Trace: An Open Platform for High-Layer Protocols, Services and Networked Applications Management

Luciano Gaspary, Edgar Meneghetti, Fabrício Wendt, Lucio Braga, Luis F. Balbinot, Roberto Storch, Liane Tarouco

Federal University of Rio Grande do Sul
Instituto de Informática
Av. Bento Gonçalves, 9500 - Agronomia - CEP 91.591-970 - Porto Alegre, Brazil

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. By using a user-friendly interface (wizards), the network manager can specify management tasks, protocol traces and action scripts. The management tasks are automatically converted to scripts by the platform and delegated to mid-level managers (MLMs) on demand. When executing a management task, the MLM delegates a protocol trace to a monitoring agent. Once programmed, this agent starts to monitor the occurrence of the trace (transaction). The information gathered by the monitoring agent is retrieved and analyzed by the mid-level manager, which may delegate the execution of procedures (action scripts) to action agents and notify the management station.

Keywords: Internet management, network management, network monitoring, scripting.

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, Napster, DNS, and HTTP server), one should use a unified framework. Besides being integrated, the size of current networks requires this management environment 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 traffic and application management and briefly
2 Related Work

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 transactions to be monitored may represent scenarios related to fault, accounting, performance and security management. Under the popular classification of management areas (device, network, traffic, system, application and service), our work is best understood as traffic and application management. Therefore, below we address related work on traffic and application management, as well as some topics on distributed management.

2.1 Traffic Management

Many approaches have been proposed to traffic management, most of them focused on monitoring. The Ntop tool [1] includes features for per-protocol network traffic characterization and usage. The Remote Network Monitoring Management Information Base Version 2 (RMON2) [2] provides mechanisms to collect information similar to Ntop. Other efforts related to monitoring are the extensible architecture proposed by Malan and Jahanian [3] and the IETF Realtime Traffic Flow Measurement (RTFM) architecture [4] (implemented by the NeTraMet tool [5]). The later is based on distributed agents (called meters) that are capable of making real-time packet flow measurement and accounting.

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 RMON2 probes, or by using low-level programming languages, like the architecture proposed by Malan and Jahanian and the Ntop tool. Many network managers just end up neglecting these possibilities due to their complexity.

The type and granularity of the collected information are important aspects associated with the monitoring. The RMON2 MIB and the Ntop tool 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 (e.g. HTTP and FTP). Advantages and disadvantages of the RMON2 MIB have been shown 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 count 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 parameters (specified using the SRL language [6]) can be read from captured packets (up to the transport layer). Information about application-layer protocols cannot be considered due to this limitation. Besides, in all approaches mentioned the same set of rules is applied to each captured packet, making it impossible to correlate messages from the same flow.

It is also important to note that many traffic 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.

2.2 Application Management

Regarding application management, there is a variety of tools available that, normally, are integrated (a desirable feature) to commercial management suites (e.g. Spectrum [10], Tivoli [11], Patrol [12], and eHealth [13]). These tools focus mostly on fault and performance management, but some address configuration, accounting and security as well.

Existing suites lack flexibility. They provide specific tools or modules to manage certain applications (e.g. Apache Web server and MS Exchange). If interested in managing another application (not supported by the
suite), the network manager must use SDKs to program the desired tools/modules, which is not an ordinary task.

The information collected by these tools has fine granularity, including system status (e.g. CPU utilization and memory usage), application logs and response time. To be able to gather such information, agents must be located in the host where the monitored application resides, thus burdening its workload. Performance-related tools such as eHealth Application Response, Spectrum Response Time, Tivoli Application Performance and Patrol for Internet Services explore several measurement techniques. The client capture method (based on the recognition of changes in the GUI on the end user’s desktop) requires agents to be pushed to the end user station. Transaction simulation measures the response time of real transactions run by any station designated as a synthetic user. A more powerful measurement technique requires application instrumentation on client and/or server side. All of these techniques are intrusive (in different degrees) and pose limitations. The client capture method may not scale well depending on the number of hosts being monitored. Application instrumentation is useful to monitor applications developed in-house, but it cannot be used to manage proprietary protocols and/or applications (e.g. web browsers and clients). Besides, it is also necessary to spend more money to train personnel on how to use the monitoring APIs.

In contrast to traffic management tools, most of the application management suites (e.g. Spectrum Application Manager, Tivoli Web Services Manager and Tivoli Management Solution for Exchange) provide the network manager with mechanisms to set thresholds to monitored attributes. When reached, a threshold may trigger a customized script to solve a potential problem and notify the management station.

2.3 Distributed Management

As for distributed management, Schönwälder et al. present in [14] 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 [15] 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).

2.4 The Trace Approach

In the next sections we present our approach. It explores non-intrusive techniques (by observing network traffic) to recognize events related to high-layer protocols, services and networked applications. Compared to other traffic management approaches ours offers an enhanced level of granularity (transactions can be monitored). Hence, more than just traffic accounting can be accomplished including fault, performance and security management (shown in the next sections). Compared to current application management solutions our approach may not provide the same fine granular information (not mandatory for many management tasks) but is supposed to be more flexible and lightweight. Our platform is further evaluated in section 6.

3 Protocol Trace Representation

This section briefly presents PTSL (Protocol Trace Specification Language), a language for the representation of protocol traces based on the concept of finite state machines (FSM). A more detailed explanation of the language appears in [16, 17]. The language is composed of graphical (Graphical PTSL) and textual (Textual PTSL) notations, described in subsections 3.1 and 3.2, respectively. These notations are not equivalent. The textual notation allows the complete representation of a trace, 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.

![Graphical representation of a trace. (a) Successful WWW request. (b) DNS request not replied because the *named* daemon is not running.](image)

**Representation of states and transitions.** States are represented by circles. From the initial state (*idle*) other *n* states can be created, but they must always be reachable through any given transition. The final state is identified by two concentric circles. In both examples (figures 1a and b) the initial and final states are the same. The state transitions are represented by unidirectional arrows. The continuous arrow indicates that the transition is triggered by the client host, while the dotted arrow determines that the transition is triggered by an event coming from the server host. The text associated with a transition is merely a label to the event that triggers it; the full specification can only be made via textual notation.

**Representation of timers.** Transitions, by default, do not have a time limit to be triggered. To associate a time-out with a transition, an explicit value (in milliseconds) must be set. In the example shown in figure 1a, the value 5000 associated to transition HTTP/1.1 200 indicates that the transition from state 2 to the initial state has up to five seconds to be triggered.

**Representation of information for cataloging and version control.** The graphical notation also offers a constructor where information about the trace, which is relevant to cataloging and version control of specifications, is included (see figure 1a and b). Besides this data, there is also a *Port* field, used to indicate the TCP or UDP port of the monitored protocol; this should only be defined when the trace is limited to a single protocol.

### 3.2 Textual Notation (Textual PTSL)

Figure 2 presents the textual specification of the trace previously shown in figure 1a. All specifications written 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`. Below, we discuss the representation of messages, message groups, and the FSM.

**Representation of messages.** Whenever the fields of a captured packet match the ones specified at a `Message` for the current state, a transition is triggered in the FSM. The way those fields are specified depends on the type of protocol to be monitored. In the case of variable-length character-based protocols where fields are split by white space characters (e.g. HTTP and SMTP), the identification of a field is made by its position within the message (this is called the FieldCounter strategy). In HTTP/1.1 200, for instance, HTTP/1.1 is at position 0 and 200 is at position 1. On the other hand, the identification of binary protocols, known by their fixed length fields (e.g. TCP), is determined by a bit offset starting from the beginning of the protocol header; it is also needed to specify the size of the field, in bits (this is the BitCounter strategy).

The trace shown in figure 1a is for a character-based protocol. The `GET` message specification is shown in figure 2 (lines 9–13). In line 10 the message is defined as being of type `client`, meaning that the state transition associated with the message will be triggered by the client host. In line 12 the only field that is analyzed is
specified. All information necessary to identify it is: fetch strategy (FieldCounter), protocol encapsulation (Ethernet/IP/TCP), field position (0), expected value (GET) and, optionally, a field description. Character-based protocol fields are always identified by this quintuple. The trace reply message HTTP/1.1 200 is shown in lines 14–19. The message type is defined in line 15 as server, i.e., the state transition will be triggered by the server host. In line 16 the MessageTimeout is set to 5000. Finally, the two fields to be analyzed are defined (lines 17 and 18).

As opposed to the example mentioned above, the trace specified by the messages in figure 1b is based on binary protocols, namely DNS and ICMP. A DNS request will trigger a state change in the FSM from idle to 2. To recognize a DNS request from the packets flowing over the network, two fields must be observed: QR (when set to 1 indicates a request to the server) and OPCODE (when set to 0 represents a standard query). Field QR is 16 bits away from the beginning of the header and its size is 1 bit. Field OPCODE starts in the seventeenth bit and occupies 4 bits.

Figure 3 presents part (the MessagesSection) of the textual specification for the trace shown in figure 1b. Lines 1–6 describe the DNS request. In line 4 the QR field is defined. The information needed to identify a binary protocol field is: fetch strategy (BitCounter), protocol encapsulation (Ethernet/IP/UDP), field position (16), field length (1), expected value (1) and, optionally, a field description. The information used to identify the OPCODE field is BitCounter, Ethernet/IP/UDP, 17, 4 and 0. The same strategy is used in the definition of ICMP message Port Unreachable in lines 7–11.
Representation of message groups. The PTSL language allows the binding of a single transition to multiple distinct events. To do that, the Group constructor must be used within the GroupsSection section. The trace presented in figure 1a monitors the occurrence of successful HTTP accesses. However, only accesses with reply code 200 are counted. Accesses with reply codes 201, 202, 203, 204, 205 and 206, which also represent successful operations, could be included into this accounting. In order to make that possible, the messages that identify these accesses must be defined (similarly to lines 14–19 in figure 2) and grouped (see figure 4). In lines 2–3 all messages that make part of the group are listed. In the graphical representation (figure 1a), the label associated with the transition from state 2 to idle changes from HTTP/1.1 200 to HTTP/1.1 20X, which is the name of the new message group (line 1).

Representation of the FSM. Lines 21–29 in figure 2 define the textual specification of the state machine shown in figure 1a. The final state is identified just after StatesSection (line 22). The states idle and 2 are defined in lines 23–25 and 26–28, respectively. The state specification only lists the events (messages and groups) that may trigger transitions, indicating, for each one, which is the next state (lines 24 and 27).

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 5 illustrates the platform’s architecture. Based on the IETF Script MIB [15], 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.

Figure 5: Components of the platform

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 5 (numbers 1, 2 and 3). This 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 6), the network manager specifies a management task (flows 1, 2 and 3 in figure 5). 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 protocol traces 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 removal 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.

Figure 6: Specification of a management task using the wizard
- 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 5). 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 [18] (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 7 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 6. 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.

![Figure 7: Sample script run by MLMs](image_url)
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 5, 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) [19]. 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.

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 [20]. 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 7), 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 5). On the script run by the MLM (figure 7), 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 7 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.

The monitoring agent was implemented in C using threads. A detailed view of its architecture is shown in figure 8. The PTSL Manager module is responsible for the integration between the Script MIB and the PTSL runtime environment. It updates the data structures used by the PTSL engine whenever a new trace is programmed to be monitored or an existing one is removed (when not needed anymore). Two threads (queue thread and PTSL engine) operate in a producer/consumer fashion. The former is responsible for capturing all packets using the pcap library and adding them to a circular queue. The latter processes each queued packet in order to find out if it is supposed to trigger state changes in the programmed state machines.

![Figure 8: A detailed view of the monitoring agent’s architecture](image)

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 sub-agent 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/octet between each pair of peers (client/server) at the granularity of protocol traces.

One disadvantage of the RMON2 MIB is that it does not have the capability of generating performance
Table 1: Information from the `alMatrixSD` table

<table>
<thead>
<tr>
<th>Source Address</th>
<th>Destination Address</th>
<th>Protocol</th>
<th>Packets</th>
<th>Octets</th>
</tr>
</thead>
<tbody>
<tr>
<td>172.16.108.1</td>
<td>172.16.108.2</td>
<td>DNS service monitoring</td>
<td>4</td>
<td>4.350</td>
</tr>
<tr>
<td>172.16.108.32</td>
<td>172.16.108.2</td>
<td>DNS service monitoring</td>
<td>8</td>
<td>7.300</td>
</tr>
<tr>
<td>172.16.108.1</td>
<td>172.16.108.254</td>
<td>Successful WWW access</td>
<td>254</td>
<td>1.202.126</td>
</tr>
<tr>
<td>125.120.10.100</td>
<td>172.16.108.254</td>
<td>Port scanning</td>
<td>20</td>
<td>3.204</td>
</tr>
</tbody>
</table>

This table 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

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
<th>Protocol</th>
<th>Successful</th>
<th>Unsuccessful</th>
<th>Responsiveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>172.16.108.1</td>
<td>172.16.108.254</td>
<td>Successful WWW access</td>
<td>127</td>
<td>232</td>
<td>6 sec.</td>
</tr>
<tr>
<td>172.16.108.1</td>
<td>200.248.252.1</td>
<td>Successful WWW access</td>
<td>232</td>
<td>112</td>
<td>17 sec.</td>
</tr>
<tr>
<td>10.10.135.125</td>
<td>200.248.252.1</td>
<td>SYN Flood</td>
<td>10.234</td>
<td>56</td>
<td>3 sec.</td>
</tr>
</tbody>
</table>

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 to restart the service such as the one shown in figure 9.

```perl
#!/usr/bin/perl
my $pid;

# Verify if the process named is executing.
if ("/var/run/named.pid" { 
    $pid =chèxist("/var/run/named.pid");
})

if (fnum snt running, restart using a HUP signal, otherwise instantiate the
    if (defined $pid) { 
        print "Restarting named (sending HUP signal);", 
            exit(HUP signal); 
    } else { 
        print "Service named was not running);", 
            /named restarted. 
    }

# Test if the process is executing.
if ("/var/run/named.pid") { 
    $pid =chèxist("/var/run/named.pid"), 
        print "The named daemon is up and running as PID $pid", 
    } else { 
        print "The named daemon could not be started;", 
    }
```

Figure 9: Perl script to restart the name resolution service

The communication between MLMs and action agents is made through the Script MIB (see flows 17 and 18 in figure 5). Once the Script MIB is programmed to run an action script, it is retrieved via HTTP from
the repository (19) and then executed (20). The script illustrated in figure 9 was written using Perl language. Although most network managers are more familiar with Perl, this language is not mandatory; Java and Tcl can also be used.

5 Case Study

The Trace Management Platform was designed to allow the management of all functional areas (FCAPS). Our group explored, through a case study, the characteristics of the platform to validate its applicability on the management of high-layer protocols and network services. Figure 10 shows a real management scenario, composed of three domains. The organization of these domains is a task that the network manager must handle to efficiently use the platform. This task is accomplished at the management environment when the MLMs and the agents are registered.

Domain 1 is composed of equipment and services related to the organization’s Internet access (dark gray in the figure). The router acts as a gateway for three distinct networks: the Internet, the internal network demilitarized zone (where the Web and DNS servers are) and the protected Intranet. There are two monitoring agents (M1 and M2) installed on dedicated monitoring stations and one action agent (A) installed on the same host where the DNS server resides. Based on this scenario, our work group defined some management tasks.

5.1 Fault Management of the DNS Service

It consists in observing the availability of the name resolution service. Through the Tcl script presented in Figure 7, the mid-level manager responsible for domain 1 programs the monitoring agent M1, which is located on the same segment as the DNS server, in order to watch for the occurrence of the trace DNS service monitoring (Figure 1b). If this trace is observed at least three times during a polling interval, the mid-level manager will request the action agent A1 to launch the script that restarts the service (Figure 9).

5.2 Accounting and Performance Management of the Web Servers

A variety of statistics related to a Web server may be generated by the platform. In our case study we illustrate how to measure the amount of accesses (successful and unsuccessful) to the web server, which include access failures and unauthorized access attempts. By collecting this information, the manager may be able to (i) know the most critical access periods and upgrade the server to support more concurrent connections, (ii) count and minimize problems with HTTP clients (e.g. by revising the web pages), and (iii) reconfigure the web server and/or the firewall to accept no more connections from hosts where unauthorized access attempts originate.

Figures 1a and 2 present the trace used to count successful accesses to the web server. Traces used to count other reply codes are very similar. Thus, for instance, in order to count unauthorized access attempts, one may define a trace to monitor the occurrence of the GET query message followed by an HTTP/1.1 401 response message. By monitoring several reply codes during regular polling intervals, MLMs can generate more detailed and information-rich reports (see figure 11).
If the monitoring agents implement the APM MIB, then further performance statistics may be gathered. With the data provided by that MIB, the manager can measure, for instance, the mean response time of an HTTP transaction (see figure 12). To monitor these times, the trace presented in figures 1a and 2 is used. The mean response time of any application-specific transaction could be monitored in this fashion. An advantage of our approach in this case, since it is based on passive monitoring, is that no synthetic traffic need to be generated and no piece of software need to be pushed to the end-user station.

The monitoring of the Web server located on the external network is done by programming the monitoring agent \( \text{M}_1 \), whereas the Intranet server is monitored by the agent \( \text{M}_2 \). If the network manager is interested in measuring the response time perceived by users accessing a certain application in the external web server, then monitoring agents should be installed on the network segments where these users are located.

5.3 Security Management of the DNS and Web Servers

The Trace Management Platform is a powerful tool for security management. It can act as a network intrusion detection system, looking for traces that represent attack scenarios. An example of a common attack is port scanning. A port scan consists of sending packets to a range of ports of a host to know which TCP and UDP services are available. When using TCP, if the host has no a service listening in a determined port, it will send back a TCP packet with the \( \text{RST} \) bit on in response to the connection attempt. Figure 13a presents this trace.

The mid-level manager programs the monitoring agent \( \text{M}_1 \) to start monitoring the trace. Furthermore, it will regularly poll the extended RMON2 MIB where the monitoring results are stored (see table 1). If during a polling interval the number of occurrences is higher than a determined value, defined by the manager the script will generate a notification to the central management station.

A similar procedure is done when one of the stations suffers from an attack known as SYN Flood. This attack consists of sending a huge number of connection setup packets (TCP packet with the \( \text{SYN} \) flag on) with a fake source address to a target host. This fake address must be unreachable or non-existent (usually a reserved value). When the target host receives these \( \text{SYN} \) packets, it creates a new entry on its connection table and sends a \( \text{SYN/ACK} \) packet back to the possible client. After sending the reply packet, the target host waits for acknowledgement from the client to establish the connection. As the source address is fake, the server will wait a long time for this reply. In a given time, the connection queue of the server will be full and all new connection requests will be discarded, creating a denial of service. This state will last until the entries on the connection table start to timeout.
The identification of this attack is done by the trace shown in figure 13b. Unlike other examples presented, this attack is identified by observing unsuccessful occurrences of the trace. This information is stored at the APM MIB, as shown in table 2.

6 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 [21]) 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).

Regarding flexibility, the proposal of the PTSL language is one of the most important contributions of this work. All traffic management 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, SMTP, etc.) 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. This is an advantage of our approach as compared to other traffic management related works (see section 2), but not a novelty for most application management suites.

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.

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 [14], where the Jasmin implementation has been evaluated.
References


