tk-based consoles. We will show that a PC suffices to provide a traffic capture with minimum loss, thus providing a cost-effective solution for probes. As far as the console is concerned we do not tight such functionality to a particular workstation so that any PC or workstation may serve as PROMIS console at any time. On the other hand, more than one PROMIS console can be active simultaneously.

Java and Tcl/Tk make intensive use of CPU resources and give worse performance in comparison to compiled languages, such as C. We compare Java to Tcl/Tk for traffic monitoring applications in the following subsections.

4.1. Probe

Probes are implemented on a PC Pentium 133 MHz, 32MB RAM, 1.5GB of hard disk and Ethernet board 3COM 3c509 ISA. The operating system is Linux Slackware 3.2. Since real-time monitoring is pursued we need to follow a multiprocess architecture for probe software. The traffic collection program is composed by 3 processes that run concurrently. Such processes exchange messages using InterProcess Communication (IPC) mechanisms. A block diagram of the program is presented in figure 2.

Figure 2. Probe block diagram

The reader process is in charge of capturing all MAC packets from the Ethernet network along with generating a timestamp for each of them. To do so, the Ethernet board is configured in promiscuous mode.

The filter process performs packet identification, using source and destination MAC address, packet size, protocol number, source and destination IP address and application. It also processes the alarms and captures.

Since different real-time measurement request can be issued to a particular probe, the request manager process forks a process per request. Therefore, several requests for traffic captures or real-time monitoring parameters can be attended concurrently, as seen in figure 3.

A key issue in network monitoring equipment is IPC. Note that the reader process is sending traffic in real-time to the filter process. Therefore, a poor throughput in the IPC mechanism implies possible packet loss. IPC mechanisms can be grouped as follows, considering consumer and producer processes:

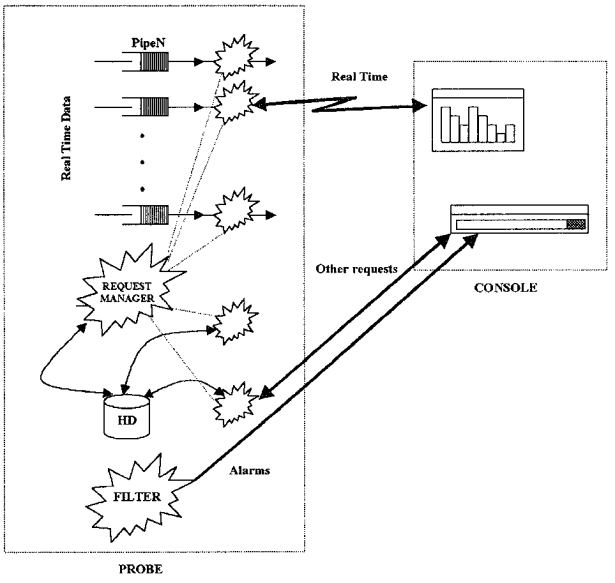

Figure 3. Filter process

- *Message queues*: Consumer and producer run in separate address spaces so that message passing is performed through kernel memory. This implies several copies that make IPC performance drop.
- *Shared memory*: Both process reader and filter run in the same address space so that they may exchange information in a very efficient manner. However, some means of synchronization need to be used. The operating system provides semaphores to do so. A system call to a semaphore means processing delay, so that there is a trade-off between synchronization delay and copies in kernel memory.

Message queues, such as System V pipes, are easier to use since they rely on the operating system kernel to perform message passing. Shared memory implies incorporating concurrency control to user software. Deadlock situations may arise if a careful design is not undertaken. However, there is an additional delay due to copies in kernel space. We evaluate message throughput through kernel space for the probe PC using two different versions of the Linux operating system: Slackware 3.2 and Red Hat 4.1. Results are shown in figure 4.

We observe from figure 4 that the sustained throughput is around 400 Mbps using System V pipes. The

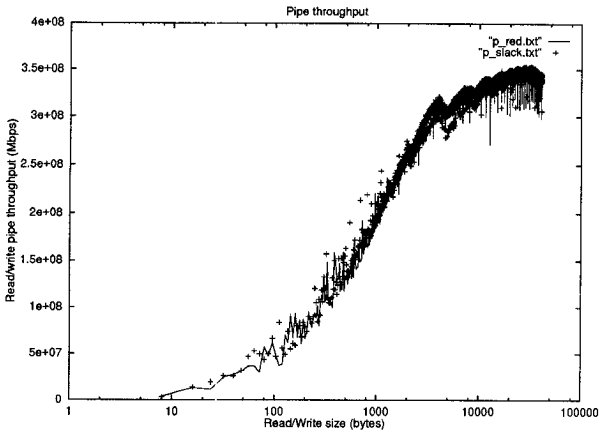

Figure 4. IPC performance

important conclusion is that IPC mechanisms based on message queues can be used for network monitoring purposes in 100 Mbps environments using simple hardware such as PCs. The use of System V pipes simplifies software design while it provides a high reliability in process synchronization, therefore we adopt such IPC mechanism for our design.

4.2. Console

We evaluate two different options regarding console design: WWW (Java applets) and Tcl/Tk. Both of them allow for a multiplatform design at the expense of a higher processing cost.

4.2.1 WWW (Java applet) console

The monitoring information resides in the probe web pages that can be either static or dynamic generated by CGIs [3]. Real-time monitoring presents a challenge since traffic display graphics need to be recalculated and redrawn in real-time. This is CPU consuming and may lead to overload of console resources.

Communication between probe and console is always performed using TCP/IP and HTTP between server and client, thus providing connection reliability. On the other hand sockets (SOCKSTREAM) between console Java applets and probe are used.

The use of WWW technologies has the advantage of information accessibility. Any host in the Internet with a WWW browser with Java support is ready to access monitoring information, which is served by the probes.

On the other hand, Java applets present several disadvantages in order to notify alarms to the console. A Java applet needs to be in continuous execution in the console side in order to attend possible alarm triggers from the probe. A preferred solution is to store alarms in the probe side and serve them on demand. However, this limitation can be overcame using a Tcl/Tk console, that we describe in the following section.

4.2.2 Tcl/Tk console

Tcl/Tk allows to program and ad-hoc console for PROMIS while keeping the multiplatform objective. It is portable to a number of platforms such as UNIX systems, Linux, Windows95, WindowsNT, OS/2 and Macintosh. On the other hand it offers more flexibility as far as statistics storage in the console. Java does not permit any access to hard disk due to security reasons. Moreover, alarms can be transmitted inmediately from probe to console.

The application is composed by a principal console window, in which the monitored network appears as a sensitive map with the location of each probe. Figure 5 presents the interface aspect.

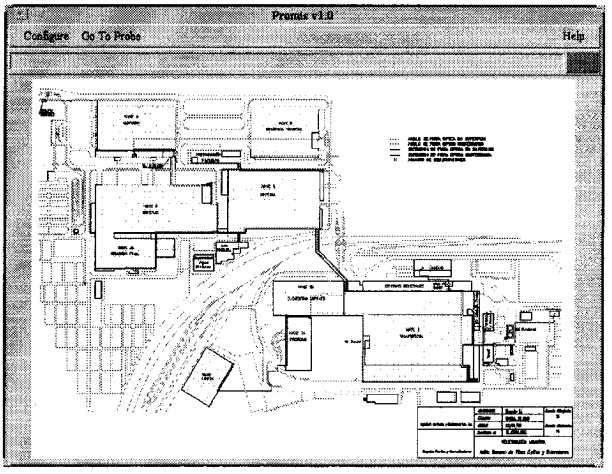

Figure 5. Tcl/Tk interface

Figure 6 presents a real time monitoring graphic of network global bytes per second. Note that all traffic peaks have been detected with zero loss, since TCP is used for transmission. Therefore we get an accurate picture of network activity.

As stated before, the main advantage of Tcl/Tk is that the console can run in autonomous manner and it provides client and server functionality. An alarm server is running continuously in the console side so that probes can open connections (SOCKSTREAM) and send alarm information. This is clearly impossible in WWW-based implementations, unless a Java-applet

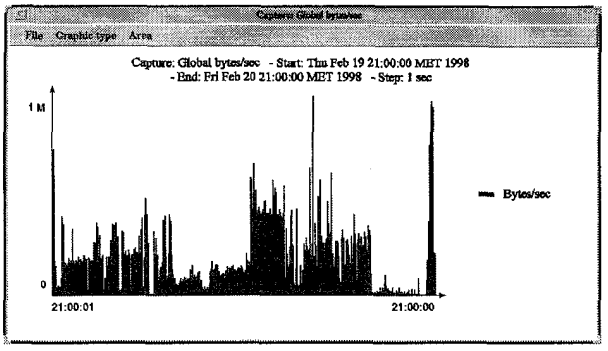

Figure 6. Real-time monitoring example

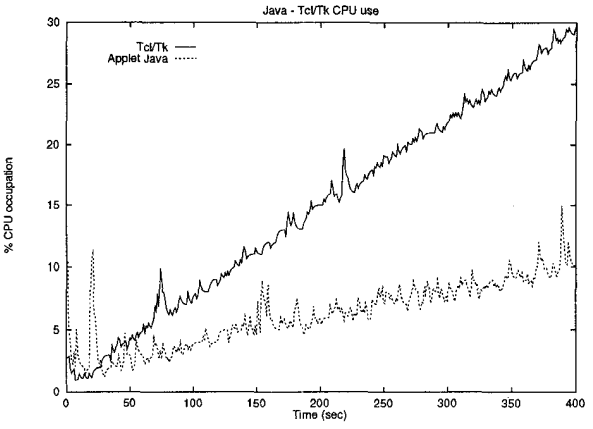

Figure 7. Performance comparison Tcl/Tk versus Java

is running continuously in the console side.

As the main disadvantage, the console software has to be installed in a particular hardware platform. Therefore, there is a trade-off between WWW universality and Tcl/Tk functionality.

Furthermore, performance for real-time monitoring is an important issue. A performance comparison between Tcl/Tk and WWW is done in the next subsection.

4.2.3 Performance comparison

We evaluate performance of Tcl/Tk versus Java. The key issue for network monitoring purposes is real-time graphics display capabilities. We code a simple algorithm that generates a plot such as the one shown in figure 6 and compare CPU occupation for Tcl/Tk and Java. The algorithm pseudocode is as follows:

```
while(true)  {
  plot a box with a random amplitude each
  second;
  rescale the graphic;
}
```

CPU occupation is measured using the UNIX *vmstat* tool in an Axil Ultima Sparc Station (167 MHz, 128 MB RAM). Results are shown in figure 7.

Interestingly, Java outperforms Tcl/Tk in CPU efficiency. This is due to the bytecode generation in Java language. Such bytecode is an intermediate step to generate machine-dependent code. Therefore, program interpretation is more efficient.

5. Performance evaluation

In order to evaluate PROMIS performance we artificially load an Ethernet network with file transfers between two hosts and perform network measurements.

We would like to evaluate PROMIS capabilities in comparison to SNMP/RMON based systems, which do not provide reliability in transport of monitoring information.

Figure 8 shows the results of real-time network measurements using PROMIS. Note that y-axis units are bytes per second. Ethernet throughput is around 8 Mbps, namely, the maximum utilization taking into account network interface card limitations. We do not detect any information loss from probe to manager, therefore the congestion period is displayed in the console accurately. The measurement step is one second, i. e. the probe is updating the console graphics display each second in real-time.

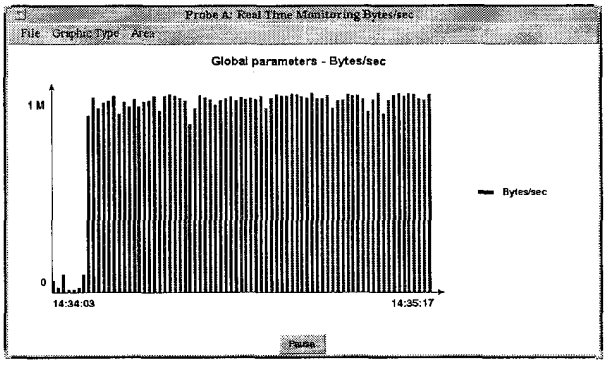

Figure 8. Performance evaluation of an overloaded network

On the other hand we perform the same experiment using UDP as transport protocol. We emulate SNMP

polling by sending ICMP ECHO packets of variable length from console to probe. We use two different hosts as console, both Sparc station with 10/100 Mbps high-performance network interface cards. The following figures are obtained using 70 series of 100 pings that are interleaved one second, for different type of network and different network utilization. The percentage of packet loss is averaged for each packet size.

In figure 9 appears the packet loss in the Local Area Network (LAN) and in a Campus Area Network (CAN), in both cases the LAN is overloaded to 80% utilization. Approximately the 7% of packets transmitted through the CAN are lost. With an unloaded LAN the loss would be negligible in both cases.

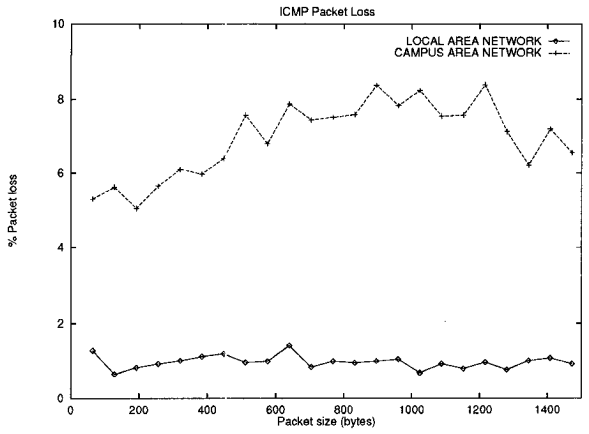

Figure 9. Packet loss in LAN and CAN

Worse results should be expected in a Wide Area Network (WAN) with several congested routers in the path from console to probe. We perform the UDP packet transmission experiment on the spanish academic network (see *http://www.rediris.es/red/#mapa red*). UDP packets are transmitted between a host in our university and a host located in Technical University of Madrid. The total number of hops in the static route between source and destination is 5 (4 routers). Figure 10 shows packet loss in such WAN with loaded and unloaded access LAN. Even though LAN utilization is 80%, mean packet loss is around 3%, since the bottleneck link is not in the access segment but in WAN routers, whose load is not significant. In both cases, loss increases with packet size as expected.

Finally, we evaluate packet loss over the Internet (transmission between our university host and an Internet Service Provider proxy), taking into account time periods. Peak hours in the Internet take place at noon (companies and universities) and evening (private users). Results are shown in figure 11. Packet loss

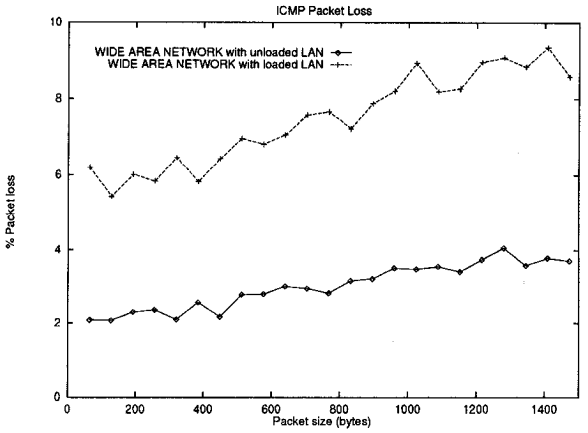

Figure 10. Packet loss in WAN

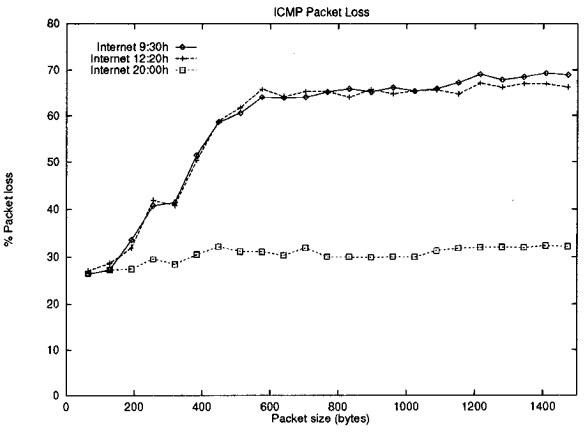

Figure 11. Packet loss over Internet

increases with packet size up to 70%, which is clearly unacceptable. The total number of hops in the static route between source and destination is 7 (6 routers). Internet bottlenecks have a dramatic effect on packet loss. As a result, the use of SNMP for network monitoring in the Internet scenario implies an unacceptable management information loss.

6. Conclusions

Current wide area networks are evolving to a high-speed scenario in which network monitoring has to be performed with higher granularity and reliability. The SNMP/RMON-based tools have not been designed for such environments since they use a non-reliable transport protocol and polling schemes for information recovery. On the other hand, current SNMP/RMON-based implementations are not prepared to provide suf-

ficient resolution in network measurements. We have presented the PROMIS tool, that overcomes the above-mentioned limitations using a reliable transport protocol and an optimized software architecture for performance. Furthermore, PROMIS provides a WWW and Tcl/Tk console, thus allowing for distributed network management in real-time.

References

[1] D. Case, M. Fedor, M. L. Schoffstall, and C. Davin. Simple network management protocol,RFC 1157. 1990.

[2] M. E. Crovella and A. Bestavros. Self-similarity in world wide web traffic: Evidence and possib le causes. In *ACM SIGMETRICS Annual Conference on Measurement and Modeling o f Computer Systems*, May 1996.

[3] E. Kim. *CGIs Developer's Guide.* SamsNet, 1996.

[4] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the self-similar nature of Ethernet traffic. *IEEE/ACM Transactions on Networking*, 2(1):1–15, January 1994.

[5] K. McCloghrie and M. T. Rose. Management information base for network management of TCP/IP-based internets: MIB-II, RFC 1213. 1991.

[6] E. Mier. Network world, bell labs evaluate SNMP on bridges. *Network World*, April 1991.

[7] E. Mier. Network world, bell labs test routers' SNMP agents. *Network World*, July 1991.

[8] V. Paxson and S. Floyd. Wide area traffic: The failure of Poisson modeling. *IEEE/ACM Transactions on Networking*, 4(2):226–244, April 1996.

[9] S. Waldbusser. Remote network monitoring management information base, RFC 1757. 1990.