Towards a Programmable Agent-based Distributed Architecture for Enterprise Application and Service Management

Luciano P. Gaspary, Luis F. Balbinot, Roberto Storch, Fabrício Wendt, Liane R. Tarouco
Federal University of Rio Grande do Sul
Institute of Informatics
Av. Bento Gonçalves, 9500 – Agronomia – CEP 91591-970 – Porto Alegre, Brazil

Abstract—**The popularization of the electronic commerce and the growing use of this business modality by companies as well as the growth in the number of applications, protocols and services that come to be executed in computer networks generate difficulties for management tools. Most of these tools are able to monitor a previously established set of protocols; monitoring of new protocols becomes possible with the update of firmware or through development kits that are hard to be assimilated by network managers. Additionally, these tools usually have little or no ability to act automatically when facing unexpected behaviors from these protocols. This paper presents an architecture for distributed management of enterprise networked applications, high-layer protocols and network services based on programmable agents. By means of a high-level graphical and textual language, the network manager is able to specify protocol traces in order to do FCAPS (Fault, Configuration, Accounting, Performance and Security) management. The observation of these traces in the network traffic leads to actions, which are also determined by the network manager. This paper describes the language used to specify traces, the architecture and some examples of application that validate the proposal.**

Keywords—**High-layer protocol management, Service management, Programmable agents.**

I. Introduction

New protocols, applications and network services are constantly being incorporated into our daily routines and executed in computer networks. The electronic commerce popularization and the growing use of this business modality by companies, for example, implies using the network to send crucial data related to the organization and its customers. The applications and protocols used to make this business feasible are critical and, therefore, need to be carefully monitored and managed.

Not only critical business applications require special attention. New protocols are often made available to the market to serve a growing number of specific functionalities and are readily adopted by network users. As a result of this fast spread, protocols that have not been thoroughly tested yet, or even faulty protocols, are disseminated to the network consuming community. According to Ho et al. [1] in many cases these anomalies are the cause of the decrease in the performance of the network and end up going unnoticed. Additionally, due to the differences among versions, entities (processes that implement a given protocol) have difficulties to establish and retain communication among themselves.

We believe that most of the research done so far has tried to provide mechanisms that ensure greater availability and better performance of the network (e.g. Hood and Ji work on proactive fault detection [2]). While the solutions to manage its physical infrastructure are well established and tested, there is still the need to investigate ways of providing an effective management of protocols, applications and services that flow over this infrastructure.

The existing management tools are not fully prepared to enable the monitoring of these new applications and protocols. Most of them only allow the monitoring of a monolithic set of protocols. The ability to observe new protocols depends on the updating of the firmware of the monitoring hardware or on the programming in low level languages like the extensible probe architecture proposed by Malan and Jahanian [3]. Due to the complexity of the task, most network managers do not take this possibility into account.

Other solutions such as Tivoli Enterprise [4] are intrusive because they require the applications, when being developed, to make special calls to monitoring procedures. This approach is suitable for the monitoring of in-house developed applications, but cannot be used to manage protocols from proprietary applications (e.g. web browsers and servers, e-mail clients and servers). Additionally, there is the need to invest in the training of developers for the use of these monitoring APIs.

Regarding monitoring, some approaches such as the IETF RMON2 MIB (Remote Network Monitoring Management Information Base version 2) store the number of packets sent/received (by a host or exchanged by host pairs) of a predefined set of high-layer protocols supported by the probe [5]. Gaspary et al. describe in [6], [7] the advantages and limitations of the RMON2 MIB. One of RMON2 weaknesses is that it does not store any information related to performance, but it has been discussed by the Remote Network Monitoring group at the IETF [8]. Furthermore, we should point out that many management tools are limited to monitoring and the network manager has to take actions manually when unexpected behaviors from these protocols are observed.

In this paper we present an architecture for distributed management of enterprise networked applications, high-layer protocols and network services. Through a graphical and textual language based on finite state machines, the network manager defines protocol traces to be observed. These specifications are readily received by one or more programmable agents that immediately start to check whether a defined trace occurs or not. The observation of these traces in the network traffic triggers actions, which are also determined by the network manager.

The paper is organized as follows: section 2 describes the language to specify protocol traces. In section 3 the architecture is presented. Some application examples that describe how this architecture could be used to fulfil fault, accounting, performance and security management are presented in section 4. In section 5 we present a summary and concluding remarks.

II. Graphical and Textual Language for Protocol Trace Representation

In this section we propose a graphical and textual language for the representation of high-layer protocol traces. The languages

are not equal. The textual one makes the complete representation of a trace possible, including the specification of both the state machine and the events that trigger the transitions. On the other hand, using the graphical language one can graphically represent the state machine and only label the events that trigger the transitions.

A. Organization of a Specification

The specification of a trace begins with the keyword `Trace` and ends with `EndTrace`. Initially, the manager may describe some optional items to the specification (see figure 1, lines 2–7). Next, it is broken down into three sections: `MessagesSection` (lines 8–10), `GroupsSection` (lines 11–13) and `StatesSection` (lines 14–16), where messages to be observed, grouping and state machines that describe the trace are respectively specified.

```
1  Trace "Attempt to access non-authorized Web page"
2  Version: 1.0
3  Description: Accounting of 403 message in response to GET.
4  Key: HTTP, 403, forbidden access
5  Port: 80
6  Owner: Luciano Paschoal Gaspary
7  Last Update: Tue, 15 Aug 2000 21:51:02 GMT
8  MessagesSection
9  ...
10 EndMessagesSection
11 GroupsSection
12 ...
13 EndGroupsSection
14 StatesSection
15 ...
16 EndStatesSection
17 EndTrace
```

Fig. 1. Schematic representation of a textual specification.

If the trace to be monitored belongs to a single application-layer protocol then the network manager may specify the TCP or UDP port number using the `Port` parameter (line 5). It will simplify packet classification during the monitoring phase.

B. State Machines

The trace of a protocol is defined through a finite state machine. The network manager may define a model to monitor just a part of or the whole protocol, or interactions that comprehend more than one protocol. Figure 2 shows two trace examples. In the first example (a), the manager is interested in monitoring the attempts to access non-authorized web pages. The trace shown in (b) does not describe a single protocol; it is rather 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 on, but the *named* daemon is not running.

As one can see states are represented by circles. The initial state has the label `idle` associated to it. The final state is represented by two concentric circles. In both examples the initial and final states are the same (`idle`). Transitions are represented by unidirectional arrows. A continuous arrow indicates that the transition is triggered by the client host, whereas a dotted arrow

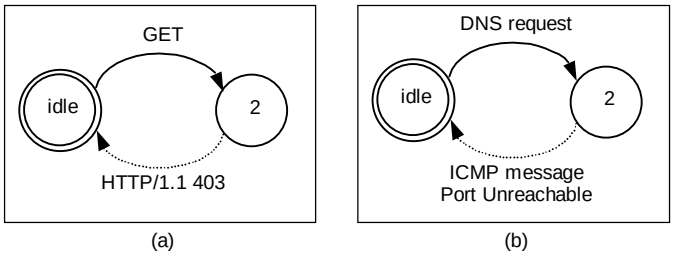

Fig. 2. Graphical representation of a trace. (a) Attempt to access a non-authorized web page. (b) DNS request not replied because *named* daemon is not executing.

denotes that it is caused by the server host. The text associated to a transition only labels the event (specified as a message or grouping in the textual language) that will trigger it. It means that the whole specification of a transition only can be done using the textual language.

The graphical representation shown in figure 2 can be mapped to the textual specification presented in figure 3.

```
FinalState: idle
State idle
"GET" GotoState 2
EndState idle

State 2
"HTTP/1.1 403" GotoState idle
EndState 2
```

(a)

```
FinalState: idle
State idle
"DNS request" GotoState 2
EndState idle

State 2
"ICMP message" GotoState idle
EndState 2
```

(b)

Fig. 3. Textual representation of state machines.

C. Transitions

In addition to making a high-level representation of traces, it is necessary to describe what causes the change of states. Before describing the adopted solution, it is important to highlight that high-layer protocols are specified in many different ways. Larmouth classifies them as character or binary-based [9]. Character-based protocols are defined as a set of text lines coded in ASCII (e.g. HTTP and SMTP). Binary protocols, on the other hand, are defined as strings of octets or bits (e.g. TCP).

Considering the differences between both protocol types, we propose state transitions to be represented by a positional approach. Taking the example shown in figure 2a, we present (see figure 4) how to represent the transition `HTTP/1.1 403`.

```
1 Message "HTTP/1.1 403"
2 MessageType: server
3 // OffsetType Encapsulation FieldNumber Verb Description
4 FieldCounter Ethernet/IP/TCP 0 HTTP/1.1 "Protocol version"
5 FieldCounter Ethernet/IP/TCP 1 403 "Access forbidden"
6 EndMessage
```

Fig. 4. Representation of character-based protocol fields.

As the transition is expected to be triggered by the server host, one must set the `MessageType` field to `server` (line 2). Since both protocol fields (`HTTP/1.1` and `403`) belong to a character-based protocol, the search for their positions within

the packets is made by fields (`FieldCounter`, lines 4–5). In this example, `HTTP/1.1` is the first string that appears on the message and therefore its offset is 0 (third parameter in line 4). The second string to appear is `403` and its offset is 1 (line 5). For each protocol field defined in a message it is also necessary to inform where to look for it (encapsulation `Ethernet/IP/TCP`, lines 4–5).

When the transition is caused by a binary protocol, the offset is presented in bits (`BitCounter`). In this case, it is necessary to inform where the field starts (`FirstBit`) and the number of bits to be observed from this offset on (`NumberOfBits`). Figure 5 shows the initial part of the DNS protocol header. A standard DNS request can be recognized by two fields: `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 far from the beginning of the header and its size is 1 bit. Field `OPCODE` starts in the seventeenth bit and occupies 4 bits.

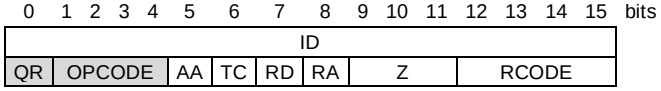

Fig. 5. Beginning of the DNS protocol header.

In figure 6 the textual representation of a standard DNS request is shown.

```
Message “DNS request”
MessageType: client
// OffsetType Encapsulation FirstBit NumberOfBits Verb Description
BitCounter Ethernet/IP/UDP 16 1 1 “field QR”
BitCounter Ethernet/IP/UDP 17 4 0000 “field OPCODE”
EndMessage
```

Fig. 6. Representation of binary protocol fields.

It is possible to group one or more messages into one single transition. For example, in figure 2a it would be possible to change the `HTTP/1.1 403` with the grouping `HTTP/1.1 4XX`. In this case the trace would monitor the rate of all WWW accesses resulting in error generated by client requests (`4XX`) instead of only observing the occurrence of forbidden access attempts. Figure 7 shows the representation of this grouping (lines 21–23).

```
1  MessagesSection
2  …
3  Message “HTTP/1.1 400”
4  MessageType: server
5  FieldCounter Ethernet/IP/TCP 0 HTTP/1.1 “Protocol Version”
6  FieldCounter Ethernet/IP/TCP 1 400 “Syntax error”
7  EndMessage
8  Message “HTTP/1.1 401”
9  MessageType: server
10 FieldCounter Ethernet/IP/TCP 0 HTTP/1.1 “Protocol Version”
11 FieldCounter Ethernet/IP/TCP 1 401 “Need user authentication”
12 EndMessage
13 Message “HTTP/1.1 403”
14 MessageType: server
15 FieldCounter Ethernet/IP/TCP 0 HTTP/1.1 “Protocol Version”
16 FieldCounter Ethernet/IP/TCP 1 403 “Forbidden access”   (cont.)
17 EndMessage                                              (cont.)
18 …
19 EndMessagesSection
20 GroupsSection
21 Group “HTTP/1.1 4XX”
22 Messages: “HTTP/1.1 400”, “HTTP/1.1 401”, “HTTP/1.1 403”, …
23 EndGroup
24 EndGroupsSection
25 StatesSection
26 FinalState: idle
27 State idle
28 “GET” GotoState 2
29 EndState idle
30 State 2
31 “HTTP/1.1 4XX” GotoState idle
32 EndState 2
33 EndStatesSection
```

Fig. 7. Representation of message grouping.

In some cases the network manager may be interested in observing the occurrence of a certain string within the data field of a certain protocol, no matter where it is located. To do that, in the definition of such a message one must use `NoOffset` as the `OffsetType` parameter. This feature is interesting, for instance, to observe the attempt of an intrusion. The example presented in figure 8 defines that every TCP packet must be tested for the occurrence of the string `/etc/passwd` (line 4).

```
1 Message “/etc/passwd”
2 MessageType: client
3 // OffsetType Encapsulation Verb
4 NoOffset Ethernet/IP/TCP /etc/passwd
5 EndMessage
```

Fig. 8. Non-specified offset message field.

We have also created a mechanism to allow the determination of a timeout to a transition to occur. To do that one must associate a timeout value (in milliseconds) to the message definition (see figure 9, line 3). When not defined, a default value is used by the network monitor.

```
1 Message …
2 MessageType: …
3 MessageTimeout: 5000
4 …
5 EndMessage
```

Fig. 9. Textual representation of timeouts.

III. The Programmable Agent-based Distributed Architecture

The architecture we propose is an extension of the existing distributed management infrastructure standardized by the IETF [10] with high-layer protocol and network service management capabilities. Figure 10 shows the main components of the architecture. It is composed of management stations, mid-level

managers, programmable monitoring agents and programmable action agents.

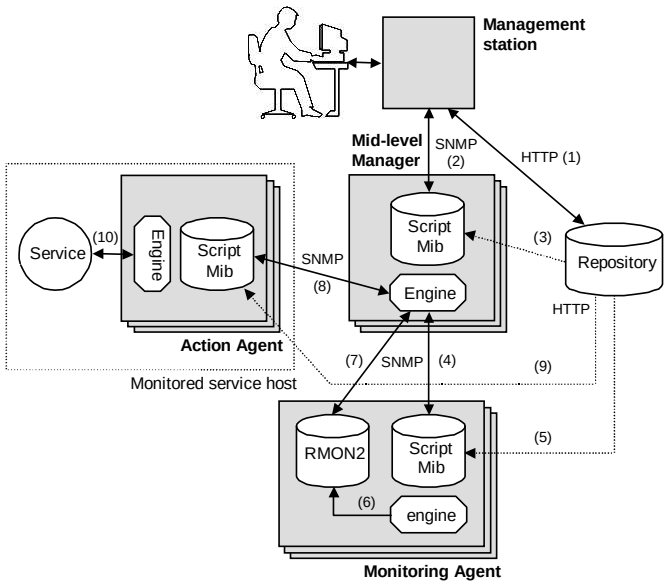

Fig. 10. Components of the architecture.

Based on the Script MIB [11], the architecture provides mechanisms for the delegation of management tasks to mid-level managers, which interact with monitoring and action agents to have them executed. Specifications of protocol traces according to the language presented in the previous section are used by mid-level managers to program the monitoring agents. Once programmed these agents start to monitor the occurrence of the traces. The information obtained is analyzed by the mid-level managers, which may ask action agents for the execution of procedures, resulting in the automation of several management tasks. The following sub-sections describe the components of the architecture and their interactions with each other.

A. Management Station

The most important activities accomplished by the network manager from the management station are (a) specification of protocol traces and actions, (b) specification, delegation, observation and interruption of management tasks and (c) receipt and visualization of traps.

As the whole architecture is based on the Script MIB, protocol traces, actions and management tasks are scripts executed by monitoring agents, action agents and mid-level managers, respectively. Protocol traces are specified by the network manager using the language previously presented. Actions are scripts developed using Java or any scripting language such as TCL and Perl. Management tasks may also be implemented using any language and coordinate monitoring and action agents. Such a script programs the monitoring agents, observes the occurrence of the trace and activates action agents when a condition associated to a protocol trace holds. The same script may also report events to the management station raising traps.

At the management station the network manager can specify traces using a graphical tool (see figure 11) or, if he knows the language, by editing a text file. The same occurs with actions and management tasks. The specification of protocol traces, actions and management tasks are stored in the repository via the HTTP protocol (figure 10, see flow (1) in diagram).

Communication between the management station and the mid-level managers takes place using the SNMP protocol (Script MIB) (2). The manager can delegate a management task to a mid-level manager as well as abort it at any time. Intermediate and final results of the execution of a management task are stored directly at the Script MIB of the mid-level manager responsible for the task and can be retrieved by the management station using the SNMP protocol. The management station may also receive SNMP traps from the mid-level managers.

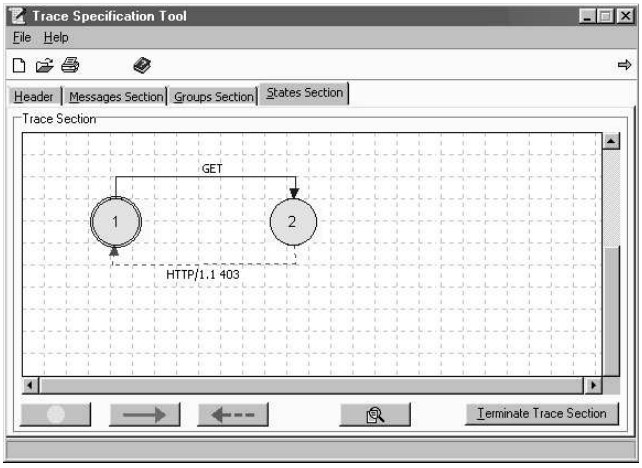

Fig. 11. Prototype of the tool for trace specifi cation.

B. Mid-level Manager

Mid-level managers execute and monitor management tasks delegated by the management station and report the most important events to it. The number of mid-level managers is determined by the network manager and depends on the size and complexity of the infrastructure to be managed.

The process of configuring mid-level managers is the following: the network manager defines a management task and stores it at the repository (1). Next, the activation of the task must be scheduled using the Script MIB (2). In order to do that, the mid-level manager has to be informed about the location of the task (script). When activated, the task is retrieved from the repository using the HTTP protocol (3).

The script executed by the mid-level manager installs the protocol trace (4) and the action script (8), requests the monitoring agent to start observing the occurrence of the protocol trace just installed (4), polls RMON2 variables periodically to monitor the occurrence of the trace (7) and, depending on what is observed, dispatches the execution of the action script (8) or raises a trap to the manager (2). The script communicates with the agents using the SNMP protocol.

C. Monitoring Agent

The monitoring agents are responsible for observing the traffic on the network segment where they are installed. They are configured by mid-level managers and are called programmable

because they are able to monitor protocol traces delegated dynamically by the network manager. This flexibility is obtained through the language presented in section II. When the mid-level manager sets the monitoring agent up (4), the former defines which protocol trace it should retrieve (it is indicated within the script that implements the task). Once retrieved the trace file is loaded by the monitoring engine (5) and the observation starts.

Whenever the occurrence of the trace is observed between any pair of hosts, information is stored within an RMON2-like MIB (6). This MIB is different from the standard because the `protocolDir` group is writable in our approach. Therefore the probe stores statistics according to the protocol traces of interest to the network manager. Additionally, the granularity of the monitoring becomes greater. Instead of storing overall statistics on traffic generated by a given protocol, statistics are generated according to the occurrence of specified traces or transactions.

The `alMatrix` group from RMON2 MIB stores statistics on the trace when the latter is observed between every pair of hosts. Table I shows the contents of the `alMatrixSD` table. It gathers the observed number of packets and octets exchanged between every pair of hosts (client/server) using the protocol traces being monitored by the probe. In the example, two traces were observed: `Attempt to access non-authorized web page` and `DNS service monitoring` (previously shown in figure 2a and b).

TABLE I

INFORMATION OBTAINED BY REFERRING TO THE ALMATRIXSD TABLE.

SourceAddress	DestinationAddress	Protocol	Pkts	Octets
17.16.10.1	17.16.10.2	Attempt...	254	120.212
17.16.10.6	20.24.20.2	Attempt...	20	10.543
17.16.10.1	17.16.10.33	DNS Serv...	4	4.350
17.16.10.32	17.16.10.33	DNS Serv...	8	7.300

The disadvantage of using RMON2 MIB is that it does not have objects capable of storing information related to performance. For this reason, our group is currently considering the possibility of using, in addition to that, an RMON2 extension, such as Application Performance Measurement MIB [8]. Table II shows the kind of information stored by this MIB. The first line shows that the trace `Successful WWW access` has been observed 127 times between hosts 17.16.10.12 and 17.16.10.2. Additionally, the mean response time was 6 seconds.

TABLE II

MIB WITH INFORMATION ON PERFORMANCE.

Client	Server	Protocol	Suc.	Unsuc.	Resp.
17.16.10.12	17.16.10.2	Suc.WWW	127	232	6 sec.
17.16.10.12	20.24.20.2	Suc.WWW	232	112	17 sec.
10.10.135.25	17.16.10.2	SYN Flood	1.023	56	3 sec.

D. Action Agent

Action agents reside in hosts where network services are executed. Their function is to perform a given operation on these services. Let us take as example the DNS service. Figure 2b shows a trace that enables to detect when the *named* daemon is not in execution. There may be a management task, configured in the mid-level manager, that monitors the occurrence of this trace. If that is the case, the action to be taken is contacting the action agent (see flow (8) in figure 10), which is located in the host where the DNS service is installed, and request the execution of a script to restart the daemon (9, 10) (see the example of such a script developed in Perl in figure 12). The result obtained by the action agent is accessible to the mid-level manager (8), which may send it to the manager for notification purposes (2).

```
#!/usr/bin/perl

my $pid;

# Verify if the process named is executing.
if (-e "/var/run/named.pid") {
   $pid = `/bin/cat /var/run/named.pid`;
}

# If named is running, restart it using a HUP signal, otherwise
# instantiate the process again.
if (defined $pid) {
   print "Restarting named (sending HUP signal)...\n";
   `/bin/kill -HUP $pid`;
} else {
   print "Starting named (was not running)...\n";
   `/usr/sbin/named &`;
}

# Test if the process is executing.
if (-e "/var/run/named.pid") {
   $pid = `/bin/cat /var/run/named.pid`;
   print "The named daemon is up and running as PID $pid\n";
} else {
   print "The named daemon could not be started!\n";
}
```

Fig. 12. Perl script to restart the *named* daemon.

IV. EVALUATION OF THE ARCHITECTURE: EXAMPLES OF APPLICATION

The proposed architecture was designed to take into account all the standard functional areas of management (FCAPS). Our research group has explored its characteristics to validate its usefulness for the management of enterprise applications, high-layer protocols and network services. In the following subsections we present examples of how the proposed architecture could be used in fault, accounting, performance and security management.

A. Fault Management

Fault management is an important target of the proposed architecture. An example of fault management concerning to high-layer protocols and network services is checking the availability of a network service and restart it if it is not running. Figure 13 shows how this can be achieved using the architecture. In this case the task delegated to the mid-level manager is supposed to monitor the DNS service.

This monitoring is performed by sniffing the packets seen in the segment. In a situation where the daemon responsible for the DNS service is not running, the agent will observe a DNS re-

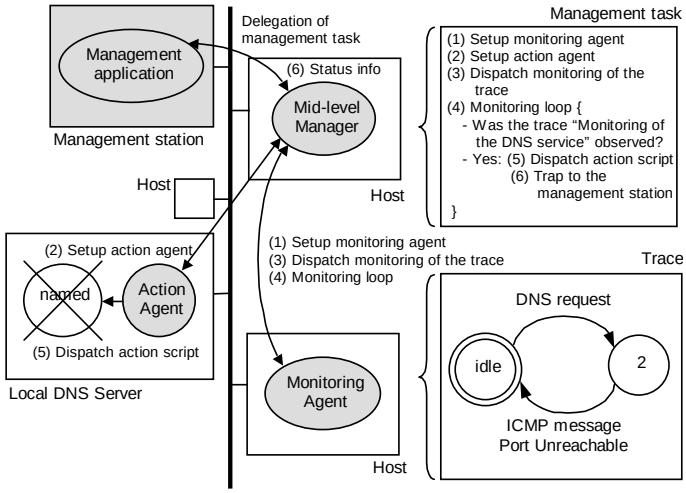

Fig. 13. Execution steps of a management task.

quest and some time later a `Port Unreachable` ICMP message from the serving host. In this case the mid-level manager should contact an action agent, which resides in the DNS serving host and request the execution of a script to restart the daemon (such as the one illustrated in figure 12). The textual specification of the trace `Monitoring of the DNS service` is shown in figure 14.

```
Trace "DNS service monitoring"

Version: 1.0
Description: Trace to detect  when named is not running
Key: named, fault, DNS service
Port:
Owner: Luciano Paschoal Gaspary
Last Update: Tue, 10 Aug 2000 15:30:58 GMT

MessagesSection

Message "DNS Request"
// See code in figure 6.
EndMessage

Message "ICMP Message"
MessageType: server
// OffsetType Encapsulation FirstBit NumberOfBits Verb Description
BitCounter Ethernet/IP 0 8 00000011 "Type field=00000011?"
BitCounter Ethernet/IP 8 8 00000011 "Code field=00000011?"
EndMessage

EndMessagesSection

StatesSection

// See code in figure 3b.

EndStatesSection

EndTrace
```

Fig. 14. Trace to monitor the DNS service.

Although protocol and networked application debugging is a subject not directly related to fault management, we point out that a network manager can specify a complete protocol or part of it and check whether or not the network traffic matches the specification. This possibility is useful not only to test third-party application, but also for specific applications of the organization. This topic however is out of the scope of the paper.

B. Accounting Management

Network users usually see the network as an inexhaustible source. They end up adding more and more applications and protocols to their daily routines which, for the administrator, generates the need of constant changes in the network infrastructure [6], [7]. These changes involve costs and, therefore, need to be justified. This is possible if the administrator is able to answer simple questions such as: which users or departments are using the network? When is it being more used? What applications are being executed? What are the activities of a given user? Are users perceiving a proper level of service? Are acquired resources being correctly allocated?

The answers to these questions may be obtained with the use of accounting mechanisms. RMON2 is a good alternative to monitor widely used protocols, but is not flexible enough when one wants to monitor/account protocols specific to the organization. The proposed architecture solves this problem, since it can work with dynamically-defined protocol specifications.

C. Performance Management

Like accounting management, performance management is taken care of by the architecture. For a given trace, it is possible to check how many times it occurred, the number of unsuccessful occurrences, average response time of traces completed successfully in a given time interval.

With this kind of information, the network manager can determine where are the bottlenecks in his network and what are the applications or traces that present more problems (with long response time or timeouts).

D. Security Management

Through the graphical/textual language presented in section II, it is possible to define traces whose observation in the network may represent intrusion attempts. As an example, we have modelled the trace for the detection of port scanning and SYN Flood.

D.1 Port Scanning

TCP port scanning is done by sending TCP Connect (`SYN`) packets to all ports of a network host in order to find out what are the TCP services it provides. If the host does not provide the service to a given port, it sends a TCP return packet with the `RST` bit on in response to the connection attempt. Figure 15 shows the textual specification of the trace `Port scanning`.

The trace specification is complemented by a management task that defines how many times the occurrence of this trace is allowed within a certain time interval. When the number of occurrences exceeds the specified limit, the mid-level manager notifies the management station of a possible intrusion attempt.

D.2 SYN Flood

The SYN Flood attack is accomplished by sending a large number of connection opening packets (packets with the `SYN` flag on) with a false origin address for a given target host. This false origin address should either be unreachable or the address of a non-existing host (one of the reserved addresses is often used).

```
Trace “Port scanning”

Version: 1.0
Descripttion: Trace based on TCP Connect and on TCP RST.
Key: TCP, scanning, SYN, RST
Port:
Owner: Luciano Paschoal Gaspary
Last Update: Tue, 16 Aug 2000 15:30:58 GMT

MessagesSection

Message “TCP SYN”
MessageType: client
// OffsetType Encapsulation FirstBit NumberOfBits Verb
BitCounter Ethernet/IP 110 1 1 “SYN field – 1 means TCP Connect”
EndMessage

Message “TCP RST”
MessageType: server
// OffsetType Encapsulation FirstBit NumberOfBits Verb
BitCounter Ethernet/IP 109 1 1 “RST field”
EndMessage

EndMessagesSection

StatesSection

FinalState: idle
State idle
“TCP SYN” GotoState 2
EndState idle

State 2
“TCP RST” GotoState idle
EndState 2

EndStatesSection

EndTrace
```

Fig. 15. Trace to monitor the occurrence of port scanning.

When the target host receives these connection opening packets (SYN), it creates an input in the connection queue and sends a response packet (SYN/ACK) to the address that requested the connection. After sending the response packet, the target host waits for a confirmation from the connection requester. As the origin address of the packets has been made up, the target host will never receive this connection confirmation (see figure 16). In a given moment, the connection queue of the target host becomes full and from then on all connection opening requests are denied and the service becomes unavailable. This unavailability lasts some seconds, because when the target host finds out that confirmation is taking too long, it removes the open connection from the list.

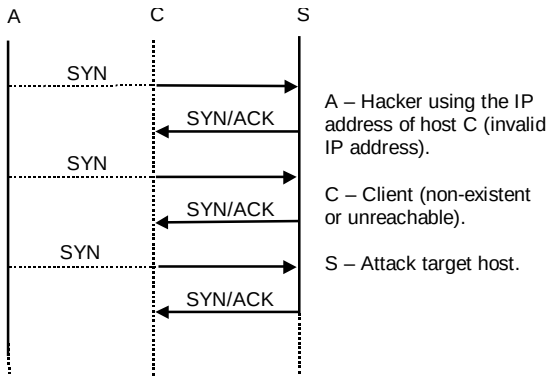

Fig. 16. The SYN Flood attack.

The trace to detect this attack is illustrated in figure 17. In this case the detection will be done by observing many unsuccessful occurrences of the trace (see third row of table II). Therefore, the management task defines, for a given time interval, the number of unsuccessful traces required for the mid-level manager to trigger the execution of the associated action. If the connection attempts origin from the same IP address the action could be to contact the action agent located in the host where the firewall resides so that it runs a small script to perform its reconfiguration in order to block TCP accesses from the host that generated the attacks.

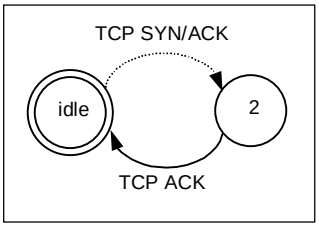

Fig. 17. Trace to detect the `SYN Flood` attack.

V. Conclusions

This work presented a distributed architecture for the management of enterprise applications, high-layer protocols and network services based on the usage of programmable agents. Motivated by the growing need companies have to monitor their high-layer protocols and, particularly, their critical applications, the work proposes a flexible architecture able to follow the fast dissemination of protocols and applications on networks (that need to be managed). The architecture may be applied either in corporate networks or in application service providers.

The most important contribution of the architecture is the granularity of the monitoring. The observation of network traffic on a transaction basis makes the understanding of protocol and networked application behaviors possible. The language proposed to specify protocol traces is simple, but the network manager has to know the format of the packets exchanged by the application or protocol to be managed. The availability of graphical tools with high level interfaces such as the prototype shown in figure 11 makes this task easier.

Another significant contribution of the architecture is the possibility to do more than just monitoring. Management tasks provide the manager with mechanisms to monitor the occurrence of protocol traces and to dispatch management scripts executed by programmable action agents. These mechanisms contribute to management automation in some scenarios.

As described by Strauß [12], “distributed management in general [13], [14] and the Script MIB specifically are expected to bring various advantages over the centralized concept suited for the raising demands in network management. A commonly mentioned advantage is the increased scalability due to the delegation of management tasks from the centralized network management station to mid-level managers. This implies that CPU and network load is also delegated to the subnets to which the mid-level managers belong. Another major advantage is concerned with the robustness of management tasks. While

centralized management systems require a reliable network, the distributed approach allows to delegate some sensible manager functions next to the observed agents. Hence these functions may become independent from less reliable WAN links, for example".

The architecture requires more work to be controlled than a single centralized management system. The management of its components becomes more complicated. It is necessary to distribute and update scripts, control running scripts, gather and correlate intermediate and final results. We believe that such operations, as well as the specification of traces, can be simplified by adding an easy-to-use interface to the management application. Currently, our research group is working on the improvement of the prototype. After that, a larger scale validation will be done.

REFERENCES

[1] L. L. Ho, C. J. Macey, and R. Hiller, "A Distributed and Reliable Platform for Adaptive Anomaly Detection in IP Networks", in *Proc. 10th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, Zurich, October 1999, pp. 33–46.

[2] C. S. Hood and C. Ji, "Intelligent Agents for Proactive Fault Detection", *IEEE Internet Computing*, vol. 2, no. 2, pp. 65–72, March/April 1998.

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

[4] C. Cook et al., "An Introduction to Tivoli Enterprise", First edition. USA: International Technical Support Organization, 1999. Available at `http://www.redbooks.ibm.com`.

[5] S. Waldbusser, "Remote Network Monitoring Management Information Base Version 2 using SMIv2". Request for Comments 2021, January 1997.

[6] L. P. Gaspary and L. R. Tarouco, "Characterization and Measurements of Enterprise Network Traffic with RMON2", in *Proc. 10th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, Zurich, October 1999, pp. 229–242.

[7] L. P. Gaspary and L. R. Tarouco, "Managing Users, Applications and Resources with RMON2", in *Proc. Global Telecommunications Conference, Symposium on Enterprise Applications and Services*, Rio de Janeiro, December 1999, pp. 1997–2001.

[8] S. Waldbusser, "Application Performance Measurement MIB", Internet Draft, November 2000.

[9] J. Larmouth, *ASN.1 Complete*, Open Systems Solutions, 1999.

[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", Internet Draft, Nortel Networks, TU Braunschweig, July 2000.

[12] F. Strauß, "Advantages and Disadvantages of the Script MIB Infrastructure", Project Report, October 2000. Available at `http://www.ibr.cs.tu-bs.de/projects/jasmin/`.

[13] Y. Yemini, G. Goldszmidt, and S. Yemini, "Network Management by Delegation", in *Proc. International Symposium on Integrated Network Management*, 1991, pp. 95–107.

[14] J. Schönwälder, "Network Management by Delegation – From Research Prototypes Towards Standards". *Computer Networks and ISDN Systems*, 29(15):1843– 1852, November 1997.