High-layer Protocol and Service Management Based on Passive Network Traffic Monitoring: the Trace Management Platform

Luciano Gaspary[123], Edgar Meneghetti[3], Fabrício Wendt[2],
Lucio Braga[3], Roberto Storch[2], Liane Tarouco[3]

[1]Universidade do Vale do Rio dos Sinos, Centro de Ciências Exatas e Tecnológicas
Av. Unisinos 950 – CEP 93.022-000 – São Leopoldo, Brazil

[2]Universidade de Santa Cruz do Sul, Departamento de Informática
Av. Independência 2293 – CEP 96.815-900 – Santa Cruz do Sul, Brazil

[3]Universidade Federal do Rio Grande do Sul, Instituto de Informática
Av. Bento Gonçalves, 9500 – Agronomia – CEP 91.591-970 – Porto Alegre, Brazil

June 12, 2002

Abstract

A fast-growing number of high-layer protocols, services and networked applications has been run over computer networks and needs to be managed. A unified, decentralized framework should be used to scale for the current large and complex computing environments. Besides, the management environment should be quickly and easily adaptable to monitor dynamic scenarios. This paper presents the Trace Management Platform, an extension of the SNMP infrastructure based on the IETF Script MIB to support integrated, distributed and flexible management of high-layer protocols, services and networked applications.

1 Introduction

Computer networks currently experience a huge growth not only in size but also in the number of services offered, high-layer protocols and networked applications that flow over it. Regarding management of these network software, some of them are not critical (e.g. ICQ and Napster) and need no special care except characterization and accounting of traffic generated by them (for network impact measurement purposes). On the other side, some high-layer protocols, services and networked applications (e.g. DNS and HTTP server) support network-dependent businesses and, therefore, need to be carefully monitored and managed. Examples of some expected management tasks also include traffic characterization and accounting and extend to service testing and fault handling, performance measurement and intrusion detection (especially those that explore high-layer protocol vulnerabilities), to mention just a few.

For the network manager to be able to accomplish these management tasks for each of the relevant protocols, services and applications, an *integrated* management environment is desirable. Instead of using specific tools to monitor them individually (e.g. ICQ, Naspter, DNS, and HTTP server), one should use a unified framework. Besides being integrated, the size of current networks requires this management envi-

ronment to be *distributed*, so that the solution scales. The third requirement for the management environment is that it should be quick and easily adaptable to monitor dynamic scenarios (e.g. new protocols, services, and applications).

This paper presents the Trace Management Platform, an extension of the SNMP infrastructure based on programmable mid-level managers, monitoring, and action agents, in order to support integrated, distributed and flexible management of high-layer protocols, services and networked applications. The paper is organized as follows: section 2 describes expressive initiatives related to high-layer protocol, and briefly comments on distributed management. Section 3 presents PTSL, a graphical/textual language for protocol trace representation. Section 4 introduces the platform, its architecture and components. Section 5 closes the paper by presenting an evaluation of the platform, some conclusions and future work.

2 Related work

Many approaches have been proposed to both high-layer protocol management and distributed management. When it comes to high-layer protocol management, monitoring is the main topic of research. The ntop tool [1] is designed for traffic measurement and monitoring, and includes features for per-protocol network traffic characterization and usage. The Remote Network Monitoring Management Information Base Version 2 (RMON-2) [2], created in 1997, provides mechanisms to collect information similar to ntop.

Other recent efforts related to monitoring are the extensible architecture proposed by Malan and Jahanian [3] and the Realtime Traffic Flow Measurement (RTFM), developed by the group with the same name at the IETF [4] and implemented by the NeTraMet tool [5]. The RTFM architecture is based on distributed agents (called *meters*) that implement the RTFM Meter MIB. These agents are capable of making realtime packet flow measurement and accounting. The MIB allows an SNMP agent to query statistical data, as well as set agent configuration data. Flow specifications are made through a set of rules defined by a language called SRL [6] and determine (a) which flows should be accounted, (b) which nodes should be treated as flow origins and (c) which level of detail is desired for each flow.

A demand instigated by the fast proliferation of protocols and applications that flow over today's computer networks is the flexibility of monitoring tools. Many existing tools are not completely prepared to allow the monitoring of new protocols and applications and operate on a fixed set of them. New protocols can only be monitored through firmware updates, as with some RMON-2 probes, or by low-level programming languages, like the architecture proposed by Malan and Jahanian and ntop. Many network managers just end up neglecting these possibilities due to their complexity.

Other solutions are intrusive, due to the fact that they require applications to be developed using specific monitoring procedure calls. This approach is only suitable when monitoring is done within applications developed in-house, but it can not be used to manage proprietary protocols and/or applications (e.g. web browsers and clients). Besides, it is also needed to spend more money on personnel training on how to use the monitoring APIs.

The type and granularity of the collected information are important aspects associated with the monitoring. The RMON2 MIB and ntop collect statistics like the number of packets sent/received by a host or the number of packets exchanged between two peers, classified accordingly to the protocol used (HTTP, FTP, etc.). Advantages and disadvantages of the RMON2 MIB have been shown by Gaspary et al. in [7]. One of the weaknesses of both approaches is the lack of information related to performance and faults. These difficulties have been discussed by the IETF RMON working group through the Application Performance Measurement MIB (APM MIB) [8].

When it comes to granularity, accounting on the RMON2 MIB is made per host, pairs of hosts and protocol used. In the case of the ntop tool, it is possible to recognize and account packet flows, which are specified by a set of low-level rules that are processed by the BSD Packet Filter (BPF) [9]. In the RTFM architecture, only predetermined protocol fields can be read from captured packets (only up to the trans-

port layer). Information about high-layer protocols can not be considered due to this limitation. Besides, as it occurs with ntop, the same set of rules is applied to each captured packet, making it impossible to correlate messages from a same flow.

That is also very important to note that many management tools like [1, 3, 5] are limited to monitoring, leaving reactive and/or proactive management to the human manager when an unexpected network behavior is observed.

As for distributed management, Schoenwaelder et al. present in [10] several approaches and existing technologies for its deployment. Technologies based on the dynamic delegation of management tasks and, in special, the potential of delegation of those tasks through the IETF Script MIB [11] are discussed and commented. By using practical examples, they show how the monitoring of thresholds and services can be delegated to mid-level managers (MLM).

3 Protocol trace representation

The key technical aspect of our work is the use of passive monitoring to observe and count transactions of high-layer protocols, services and networked applications. The platform is not general-purpose, but specifically geared towards running and analyzing protocol traces and triggering custom scripts when certain conditions are met. The transactions to be monitored may represent scenarios related to fault, accounting, performance and security management.

In order to be able to monitor protocol traces it is necessary to define what exactly should be monitored (the protocol interactions to be observed). For this purpose, we have defined PTSL (Protocol Trace Specification Language), a language for the representation of protocol traces based on the concept of finite state machines (FSM). We have chosen this formalism because it provides a natural and intuitive view of protocols' behavior. A more detailed explanation of the language appears in [12, 13]. The language is composed of graphical (Graphical PTSL) and textual (Textual PTSL) notations, mentioned in subsections 3.1 and 3.2, respectively. These notations are not equivalent. The textual notation allows the complete representation of a trace in a descriptive fashion, including the specification of the FSM and the events that trigger transitions. In turn, the graphical notation covers only a subset of the textual notation, offering the possibility of graphically representing the FSM and only labeling the events that trigger transitions.

3.1 Graphical notation (Graphical PTSL)

The network manager can create a specification to monitor the whole protocol or just part of it. Interactions between more than one protocol can also be represented. Figures 1a and b show two trace examples. In the first case, (a), the trace monitors successful transactions to a Web server. The second trace, (b), does not describe a single protocol; instead it is made up of a name resolution request (DNS protocol), followed by an ICMP Port Unreachable message. This trace occurs when the host where the service resides is active, but the *named* daemon is not running.

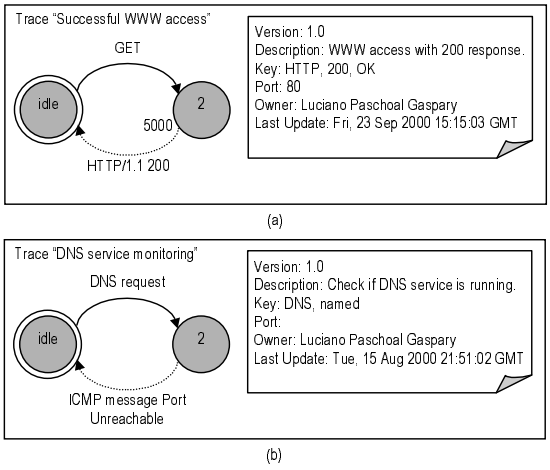

Figure 1: Graphical representation of a trace.

3.2 Textual notation (Textual PTSL)

Figure 2 presents the textual specification of the trace previously shown in figure 1a. All specifications writ-

ten in Textual PTSL start with the *Trace* keyword and end with the *EndTrace* keyword (lines 1 and 30). Catalog and version control information come right after the *Trace* keyword (lines 2–7). Forthwith, the specification is split into three sections: *MessagesSection* (lines 8–20), *GroupsSection* (not used in this example) and *StatesSection* (lines 21–29). In *MessagesSection* and *GroupsSection* the events that trigger transitions are defined. The FSM that specifies the trace is defined in *StatesSection*.

```
1  Trace "Successful WWW access"
2  Version: 1.0
3  Description: WWW access with 200 response.
4  Key: HTTP, 200, OK
5  Port: 80
6  Owner: Luciano Paschoal Gaspary
7  Last Update: Fri, 23 Sep 2000 15:15:03 GMT
8  MessagesSection
9  Message "GET"
10 MessageType: client
11 // OffsetType Encapsulation FieldNumber Verb Description
12 FieldCounter Ethernet/IP/TCP 0 GET "Request for a HTTP object"
13 EndMessage
14 Message "HTTP/1.1 200"
15 MessageType: server
16 MessageTimeout: 5000
17 FieldCounter Ethernet/IP/TCP 0 HTTP/1.1 "Protocol version"
18 FieldCounter Ethernet/IP/TCP 1 200 "Reply code"
19 EndMessage
20 EndMessagesSection
21 StatesSection
22 FinalState: idle
23 State idle
24 "GET" GotoState 2
25 EndState
26 State 2
27 "HTTP/1.1 200" GotoState idle
28 EndState
29 EndStatesSection
30 EndTrace
```

Figure 2: Textual representation of a trace.

4 Architecture of the Trace Management Platform

The Trace Management Platform is an extension of the SNMP centralized management infrastructure. Through a three-tier model, it supports the distributed management of high-layer protocols, services and networked applications. Figure 3 illustrates the platform's architecture. Based on the IETF Script MIB [11], it provides mechanisms to allow a management station to delegate management tasks to mid-level managers (MLMs) that, in turn, interact with monitoring and action agents to execute these tasks. PTSL specifications are used by MLMs to program monitoring agents that start sniffing packets flowing on the network and wait for traces to happen. With the information gathered from the monitoring process, the MLMs may launch procedures on action agents (Tcl or Perl scripts), enabling the automation of several management tasks (including reactive and proactive tasks). The platform also has notification mechanisms (traps) so that agents are able to report asynchronous events to scripts running on MLMs. These MLMs are then able to filter and/or correlate these traps and signal the occurrence of major events to the network management station (NMS). The components of the platform are presented below.

4.1 Management station

The platform is made of one or more management stations (managers). Through a web browser, the human manager has access to the management environment located on a Web server. For convenience, our research group chose the PHP language and the MySQL database to develop this environment. The highlighted modules on the management station may be hosted in the same station where the manager resides. If there is more than one management station, they may share the same environment core.

The most important tasks accomplished by the network manager from a management station are:

- *Registration of MLMs and agents:* to ease the coordination among the management station, MLMs and agents, the network manager must define who are the MLMs on the network, as well as the agents located (hierarchically) below these managers. Such binding is important to define management boundaries. When delegating a management task, the MLM will only manipulate those agents it is a parent of. The necessary interactions to this registration are presented in figure 3 (numbers 1, 2 and 3). This

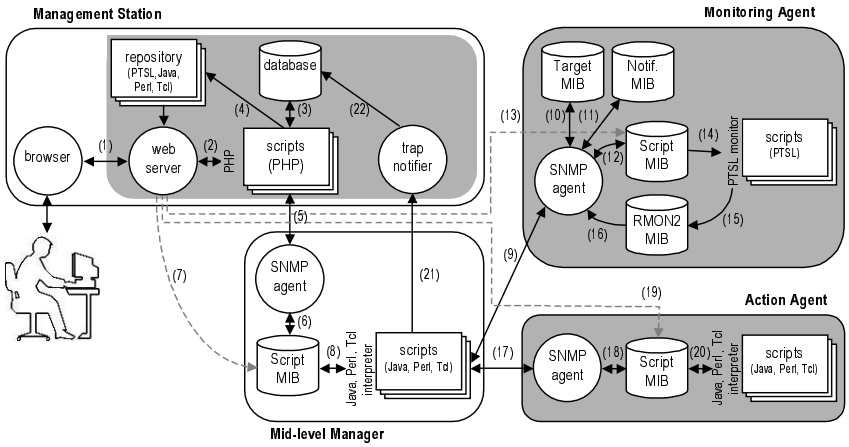

Figure 3: Components of the platform

numbering will be used henceforth in this section to illustrate the platform's data flow.

- *Specification of a protocol trace (PTSL script):* by using the language introduced in section 3, it is possible to specify a protocol trace. By means of a wizard, the network manager may specify the trace from scratch or reuse existing traces stored on the database, deriving a new specification based on previously defined traces (1, 2 and 3). For further use of this trace specification, it must be mapped from the database onto a text file and stored in the repository (4).

- *Specification of an action (Java, Perl or Tcl script):* the action scripts do not necessarily have to be specified using the web-based environment facilities. It is possible to upload a script to the repository (1, 2 and 4). It is recommended to test these scripts to exhaustion before sending them to the repository. Most Script MIB runtime environments offer debugging capabilities, but some do not.

- *Specification of a management task:* again, by using a wizard the management environment provides (see figure 4), the network manager specifies a management task (flows 1, 2 and 3 in figure 3). When defining the task, the network manager informs the trace to be observed, the identification of the object belonging to the extended RMON2 MIB (explained later), where the observations of the chosen protocol trace will be counted, the polling interval and the actions to be triggered when certain thresholds are reached. These specifications, as usually happens with PTSL specifications, are kept within the database.

- *Delegation of a management task:* to delegate a task to an MLM, the task must be retrieved from the database (1, 2 and 3). Besides, the network manager must choose the mid-level manager, the monitoring agent and the action agent (the latter is not mandatory) that will be responsible for the execution of the task. A corresponding Tcl script is automatically generated and made available at the repository (4). After going through these steps, the execution of the script is delegated to the MLM (5, 6) via SNMP (Script MIB).

- *Monitoring of a management task:* during the execution of a management task, the manager may query the MLM to get intermediate results of a running task (1, 2, 5 and 6).

- *Interruption of a management task:* the interruption of a management task requires the re-

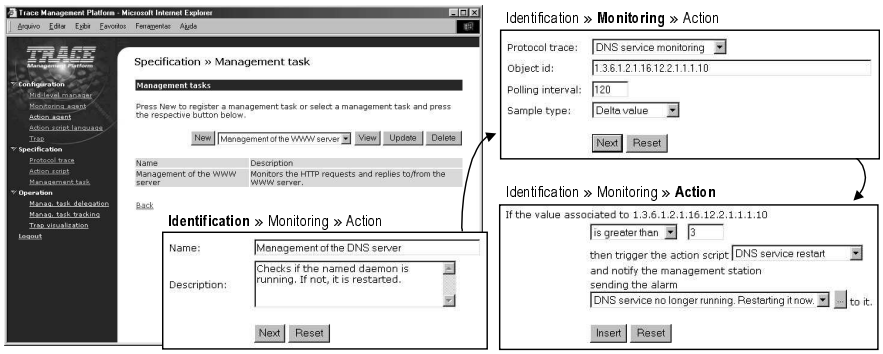

Figure 4: Specification of a management task using the wizard

moval of all programming made on the monitoring and action agents involved. Only after that it will be possible to terminate the execution of the script (i.e., the management task) at the MLM (1, 2, 5 and 6). It is important to mention that the latest release of the Script MIB provides mechanisms to expire and remove old (possibly forgotten) entries automatically.

- *Receiving and viewing traps:* the manager may receive traps through a module called Trap Notifier (21). When received, all traps are stored on the database (22). Traps are permanently retrieved by a PHP script (3) that updates the manager's web browser (2 and 1) using HTTP push technology.

4.2 Mid-level manager

The MLM runs and monitors management tasks delegated by NMSs and reports major events back to these stations. There may be one or more MLMs inside each network. The number of MLMs is determined by the network manager and depends on several factors (e.g. the size and complexity of the network infrastructure or human administrative boundaries).

The delegation of a task to an MLM, as mentioned, is performed by NMSs through SNMP primitives, which are supported by the PHP language (flows 5 and 6 in figure 3). When a new entry is created on the Script MIB launch table, the agent automatically downloads the script from the configured URL (7). After this table entry is enabled, the agent is then ready to start running the script (8).

As stated before, the management tasks specified by the network manager are automatically converted to Tcl scripts in order to be run by MLMs. Although Jasmin [14] (the Script MIB implementation used in our prototype) also supports Java and Perl, we have chosen Tcl because it has inherent network management characteristics and several libraries to support network management operations, besides being flexible and portable. The complexity of scripts run by MLMs is not a critical factor since all specification and delegation of management tasks is made by *wizards*, even though Tcl scripts can be easily written and understood by those not familiar with the language.

Figure 5 presents a sample script used to monitor the occurrence of a trace. It was generated automatically by the platform from the specification presented in figure 4. In lines 8–11 and 12–16 the monitoring and action agents are programmed, respectively. The monitoring agent is, in line 17, asked to start observing the network for the occurrence of the trace just programmed. Then, the MLM polls it every 120 second (line 28) to get information (lines 3 and 20) and checks whether the trace has been counted or not (line 22). If the trace has been observed three times within an interval, another script is launched at the action agent (line 23), to run a management procedure. Intermediate and final results are generated by

the script (lines 28 and 29) and made available in the Script MIB.

The script running at the MLM can configure which traps it wishes to receive by using the Target and Notification MIB's installed on monitoring and action agents. On the Target MIB, the MLM sets its IP address and UDP port number to where traps are sent (see figure 3, flows 9 and 10) (this port number must be unique among all scripts running at the MLM). The Notification MIB allows the script to set which traps it wishes to receive (these are filtered at the notifier) (9, 11) [15]. If the script implements a trap handler, it can run a procedure whenever a trap arrives. Traps can be correlated and a more valuable notification may be sent to the NMS (21). This configuration of trap sinks eases the implementation of monitoring and action scripts, since they do not have to care about which are their trap sinks and which credentials should be used to send them.

```
1  package require Tnm 3.0
2  package require Trace 1.0

3  set oid "protocolDist.protocolDistStatsEntry.protocolDistStatsPkts.1.10"
4  set prev 0

5  if { [catch {::Tnm::snmp generator -address $agent} s] } {
6     ::Tnm::smx exit -code runtimeError "Error creating SNMP session: $s"
7  }

8  if {[catch {Trace::InstallScript $ma $m_owner $m_name $m_lang $m_src \
9        $m_descr $m_args $m_ltime $m_etime $m_mrun $m_mcomp} e]} {
10    ::Tnm::smx exit -code runtimeError "Error installing script: $e"
11 }

12 if {[catch {Trace::InstallScript $aa $a_owner $a_name $a_lang $a_src \
13       $a_descr $a_args $a_ltime $a_etime $a_mrun $a_mcomp} e]} {
14    ::Trace::UninstallScript $ma $m_owner $m_name
15    ::Tnm::smx exit -code runtimeError "Error installing script: $e"
16 }

17 ::Trace::RunScript $ma $m_owner $m_name 0

18 proc monitor {} {
19    global s oid prev
20    set val [$s get $oid]
21    set val [lindex [lindex $val 0] 2]
22    if {[expr $val - $prev] > 3} {
23       ::Trace::RunScript $aa $a_owner $a_name 1
24    }
25    set prev $val
26 }

27 ::Tnm::job create \
28       -interval 120000 -error {::Tnm::smx exit -code runtimeError $errorInfo} \
29       -exit {::Tnm::smx exit} -command {monitor}

30 vwait forever
```

Figure 5: Sample script run by MLMs

It can be noted that the communication between MLMs and monitoring or legacy SNMP agents (handled by the Tcl scripts) is made through SNMP primitives provided by Tcl through the Scotty package [16]. The same happens between MLMs and the management station when traps are sent. The programming of the Script MIB on the monitoring (9, 12) and action agents (17, 18) is made with the aid of a specially developed Tcl package (see line 2 in figure 5), called Trace.

4.3 Monitoring agent

The monitoring agents count the occurrence of traces on the network segment where they are located. They are called extensible because the traces to be monitored can be dynamically configured. The configuration of which traces should be monitored at a given moment is made by the MLM through the Script MIB (see flows 9 and 12 in figure 3). On the script run by the MLM (figure 5), it is possible to see how the monitoring agent is programmed (lines 8–11). One of the parameters passed is the URL of the script (PTSL specification) that will be run. When the MLM requests the installation and execution of a script, it is retrieved from the repository via HTTP (13) and executed (14). Actually, the PTSL is not executable. The semantics associated to line 17 in figure 5 makes the monitoring agent start monitoring a new trace. In an analogous way, the interruption of a script on the Script MIB means programming the monitoring agent so that it ceases monitoring the trace defined by the script.

Every time a trace is observed between any pair of peers, data is stored on a mySQL database. This database is source of information for the SNMP subagent that implements an extended version of the RMON2 MIB [2, 7] (15) (also developed by our research group). One of the differences between our MIB and RMON2 is that the *protocolDir* group, which indicates which protocol encapsulations the agent is capable to monitor, now allows protocol traces to be indexed.

The *alMatrix* group from the RMON2 MIB stores statistical data about the trace when it is observed between each pair of peers. Table 1 illustrates the contents of the *alMatrixSD* table of our extended

MIB. It counts the number of packets/octets between each pair of peers (client/server) at the granularity of protocol traces.

Table 1: Information from the alMatrixSD table

Source	*Destination*	*Protocol*	*Packets*	*Octets*
172.16.108.1	172.16.108.2	DNS service monitoring	4	4.350
172.16.108.32	172.16.108.2	DNS service monitoring	8	7.300
172.16.108.1	172.16.108.254	Successful WWW access	254	1.202.126
125.120.10.100	172.16.108.254	Unsuccessful TCP connection attempt	20	3.204

One disadvantage of the RMON2 MIB is that it does not have the capability of generating performance information. Because of that, our group is currently evaluating the possibility of using an extension of the RMON2 MIB, the Application Performance Measurement MIB [8]. Table 2 presents the type of information stored by this MIB. The first line indicates that the *Successful WWW access* trace was observed 127 times between hosts *172.16.108.1* and *172.16.108.254*. The number of traces that did not complete with success was 232 and the mean response time for successful observations was 6 seconds.

Table 2: MIB with performance information

Client	*Server*	*Protocol*	*Success.*	*Unsuccess.*	*Responsiv.*
172.16.108.1	172.16.108.254	Successful WWW access	127	232	6 sec.
172.16.108.1	200.248.252.1	Successful WWW access	232	112	17 sec.
10.10.135.125	200.248.252.1	SYN Flood	10.234	56	3 sec.

4.4 Action agent

Through monitoring agents, MLMs are able to evaluate whether a trace has occurred or not. Traces may represent network service failures, intrusion attempts, service performance degradation, and other problems. In this context, the action agents are responsible for the execution of reactive (and potentially proactive) management procedures created to autonomously handle these problems. Let's take, for instance, the DNS service monitoring. When a mid-level manager detects that the service is not running (through the monitoring loop), it can ask an action agent (located on the same host of the service) to run a script developed in Java, Perl or Tcl to restart the service.

The communication between MLMs and action agents is made through the Script MIB (see flows 17 and 18 in figure 3). Once the Script MIB is programmed to run an action script, it is retrieved via HTTP from the repository (19) and then executed (20).

5 Conclusions and future work

This paper presented an open platform for integrated, distributed and flexible management of high-layer protocols, services and networked applications based on the use of programmable agents. Based on the IETF SNMP standard, this platform does not require major changes in existing management systems (which took years to consolidate). The use of MIBs as source of information for the management tasks makes our approach more homogeneous. Information that in other approaches depend on proprietary mechanisms to be gathered (e.g. CPU utilization and memory usage) is retrieved from standardized MIBs (e.g. Host Resources MIB [17]) in ours. Since the platform's management environment has been developed using PHP, it can be fully customized (e.g. new wizards can be easily created).

The effort to monitor a new application is not high with the proposed approach. To do that the network manager is supposed to select the protocol traces of his interest and specify them using the PTSL language. Once the protocol traces are specified, the network manager must define the management tasks (what is going to happen when the traces have been observed). As the platform provides wizards to accomplish both tasks, the complexity of monitoring a new application does not reside in the usage of the platform, but in the knowledge that the network manager must have of the protocol used by the application to be monitored.

Regarding flexibility, the proposal of the PTSL language is one of the most important contributions of this work. All approaches discussed and listed in section 2 are limited to the accounting of sent/received packets between pairs of peers, classifying them according to protocols [2] or flows [1, 5]. In these approaches, the manager has access to information

limited to the style "host A sent n octets/packets to host B", with filters to some well-known protocols (e.g. HTTP and SMTP) or packets with specific header fields. The innovations aggregated with PTSL increase the granularity in which protocols are monitored, enabling the analysis of the behavior of a protocol or just part of a protocol by introducing the representation of desired traces. This feeds the network manager with more accurate information, which will help him/her deploy fault, configuration, accounting, performance and security management to high-layer network protocols and services. Using the previous example, the language allows the accounting of successful, unsuccessful and unauthorized HTTP accesses, as well as many other possible HTTP behavior. The PTSL power of expression is another strong point. While many approaches allow the selection of packets based on a few predetermined header fields only up to the transport layer [5], PTSL goes further, allowing the use of filters based on any protocol, all the way up to the application layer.

Integrated management is an inherent characteristic of the platform. Instead of using specific tools to monitor individual protocols and services (web, video-on-demand, etc.), one can use the Trace Management Platform to monitor such protocols, services and applications through a unified framework. By delegating the functionality of these tools to distributed management stations, our approach burdens off the workload on the hosts where these services are installed.

One positive aspect of the Trace Management Platform is the possibility of making effective management of high-layer network protocols, services and applications by integrating the PTSL language with programmable monitoring agents and by associating the occurrence of specific traces to dynamically programmable actions, enabling the automation of a set of management procedures. The proposed platform is not limited to monitoring. On the contrary, it provides a more complete and broader solution that includes the execution of actions, enabling both reactive and proactive management.

Another positive aspect of the platform is a significant increase of scalability in relation to the traditional SNMP management paradigm, since it can delegate management tasks, previously processed only at the centralized management station, to MLMs. The robustness aggregated to the management tasks also represents an important contribution. The platform allows the delegation of management functions to MLMs that are closer to the monitored agents; if the connection is lost between the centralized management station and the MLM, these management tasks will still be able to run. The delegation is not only about tasks, but it also will delegate CPU cycles and will keep polling as close as possible to the management targets.

Regarding security, the Script MIB supports all facilities provided by SNMPv3, including the User-based Security Model (USM) and the View-based Access control (VACM). In [18] Schönwälder and Quittek describe the security aspects related to the Script MIB in detail. Using these facilities it is assured that the monitoring and action agents cannot be "reprogrammed" by a person who is not allowed to do this.

However, this distributed platform demands more work to be controlled. The component management becomes quite a complex task. Included in the component management are the distribution and update of scripts, as well as the retrieval and correlation of results. The creation of mechanisms that allow transparent use of the platform (e.g. wizards) helped us to hide much of this complexity from the network manager.

We are now working to release a version of the platform under the GPL2 licence. Performance tests of both the Script MIB and the monitoring engine are being carried out. However, Schönwälder presents good results in [10], where the Jasmin implementation has been evaluated.

References

[1] L. Deri and S. Suin. Ntop: Beyond Ping and Traceroute. *Proc. 10th IFIP/IEEE Workshop on Distributed Systems: Operations and Management*, Zurich, Oct. 1999, p. 271–283, Springer Verlag.

[2] S. Waldbusser. Remote Network Monitoring Management Information Base Version 2 using

SMIv2. RFC 2021, INS, Jan. 1997.

[3] G. Malan and F. Jahanian. An Extensible Probe Architecture for Network Protocol Performance Measurement. *Proc. SIGCOMM*, Vancouver, Sep. 1998.

[4] N. Brownlee, C. Mills and G. Ruth. Traffic Flow Measurement: Architecture. RFC 2722, The University of Auckland, GTE Laboratories, Inc., GTE Internetworking, Oct. 1999.

[5] N. Brownlee. NeTraMet. http://www.auckland.ac.nz/net/Internet/rtfm/.

[6] N. Brownlee. SRL: A Language for Describing Traffic Flows and Specifying Actions for Flow Groups. RFC 2723, The University of Auckland, Oct. 1999.

[7] L. P. Gaspary and L. R. Tarouco. Characterization and Measurements of Enterprise Network Traffic with RMON2. *Proc. 10th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, Zurich, Oct. 1999, p. 229–242, Springer Verlag.

[8] S. Waldbusser. Application Performance Measurement MIB. Internet Draft, February 2002.

[9] S. McCanne and V. Jacobson. The BSD Packet Filter: A New Architecture for User-Level Packet Capture. *Proc. Winter USENIX Conference*, 1993.

[10] J. Schönwälder, J. Quittek and C. Kappler. Building Distributed Management Applications with the IETF Script MIB. *IEEE Journal on Selected Areas in Communications*, 18(5):702–714, 2000.

[11] D. Levi and J. Schönwälder. Definitions of Managed Objects for the Delegation of Management Scripts. Request for Comments 3165, Nortel Networks, TU Braunschweig, August 2001.

[12] L. P. Gaspary, L. F. Balbinot, R. Storch, F. Wendt and L. R. Tarouco. Towards a Programmable Agent-based Architecture for Enterprise Application and Service Management. *Proc. First IEEE/IEC Enterprise Networking Applications and Services Conference*, Atlanta, Jun. 2001, p. 39–46.

[13] L. P. Gaspary, L. F. Balbinot, R. Storch, F. Wendt and L. R. Tarouco. Distributed Management of High-Layer Protocols and Network Services through a Programmable Agent-Based Architecture. *Proc. IEEE International Conference on Networking*, Colmar, France, Jul. 2001, part 1, p. 204–217, Springer Verlag.

[14] TU Braunschweig, NEC C&C Research Laboratories. Jasmin - A Script MIB Implementation. http://www.ibr.cu.tu-bs.de/projects/jasmin.

[15] D. Levi, P. Meyer and B. Stewart. SNMP Applications. RFC 2573, SNMP Research, Inc. , Secure Computing Corporation, Cisco Systems, Apr. 1999.

[16] TU Braunschweig. Scotty. http://www.ibr.cs.tu-bs.de/projects/scotty/.

[17] S. Waldbusser and P. Grillo. Host Resources MIB. RFC 2790, Lucent Technologies Inc. , WeSync.com, Mar. 2000.

[18] J. Schönwälder, J. Quittek. Secure Internet Management By Delegation. *Computer Networks*, 35(1):39–56, January 2001.