Uma Arquitetura para Gerenciamento Distribuído e Flexível de Protocolos de Alto Nível e Serviços de Rede

Luciano Gaspary, Luis F. Balbinot, Roberto Storch, Fabrício Wendt, Liane Tarouco

Universidade Federal do Rio Grande do Sul
Instituto de Informática
Av. Bento Gonçalves, 9500 - Agronomia - CEP 91.591-970 - Porto Alegre, Brasil
{paschoal,hades}@inf.ufrgs.br, roberto@unisc.br, fabricio.wendt@bol.com.br, liane@penta.ufrgs.br

Resumo

Este artigo propõe uma arquitetura para gerenciamento distribuído de protocolos de alto nível e serviços de rede. Baseada na MIB Script do IETF, a arquitetura **Trace** oferece mecanismos para a delegação de tarefas de gerenciamento a gerentes intermediários, que interagem com agentes de monitoração e agentes de ação para executá-las. O artigo introduz PTSL (*Protocol Trace Specification Language*), uma linguagem gráfica/textual criada para permitir que gerentes de rede especifiquem traços de protocolos. As especificações são usadas pelos gerentes intermediários para programar os agentes de monitoração. Uma vez programados, esses agentes passam a monitorar a ocorrência dos traços. As informações obtidas são analisadas pelos gerentes intermediários, que podem requisitar a agentes de ação a execução de procedimentos (ex: *scripts* Perl), possibilitando a automação de diversas tarefas de gerenciamento.

Abstract

This paper proposes an architecture for distributed management of high-layer protocols and network services. Based on the IETF Script MIB, the **Trace** architecture provides mechanisms for the delegation of management tasks to mid-level managers, which interact with monitoring and action agents to have them executed. The paper introduces PTSL (*Protocol Trace Specification Language*), a graphical/textual language created to allow network managers to specify protocol traces. The specifications 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 (e.g. Perl scripts), making the automation of several management tasks possible.

Palavras-chave: gerenciamento de redes de computadores, monitoração, Internet.

1 Introdução

A utilização das redes de computadores como suporte para um crescente número de negócios e aplicações críticas tem estimulado a busca de soluções de gerenciamento que permitam manter em funcionamento não apenas a infra-estrutura física da rede, mas também os protocolos e serviços que a compõem. A popularização do comércio eletrônico e a crescente adoção dessa modalidade de negócio pelas empresas, por exemplo, implica em passar a usar a rede para trafegar dados vitais da organização e dos clientes. Os protocolos e serviços que suportam essas aplicações são críticos e, portanto, precisam ser cuidadosamente monitorados e gerenciados.

Não somente as aplicações críticas requerem atenção especial. Freqüentemente são lançados no mercado novos protocolos para atender a um conjunto crescente de funcionalidades específicas, que são prontamente adotados pelos usuários da rede. Como conseqüência dessa

rápida proliferação, protocolos pouco testados ou até mesmo com falhas são disseminados no mercado. Em muitos casos essas anomalias, bem como o uso não calculado dos recursos, são a causa da degradação do desempenho da rede e acabam passando despercebidas.

Para que o gerente de rede possa garantir alta disponibilidade e eficiência da rede, é preciso um ambiente de gerenciamento flexível, que possa ser rápida e facilmente adaptado para monitorar cenários cada vez mais dinâmicos. Além de flexível, o aumento crescente de tamanho das redes requer que esse ambiente de gerenciamento seja distribuído para que a solução seja eficiente e escalável.

Este artigo apresenta uma arquitetura para gerenciamento distribuído e flexível de protocolos de alto nível e serviços de rede baseada em agentes programáveis e está organizado da seguinte forma: a seção 2 descreve e compara as iniciativas expressivas de pesquisa sobre gerenciamento de protocolos de alto nível e gerenciamento distribuído. A seção 3 apresenta PTSL, uma linguagem gráfica/textual para especificação de traços de protocolos. A seção 4 introduz a arquitetura **Trace** e seus componentes. Na seção 5 são apresentados alguns estudos de casos propostos para validar a arquitetura. A seção 6 encerra o artigo apresentando uma avaliação da arquitetura, as conclusões e os trabalhos a serem ainda realizados.

2 Trabalhos Relacionados

Muitas abordagens têm sido propostas tanto no que se refere a gerenciamento de protocolos de alto nível quanto a gerenciamento distribuído. O principal tópico trabalhado em pesquisas sobre gerenciamento de protocolos de alto nível é *monitoração*. A ferramenta `ntop` [1] destina-se à monitoração e medição de tráfego, incluindo funções que viabilizam sua caracterização e o cálculo de utilização da rede por protocolo. Tarouco, Carrilho e Silveira descrevem em [2, 3, 4] MIBs para a monitoração de protocolos. Tarouco apresenta uma proposta para gerenciamento de correio eletrônico. Carrilho introduz um modelo para o gerenciamento do protocolo FTP. Silveira apresenta uma MIB genérica para a monitoração de alguns protocolos de alto nível. Essas três propostas surgiram quando ainda não havia nenhuma iniciativa padronizada nesse sentido. O padrão RMON2 (*Remote Networking Monitoring Management Information Base Version 2*) [5], definido em 1997, agrega grande parte das funcionalidades previstas nos projetos recém citados.

Outros trabalhos recentes ligados à monitoração são a arquitetura extensível proposta por Malan e Jahanian [6] e a arquitetura RTFM (*Realtime Traffic Flow Measurement*), desenvolvida pelo grupo de trabalho de nome homônimo no IETF [7] e implementada pela ferramenta `NeTraMet` [8]. O princípio básico do funcionamento dessa arquitetura está baseado em agentes distribuídos (nela chamados de *meters*) que implementam a MIB RTFM *Meter* [9]. Esses agentes são capazes de realizar a medição e a contabilização de fluxos de pacotes em tempo real. A MIB permite que um gerente SNMP consulte as informações estatísticas, bem como configure o agente. A especificação de fluxos é feita através de um conjunto de regras especificadas por uma linguagem denominada SRL [10] e determina (a) quais fluxos devem ser contabilizados, (b) que nodos devem ser considerados origens de fluxos e (c) qual o nível de detalhamento desejado para cada fluxo.

Uma exigência provocada pela rápida proliferação de protocolos e aplicações que são executadas sobre as redes de computadores é que as ferramentas destinadas à monitoração sejam *flexíveis*. Boa parte das ferramentas existentes não estão completamente preparadas para permitir a monitoração de novos protocolos e aplicações e operam com base em um conjunto pré-determinado deles. A capacidade para poder observar novos protocolos depende de atualização do *firmware* do equipamento de monitoração, como os probes RMON2, ou da programação em linguagens de baixo nível como a arquitetura proposta por Malan e Jahanian e a ferramenta `ntop`. Nesse último caso, grande parte dos gerentes de rede acaba ignorando a possibilidade de personalizar o que deva ser monitorado devido à complexidade da tarefa.

Outras soluções, como o Tivoli Enterprise [11], são intrusivas, pois requerem que as aplicações, ao serem desenvolvidas, façam chamadas especiais a procedimentos de monitoração. Essa abordagem é aplicável para a monitoração das aplicações desenvolvidas na própria empresa, mas não pode ser usada no gerenciamento de protocolos oriundos de aplicações proprietárias (ex: navegadores e servidores Web, clientes e servidores de e-mail). Além disso, é preciso investir na capacitação dos desenvolvedores para a utilização das APIs de monitoração.

O *tipo de informação* coletada e a *granularidade* dessas informações são aspectos importantes associados à monitoração. A MIB RMON2 e a ferramenta `ntop` coletam estatísticas como o número de pacotes enviados/recebidos por uma estação ou o número de pacotes trocados entre duas estações, que são separadas de acordo com o protocolo utilizado. As potencialidades e limitações de RMON2 foram apresentadas por Gaspary et al. em [12]. Uma das fraquezas de ambas as abordagens é a falta de informações relacionadas a desempenho e falhas. Essas dificuldades têm sido discutidas pelo grupo RMON do IETF através da MIB APM (*Application Performance Measurement*) [13].

No que se refere à *granularidade*, a contabilização realizada por probes RMON2 é feita por estação, pares de estações e protocolo utilizado. No caso da ferramenta `ntop`, é possível reconhecer e contabilizar fluxos de pacotes, especificados por regras em baixo nível que são usadas pelo filtro de pacotes BPF (*BSD Packet Filter*). Na arquitetura RTFM apenas parâmetros pré-determinados podem ser consultados dos pacotes capturados (somente até a camada de transporte). Informações de protocolos de níveis mais altos não podem ser consideradas devido a essa limitação. Além disso, a exemplo do que também ocorre com a ferramenta `ntop`, o mesmo conjunto de regras é aplicado para cada pacote recebido, impossibilitando a correlação de mensagens dentro de um determinado fluxo.

Vale ressaltar, ainda, que muitas das ferramentas de gerenciamento como [1, 6, 8] são limitadas à monitoração, cabendo ao gerente de rede tomar atitudes manualmente quando comportamentos inesperados desses protocolos são observados.

No que se refere a gerenciamento distribuído, Schönwälder et al. apresentam em [14] diversas abordagens e tecnologias existentes para a sua operacionalização. Tecnologias baseadas na delegação dinâmica de tarefas e, em especial, o potencial de delegação de tarefas de gerência através da MIB Script [15] são abordadas e comentadas. Através de exemplos práticos, eles demonstram como tarefas de monitoração de serviços e verificação de *thresholds*, entre outras, podem ser delegadas para gerentes de nível médio.

As próximas seções apresentam uma nova abordagem para gerenciamento de protocolos de alto nível e serviços de rede. A avaliação dessa abordagem comparada aos trabalhos mencionados nesta seção é apresentada na seção 6.

3 Representação de Traços de Protocolos

Esta seção apresenta PTSL (*Protocol Trace Specification Language*), uma linguagem para representação de traços de protocolos baseada no conceito de máquinas de estados finitas. A linguagem é composta de uma notação gráfica (*Graphical* PTSL) e uma textual (*Textual PTSL*). As notações não são equivalentes. A textual permite a representação completa de um traço, incluindo a especificação da máquina de estados e os eventos que provocam as transições. A notação gráfica, por sua vez, equivale a um sub-conjunto da textual, oferecendo a possibilidade de representar pictoricamente a máquina de estados e de apenas rotular os eventos que habilitam as transições.

3.1 Notação Gráfica (*Graphical* PTSL)

O gerente da rede pode criar uma especificação para monitorar todo ou apenas parte de um protocolo, ou interações abrangendo mais de um protocolo. As figuras 1 e 2 ilustram dois

exemplos de traços. No primeiro caso, o traço permite monitorar os acessos bem sucedidos a um servidor WWW. Já o traço apresentado na figura 2 não é constituído de mensagens de um único protocolo; é composto por uma requisição de resolução de nome (protocolo DNS), seguido de uma mensagem ICMP indicando *Port Unreachable*. Esse traço ocorre quando a estação onde o serviço DNS reside está ligada, mas o *daemon named* não está em execução.

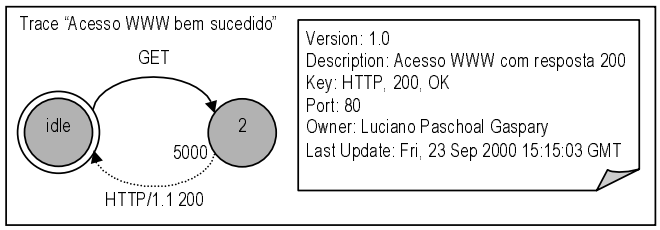

Figura 1: Representação de um traço usando *Graphical* PTSL

Representação de estados e transições. Os estados são representados por círculos. A partir do estado inicial denominado `idle` podem ser criados outros n estados, desde que os mesmos sejam sempre atingíveis por alguma transição. O estado final é identificado por dois círculos concêntricos. Em ambos os exemplos (figuras 1 e 2), os estados inicial e final são o mesmo (`idle`). As transições são representadas por setas unidirecionais. A seta contínua indica que a transição é provocada pela estação cliente, enquanto a pontilhada determina que a transição é disparada por um evento provocado pela estação servidora. O texto associado a uma transição apenas rotula o evento causador da transição; a especificação do mesmo só pode ser realizada através da notação textual.

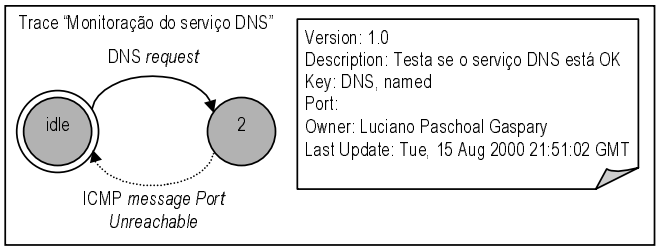

Figura 2: Traço composto de mensagens de mais de um protocolo

Representação de temporizadores. As transições não possuem limite de tempo para ocorrer. Para determinar um *timeout* para uma transição, é preciso associar um valor (em milissegundos) a ela. No exemplo ilustrado na figura 1, o valor `5000` associado à transição `HTTP/1.1 200` indica que a transição do estado `2` para o estado inicial tem 5 segundos para ocorrer (após a observação na rede da respectiva primitiva `GET`).

Representação de informações para catalogação e controle de versão. A notação gráfica oferece ainda um construtor onde são incluídas informações sobre o traço, que são relevantes para a catalogação e o controle de versão das especificações (vide figuras 1 e 2). As informações armazenadas para um traço são: `Version` (versão da especificação), `Description` (breve comentário a respeito do traço), `Key` (palavras-chave relacionadas ao traço), `Owner` (identificação do indivíduo que especificou o traço) e `Last Update` (data e hora da última atualização da especificação). Além dessas informações, existe o campo `Port`, usado para indicar a porta TCP ou UDP do protocolo sendo monitorado; este só deve ser preenchido quando o traço for restrito a um único protocolo de aplicação.

3.2 Notação Textual (*Textual* PTSL)

A figura 3 apresenta a especificação textual do traço ilustrado anteriormente na figura 1. Toda especificação descrita em *Textual* PTSL inicia com a palavra `Trace` e encerra com `EndTrace` (linhas 1 e 30). As informações para catalogação e controle de versão aparecem logo após a palavra-chave `Trace` (linhas 2 a 7). Em seguida, a especificação é dividida em três seções: `MessagesSection` (linhas 8 a 20), `GroupsSection` (não usada nesse exemplo) e `StatesSection` (linhas 21 a 29). Em `MessagesSection` e em `GroupsSection` são definidos os eventos que provocam as transições. Em `StatesSection` é definida a máquina de estados que representa o traço.

```
1  Trace "Acesso WWW bem sucedido"
2  Version: 1.0
3  Description: Acesso WWW com resposta 200
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 "Requisição de uma página Web"
13 EndMessage
14 Message "HTTP/1.1 200"
15 MessageType: server
16 MessageTimeout: 5000
17 FieldCounter Ethernet/IP/TCP 0 HTTP/1.1 "Versão do protocolo"
18 FieldCounter Ethernet/IP/TCP 1 200 "Código de retorno"
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
```

Figura 3: Representação de um traço usando *Textual* PTSL

Representação de mensagens. O evento que provoca a evolução de um máquina de estados é a observação na rede de um pacote que apresente campos com valores equivalentes aos especificados em uma mensagem (`Message`). A forma de especificar os campos a serem analisados depende do tipo de protocolo a ser monitorado. No caso de protocolos baseados em caracteres, que possuem campos de tamanho variável separados por espaços em branco (ex: HTTP, SMTP), a identificação de um campo é determinada pela posição do mesmo na mensagem (estratégia `FieldCounter`). Em `HTTP/1.1 200`, por exemplo, `HTTP/1.1` ocupa a posição 0 e `200`, a posição 1 da mensagem. Por outro lado, a identificação de campos em protocolos binários, caracterizados por possuirem campos de tamanho fixo (ex: TCP), é determinada por um *offset* em bits a partir do início do cabeçalho do protocolo em questão até o início do campo desejado; além da posição inicial do campo é preciso indicar ainda o número de bits que o campo ocupa (estratégia `BitCounter`).

O traço ilustrado na figura 1 é de um protocolo baseado em caracteres. A especificação da mensagem `GET` é ilustrada na figura 3 (linhas 9 a 13). Na linha 10 a mensagem é definida como sendo do tipo `client`, significando que a transição de estado à qual essa mensagem está associada será provocada pela estação cliente. Na linha 12 é especificado o único campo a ser analisado.

As informações necessárias para identificá-lo são: estratégia de localização (`FieldCounter`), encapsulamento do protocolo (`Ethernet/IP/TCP`), posição do campo (`0`), valor esperado (`GET`) e, opcionalmente, descrição do campo. Campos de protocolos baseados em caracteres são sempre identificados por esses cinco elementos.

A especificação da mensagem de retorno do traço `HTTP/1.1 200` é apresentada nas linhas 14 a 19. O tipo da mensagem é definido na linha 15 como `server`, ou seja, a transição de estado será provocada pela estação servidora. Na linha 16 `MessageTimeout` é definido com o valor `5000`, que corrresponde ao intervalo de tempo a ser aguardado pela observação da mensagem na rede. Por fim, são determinados os dois campos a serem analisados (linhas 17 e 18).

Em oposição ao exemplo supracitado, o traço especificado na figura 2 é baseado em protocolos binários: o DNS e o ICMP. A mensagem que provocará a evolução da máquina de estados do estado `idle` para o estado `2` é uma requisição DNS. Para filtrar, entre os pacotes monitorados, uma requisição DNS é preciso observar dois campos: o `QR` que, quando possui valor `1`, indica uma consulta ao servidor e o `OPCODE` que, quando possui valor `0`, indica que é uma consulta padrão. A identificação desses campos é realizada através da estratégia `BitCounter`. O campo `QR` está localizado a 16 bits do início do cabeçalho e seu tamanho é de 1 bit. O campo `OPCODE`, por sua vez, inicia no décimo sétimo bit e seu tamanho é de 4 bits.

A figura 4 apresenta a especificação textual da requisição DNS (linhas 1 a 6). Na linha 4 é identificado o campo `QR`. As informações necessárias para identificar o campo de um protocolo binário são: estratégia de localização (`BitCounter`), encapsulamento do protocolo (`Ethernet/IP/UDP`), posição do campo (`16`), tamanho do campo (`1`), valor esperado (`1`) e, opcionalmente, descrição do campo. As informações usadas para identificar o campo `OPCODE`, na linha 5, são `BitCounter`, `Ethernet/IP/UDP`, `17`, `4` e `0`. A mesma estratégia é usada para a definição da mensagem `ICMP message Port Unreachable` nas linhas 7 a 11.

```
1  Message "DNS request"
2  MessageType: client
3  // OffsetType Encapsulation FirstBit NumberOfBits Verb Description
4  BitCounter Ethernet/IP/UDP 16 1 1 "Campo QR – 1 significa DNS request"
5  BitCounter Ethernet/IP/UDP 17 4 0000 "Campo OPCODE – 0 significa Standard query"
6  EndMessage

7  Message "ICMP message Port Unreachable"
8  MessageType: server
9  BitCounter Ethernet/IP 0 8 00000011 "Campo Type 00000011: Destination unreachable"
10 BitCounter Ethernet/IP 8 8 00000011 "Campo Code 00000011: Port unreachable"
11 EndMessage
```

Figura 4: Identificação de campos em protocolos binários

Representação de agrupamentos de mensagens. A linguagem PTSL permite associar a ocorrência de uma transição a mais de um evento. Para tal, pode-se utilizar o construtor `Grouper` na seção `GroupsSection`. O traço apresentado na figura 1 possibilita observar a ocorrência de acessos bem sucedidos a um servidor WWW. Entretanto, apenas os acessos com retorno 200 são contabilizados. Os acessos com retorno 201, 202, 203, 204, 205 e 206, que também denotam operações bem sucedidas, podem ser incluídos nessa contabilização se forem especificadas (a) as mensagens que permitem identificá-los (especificação semelhante à ilustrada nas linhas 14 a 19 da figura 3 e (b) um agrupamento como o apresentado na figura 5. Nas linhas 2 e 3 são listadas as mensagens, já definidas na seção `MessagesSection`, que pertencem ao agrupamento. Na representação visual (figura 1), o rótulo associado à transição do estado `2` para `idle` deixa de ser `HTTP/1.1 200` e passa a ser `HTTP/1.1 200 || 201 || 202 || 203 || 204 || 205 || 206`, título do agrupamento (linha 1).

Representação da máquina de estados. As linhas 21 a 29 da figura 3 apresentam a especificação textual da máquina de estados ilustrada na figura 1. O estado final é identificado logo

```
1  Group "HTTP/1.1 200 || 201 || 202 || 203 || 204 || 205 || 206"
2  Messages: "HTTP/1.1 200", "HTTP/1.1 201", "HTTP/1.1 202", "HTTP/1.1 203", "HTTP/1.1 204", "HTTP/1.1 205",
3            "HTTP/1.1 206"
4  EndGroup
```

Figura 5: Representação de agrupamentos

após o início da seção `StatesSection` (linha 22). Os estados `idle` e `2` são definidos, respectivamente, nas linhas 23 a 25 e 26 a 28. A especificação de um estado se resume a listar os eventos (mensagens e agrupamentos) que podem provocar uma transição e indicar, para cada um deles, o próximo estado (linhas 24 e 27).

4 A Arquitetura Trace

A arquitetura **Trace** [16] é uma extensão da infra-estrutura centralizada de gerenciamento SNMP para, através de um modelo em três camadas, suportar o gerenciamento distribuído de protocolos de alto nível e serviços de rede. A figura 6 ilustra um esquema da arquitetura. Implantada com base na MIB Script [15], ela oferece mecanismos para que, a partir de uma estação de gerenciamento, seja possível a delegação de tarefas de gerenciamento a gerentes intermediários que, por sua vez, interagem com agentes de monitoração e agentes de ação para concretizá-las. Especificações PTSL são usadas pelos gerentes intermediários para programar os agentes de monitoração, que passam a observar a ocorrência dos traços. De posse de informações obtidas através da monitoração, os gerentes intermediários podem requisitar a agentes de ação a execução de procedimentos (ex: *scripts* Perl), viabilizando a automação de diversas tarefas de gerenciamento. A arquitetura apresenta ainda mecanismos de notificação (*traps*) para que os agentes possam reportar eventos ao gerente intermediário e para que este possa sinalizar a ocorrência de eventos significativos à estação de gerenciamento. A seguir são apresentados os componentes da arquitetura.

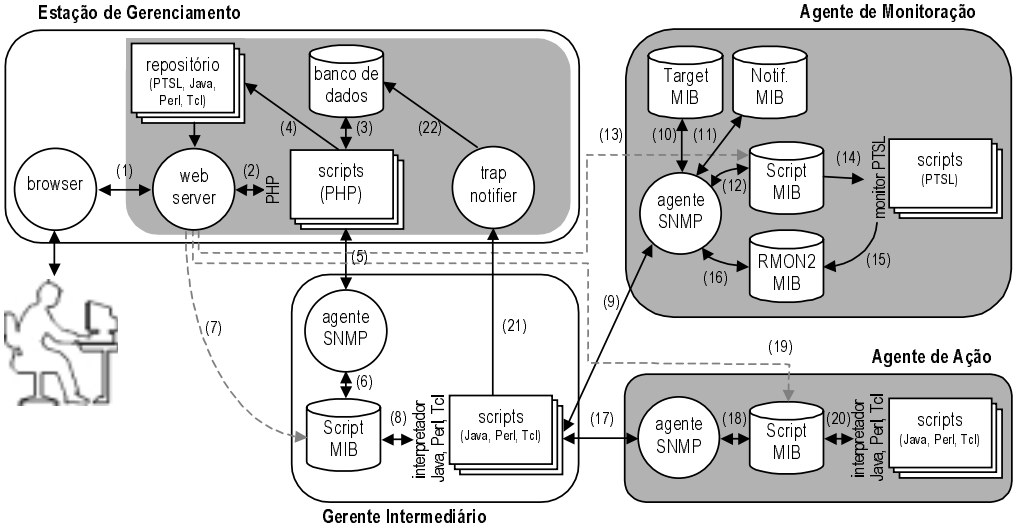

Figura 6: Componentes da arquitetura **Trace**

Estação de gerenciamento. A arquitetura é composta por uma ou mais estações de gerenciamento (gerentes). Desde que respeitadas as interfaces com os demais componentes da arquitetura, nada impede que as aplicações de gerenciamento sejam desenvolvidas com tecnologias variadas. A figura 6, entretanto, sugere que a interface do gerente de rede com a arquitetura

de gerenciamento seja baseada na Web. Através de um navegador Internet, o gerente acessa o ambiente de gerenciamento em um servidor Web. Nosso grupo de pesquisa elegeu a linguagem PHP para desenvolver o ambiente. Os módulos destacados na estação de gerenciamento (figura 6) podem estar hospedados na própria estação onde se encontra o gerente ou em uma outra estação. Se houver mais de uma estação de gerenciamento, elas poderão compartilhar o mesmo núcleo do ambiente.

As atividades mais importantes realizadas pelo gerente de rede a partir de uma estação de gerenciamento são:

- *Cadastramento de gerentes intermediários e agentes*: para facilitar a coordenação entre estação de gerenciamento, gerentes intermediários e agentes, o gerente da rede deve determinar quem são os gerentes intermediários de sua rede, bem como os agentes subordinados a cada um deles. Esse amarramento é importante para organizar que gerentes são responsáveis por que agentes. Ao programar uma tarefa de gerenciamento o gerente intermediário manipulará os agentes a ele subordinados. As interações necessárias para o cadastramento são representadas na figura 6 por (1, 2, 3). Essa numeração será usada ao longo da seção para ilustrar o fluxo de dados na arquitetura.
- *Especificação de um traço de protocolo (script PTSL)*: usando a linguagem apresentada na seção 3, é possível especificar um traço de protocolo. O gerente de rede pode usar o ambiente para especificar um traço a partir do zero ou para recuperar traços existentes no repositório e derivar uma nova especificação a partir de um traço já definido (1, 2, 3). Para que possa ser utilizada posteriormente por um agente de monitoração, a especificação do traço, armazenada no banco de dados, precisa ser mapeada para um arquivo texto e disponibilizada no repositório (4).
- *Especificação de uma ação*: os *scripts* de ação não precisam obrigatoriamente ser especificados usando as funcionalidades do ambiente. É possível realizar o *upload* de um *script* desenvolvido em Java, Perl ou Tcl para o repositório (1, 2, 4). Antes disso, é recomendável que o mesmo seja exaustivamente testado.
- *Especificação de uma tarefa de gerenciamento*: através de um *wizard* disponibilizado pelo ambiente de gerenciamento, o gerente de rede especifica uma tarefa de gerenciamento (1, 2, 3). Ao defini-la, o gerente de rede informa o traço a ser observado, os agentes responsáveis pela monitoração, os objetos a serem consultados, as ações a serem realizadas e os agentes responsáveis por executá-las. Em alguns casos, se a tarefa depender do recebimento de *traps* de agentes de monitoração e ação para poder cumprir com a sua função, é preciso informar ainda as *traps* que o gerente intermediário pode receber (que são do interesse da tarefa), a ação associada ao recebimento de cada uma delas e a indicação das notificações que devem ser repassadas à estação de gerenciamento. Essas especificações, a exemplo do que ocorre com as especificações PTSL, são mantidas no banco de dados.
- *Delegação de uma tarefa de gerenciamento*: para delegar uma tarefa a um gerente intermediário é preciso recuperá-la do banco de dados (1, 2, 3). A partir das informações armazenadas, um *script* em Java, Perl ou Tcl correspondente é gerado automaticamente e disponibilizado no repositório (4). Em seguida a execução do *script* é delegada ao gerente intermediário (5, 6) via SNMP (MIB Script).
- *Acompanhamento de uma tarefa de gerenciamento*: durante a execução de uma tarefa de gerenciamento, o gerente pode consultar o gerente intermediário para obter resultados parciais da execução da tarefa (1, 2, 5, 6).
- *Interrupção de uma tarefa de gerenciamento*: a interrupção de uma tarefa de gerenciamento requer a desprogramação dos agentes de monitoração e ação envolvidos. Só após é possível finalizar a execução do *script*, que corresponde à tarefa de gerenciamento, no gerente intermediário (1, 2, 5, 6).

- *Recebimento e visualização de traps*: o gerente pode receber *traps* via um módulo denominado notificador de *traps* (21). Quando recebidas elas são armazenadas no banco de dados (22). As *traps* são permanentemente recuperadas por um *script* (3) que atualiza o navegador do gerente (2, 1) (tecnologia *push*) para que ele receba imediatamente a notificação.

Gerente Intermediário. O gerente intermediário tem por função executar e acompanhar tarefas de gerenciamento, delegadas pela estação central de gerenciamento, e reportar eventos significativos a ela. Uma rede pode possuir um ou mais gerentes intermediários. O número de gerentes intermediários é determinado pelo gerente da rede e depende de fatores como tamanho e complexidade da infra-estrutura a ser gerenciada.

A delegação de uma tarefa a um agente intermediário, como já mencionado, é realizada pela estação de gerenciamento através de primitivas SNMP, suportadas pela linguagem PHP (5, 6). Ao receber, entre outros parâmetros, o endereço URL do *script* a ser executado, a MIB Script recupera o *script* do repositório via HTTP (7) e inicia a execução do mesmo (8).

Os *scripts* executáveis pelo gerente intermediário podem ser codificados em qualquer linguagem de programação, desde que o ambiente de gerenciamento implemente o mapeamento da especificação da tarefa de gerenciamento, mantida no banco de dados, para a linguagem. A implementação da MIB Script usada na prototipação da arquitetura é o Jasmin [17], que atualmente suporta Java, Perl e Tcl. A complexidade dos *scripts* executados pelo gerente intermediário não chega a ser um fator crítico, uma vez que a especificação e a delegação das tarefas de gerenciamento são realizadas através de *wizards*.

A figura 7 apresenta um *script* exemplo simplificado em Java. Trata-se da tarefa que realiza a monitoração do serviço DNS. Um agente de monitoração é programado para observar a ocorrência do traço `Monitoração do serviço DNS`, ilustrado na figura 2. O gerente intermediário consulta, a cada 60 segundos, o agente de monitoração para saber se o traço foi observado na rede. Caso ele tenha ocorrido, o agente de ação, hospedado na estação onde se encontra o serviço, é programado para executar um *script* que reinicia o *daemon named*.

Os *scripts* gerados possuem um formato padrão para todas as tarefas de gerenciamento. As declarações e os procedimentos mais significativos do *script* são: (a) instalação do *script* de monitoração (linhas 4 a 7), (b) criação de uma instância do *script* de monitoração (linhas 9 a 11), (c) instalação do *script* de ação (similar ao item a), (d) criação de uma instância do *script* de ação (similar ao item b), (e) execução do *script* de monitoração (linha 15), (f) identificação das variáveis a serem monitoradas (linhas 16 a 25), (g) laço de monitoração (linhas 27 a 38) e (h) execução do *script* de ação (linhas 33 e 34), dependendo dos valores observados (linha 31).

O *script* executado pelo gerente intermediário pode ainda cadastrar, junto aos agentes de monitoração e ação, as *traps* que deseja receber. Através da MIB *Target* é feita a identificação do gerente intermediário (endereço IP) e da porta UDP usada para o envio das *traps* (9, 10) ao *script*. A MIB *Notification*, por sua vez, permite a configuração (através de filtros) das *traps* que devem ser enviadas a ele (9, 11) [18]. O *script*, se possuir uma ação associada ao recebimento da *trap*, a executa. Essa *trap* pode ainda ser correlacionada com outras recebidas e, como resultado, o *script* pode enviar uma única notificação à estação de gerenciamento (21).

Como é possível observar, a comunicação do gerente intermediário com agentes de monitoração e agentes SNMP convencionais, realizada pelo *script*, se dá usando primitivas SNMP (linhas 16 a 25). O mesmo ocorre na comunicação do gerente intermediário com a estação de gerenciamento na geração de *traps*. Já a programação da MIB Script nos agentes de monitoração e ação (interações 9, 12 e 17, 18 na figura 6) é feita com o uso de uma biblioteca que permite a manipulação dessa MIB através de primitivas de mais alto nível (linhas 4 a 7, 9 a 11, 15, 33 e 34).

```
1  public class CreateScript {
2     public static void main( String [] args) throws Exception {
3     // Instalação do script de monitoração
4     ScriptMib monit_scriptMib = new ScriptMib(monit_agent, port, community);
5     Language langPTSL = new Language(monit_scriptMib, 10);
6     Script monit_script = new Script(monit_scriptMib, owner, monit_scriptname, monit_scriptdescr, langPTSL, trace_url);
7     monit_script.enable();
8     // Criação do launch do script de monitoração
9     String monit_launchname = new String(monit_scriptname + "-" + System.currentTimeMillis());
10    Launch monit_launch = new Launch(monit_scriptMib, owner, monit_launchname, monit_script);
11    monit_launch.enable();
12    // Instalação do script de ação
13    // Criação do launch do script de ação
14    // Execução do script de monitoração
15    Run monit_run = monit_launch.start();
16    String monitor_community = new String("public");
17    SnmpPeer peer = new SnmpPeer(monit_agent, InetAddress.getByName(monit_agent), monitor_community);
18    peer.port = 161;
19    SnmpConnection connection = new SnmpConnection(peer);
20    Vector oids = new Vector();
21    // OID do objeto protocolDistStatsPkts na interface 1 para o encapsulamento 10 (DNS Query–ICMP Port Unreachable)
22    String dns_trace_oid = "1.3.6.1.2.1.16.12.2.1.1.1.10";
23    Varbind[] result1, result2;
24    oids.addElement(new OID(dns_trace_oid));
25    result1 = connection.getRequest(oids);
26    // Loop de monitoração de objeto por SNMP
27    while (true) {
28       Thread.sleep(60000);
29       result2 = connection.getRequest(oids);
30       // Testa se o traço ocorreu nos últimos 60 segundos
31       if (((Long)result2[0].getArray()[1].content).intValue() – ((Long)result1[0].getArray()[1].content).intValue()> 0) {
32          // Sim, ocorreu.  Execução do script de ação
33          Run action_run = action_launch.start();
34          action_run.block(); // Espera pelo término da execução
35          // Recuperação do resultado do script para verificar se executou com sucesso ou não
36       }
37       result1 = result2;
38    }
39    }
40 }
```

Figura 7: Exemplo de *script* executado pelos agentes intermediários

Agente de Monitoração. Os agentes de monitoração contabilizam a ocorrência de traços no segmento onde se encontram. São denominados extensíveis porque os traços a serem monitorados podem ser configurados dinamicamente, sem a necessidade de recompilar esses agentes. Esta flexibilidade é obtida através da linguagem PTSL, apresentada na seção 3. Na prática, os agentes realizam a leitura de arquivos PTSL, organizam algumas estruturas em memória e iniciam o processo de monitoração.

A determinação de que traços devam ser monitorados em um dado momento é realizada pelo gerente intermediário. A interface de comunicação entre o gerente intermediário e o agente de monitoração é a MIB Script (9,12). No *script* executado pelo gerente intermediário, apresentado na figura 7, é possível observar como é feita a programação de um agente de monitoração (linhas 4 a 11 e 15). Um dos parâmetros informados nesse processo é o endereço URL do *script* (especificação PTSL) a ser executado. Ao receber do gerente intermediário a solicitação de execução de um determinado *script*, esse é recuperado do repositório via HTTP (13) e executado (14).

Na realidade, uma especificação PTSL não é executável. A semântica associada à `monit launch.start()` (linha 15 da figura 7) é fazer com que o agente de monitoração passe a monitorar o novo traço. De forma análoga, a interrupção de um *script* na MIB Script significa programar o agente de monitoração para que ele cesse a monitoração do traço definido por esse *script*.

Toda vez que a ocorrência do traço é observada entre qualquer par de estações, informações são armazenadas em uma MIB similar à RMON2 [5, 12](15). Uma das diferenças da MIB usada com relação à RMON2 é que o grupo `protocolDir`, que indica os protocolos que o agente é capaz de monitorar, passa a permitir que traços de interesse, e não apenas encapsulamentos de protocolos completos, possam ser indexados. Como conseqüência, a granularidade da monitoração torna-se maior. Ao invés de armazenar estatísticas globais sobre o tráfego gerado por um determinado protocolo, as estatísticas são geradas de acordo com a ocorrência dos traços especificados.

O grupo `alMatrix`, da MIB RMON2, armazena estatísticas sobre o traço, quando observado entre cada par de estações. A tabela 1 ilustra o conteúdo da tabela `alMatrixSD`. Ela contabiliza o número de pacotes e octetos entre cada par de máquinas (cliente/servidor). No caso, três traços foram observados: `Monitoração do serviço DNS`, `Acesso WWW bem sucedido` e `Acesso inválido a serviço TCP`.

Tabela 1: Informações obtidas com consulta à tabela `alMatrixSD`

Source Address	*Destination Address*	*Protocol*	*Packets*	*Octets*
172.16.108.1	172.16.108.2	Monitoração do serviço DNS	4	4.350
172.16.108.32	172.16.108.2	Monitoração do serviço DNS	8	7.300
172.16.108.1	172.16.108.254	Acesso WWW bem sucedido	254	1.202.126
125.120.10.100	172.16.108.254	Acesso inválido a serviço TCP	20	3.204

A desvantagem em usar a MIB RMON2 é que ela não possui objetos capazes de armazenar informações relacionadas a desempenho. Por essa razão, nosso grupo avalia atualmente a possibilidade de utilizar, adicionalmente, uma extensão da RMON2, como a MIB *Application Performance* [13]. A tabela 2 apresenta o tipo de informações armazenadas pela MIB. A primeira linha indica que o traço `Acesso WWW bem sucedido` foi observado 127 vezes entre as máquinas `172.16.108.1` e `172.16.108.254`. O número de traços que não completaram com sucesso foi de 232. Incluem-se nesta estatística todas as mensagens `GET` para as quais um retorno diferente de `HTTP/1.1 200` foi observado. Além disso, o tempo médio de resposta das observações bem sucedidas foi `6` segundos.

Tabela 2: MIB com informações sobre desempenho

Client	*Server*	*Protocol*	*Success.*	*Unsuccess.*	*Responsiv.*
172.16.108.1	172.16.108.254	Acesso WWW bem sucedido	127	232	6 sec.
172.16.108.1	200.248.252.1	Acesso WWW bem sucedido	232	112	17 sec.
10.10.135.125	200.248.252.1	Fase final da conexão	10.234	56	3 sec.

Agente de Ação. Através dos agentes de monitoração, o gerente intermediário tem condições de avaliar a ocorrência ou não de um traço. Conforme foi apresentado, os traços podem indicar a falha em um serviço de rede, a tentativa de intrusão, a queda de desempenho em algum serviço, entre outros problemas. Nesse contexto, os agentes de ação são responsáveis pela execução de um procedimento de gerenciamento criado para combater, sem a intervenção humana e de forma automática, o problema detectado.

Tomemos como exemplo o serviço DNS. O gerente intermediário, ao detectar que o serviço não está disponível (através do laço de monitoração), solicita ao agente de ação, localizado na estação onde o serviço reside, que execute um procedimento para reinicar o *daemon named*.

A comunicação entre o gerente intermediário e o agente de ação também se dá através da MIB Script (17,18). No *script* exemplo da figura 7, não é apresentada a criação e instanciação do *script* no agente de ação, mas o procedimento é similar ao realizado para programar o agente de monitoração. O endereço URL de um *script* localizado no repositório é passado para a MIB Script executar (17, 18, 19, 20). O *script*, nesse exemplo, é escrito usando a linguagem Perl. A utilização dessa linguagem no desenvolvimento de *scripts* de ação é bastante adequada, posto que é poderosa e amplamente utilizada para programar procedimentos de administração/gerenciamento. A figura 8 ilustra o *script* usado para reiniciar o serviço DNS.

```
#!/usr/bin/perl

my $pid;

# verifica se o processo named esta executando
if (-e "/var/run/named.pid") {
    $pid = `/bin/cat /var/run/named.pid`;
}

# se named está executando, reinicia através de um sinal HUP, senão inicia o processo novamente
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 &`;
}

# verifica se o processo está em execução
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";
}
```

Figura 8: *Script* Perl para reiniciar o *daemon named*

5 Estudos de Caso

A arquitetura **Trace** foi projetada para permitir o gerenciamento de todas as áreas funcionais (FCAPS). Nosso grupo explorou, através de estudos de caso, as características da arquitetura para validar a sua aplicabilidade no gerenciamento de protocolos de alto nível e serviços de rede.

A figura 9 ilustra um cenário real de gerenciamento, composto de três domínios. A organização desses domínios é uma tarefa que o gerente da rede precisa realizar para fazer uso eficiente da arquitetura. Esta tarefa é concretizada via ambiente de gerenciamento, no momento do cadastramento de gerentes intermediários e agentes.

O domínio 1 é composto por equipamentos e serviços relacionados ao acesso da organização à Internet (em cinza escuro na figura). O roteador, por intermédio de uma interface serial, é o elo de ligação com a mesma. Além da interface serial, o roteador possui duas interfaces Ethernet. À primeira interface está conectado um *hub*, que tem ligado a ele duas estações: uma hospeda o servidor de DNS e outra, o servidor Web. À segunda interface do roteador está ligada uma estação com duas interfaces de rede. Esta executa um *firewall* e, portanto, representa a divisa entre a rede externa e a interna. Ligado à outra inteface do *firewall* (interface interna) está um *hub*. Nele estão conectados o servidor Web responsável pela Intranet da organização e um *switch* que segmenta a rede interna em várias sub-redes. Os demais equipamentos (em branco na figura) são a estação de gerenciamento, o gerente intermediário do domínio 1 (ambos conectados ao *switch*) e duas estações dedicadas à tarefa de monitoração. Com base nesse cenário, nosso grupo de trabalho determinou algumas tarefas de gerenciamento, apresentadas a seguir.

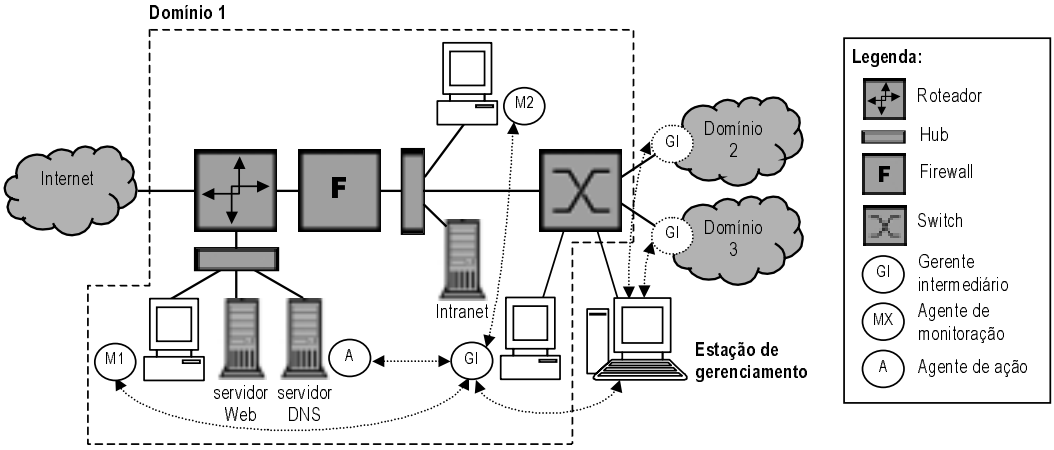

Figura 9: Uma rede e alguns de seus serviços

Monitoração da ocorrência de falhas no servidor de DNS. Consiste em observar a falha de funcionamento do *daemon named*. O gerente intermediário responsável pelo domínio 1 programa, através do *script* Java apresentado na figura 7, o agente de monitoração M1, localizado no mesmo segmento onde se encontra o servidor de DNS, para observar a ocorrência do traço `Monitoração do serviço DNS` (figura 2). Caso ele seja observado pelo menos uma vez em um intervalo de consulta, o gerente intermediário solicita ao agente de ação A1 a execução do *script* para reiniciar o serviço (figura 8).

Contabilização dos acessos ao servidor Web. Consiste em medir o volume de acessos ao servidor Web, não apenas os bem sucedidos, mas também os mal sucedidos e as tentativas não autorizadas. Essas informações permitem ao gerente, por exemplo, (a) conhecer os horários mais críticos de acesso e providenciar a expansão ou reconfiguração do servidor Web para suportar mais acessos simultâneos, (b) contabilizar a ocorrência de problemas em clientes HTTP no acesso a páginas mantidas no servidor e minimizar o problema revisando as páginas mantidas no *site* e (c) reconfigurar o servidor Web para não mais aceitar acessos da estação de onde partiram as tentativas de acesso não autorizadas.

As figuras 1 e 3 apresentam o traço para contabilizar acessos bem sucedidos ao servidor Web. A figura 5 ilustra como associar várias mensagens a uma mesma transição. Os traços para contabilizar os acessos mal sucedidos e as tentativas não autorizadas são similares. Para contabilizar os acessos mal sucedidos é preciso definir um traço que monitore a ocorrência da primitiva `GET` seguida de uma resposta `HTTP/1.1 4XX`, onde `XX` vária de `00` a `15`. O mesmo ocorre na definição do traço para monitorar tentativas de acesso não autorizadas, que é caracterizado pela ocorrência da primitiva `GET` com retorno `HTTP/1.1 3XX`, onde `XX` vária de `00` a `05` [19].

A monitoração dos acessos ao servidor WWW localizado na rede externa é realizada através da programação do agente de monitoração M1, enquanto que o servidor responsável pela Intranet é monitorado através da programação do agente M2. Cada uma das monitorações faz parte de uma tarefa de gerenciamento distinta, embora executem procedimentos similares. A cada iteração do laço de monitoração presente no *script* executado pelo agente intermediário, o valor obtido (vide tabelas 1 e 2) é comparado com limiares definidos pelo gerente. Para cada intervalo de consulta em que o número de ocorrências do traço supera o limiar definido, o *script* gera uma notificação à estação central de gerenciamento.

Monitoração da segurança nos servidores de DNS e WWW. Os servidores de DNS e WWW localizados na rede externa estão vulneráveis a ataques maliciosos, oriundos da Internet. A monitoração da segurança nos servidores de DNS e WWW consiste em realizar a monitoração

das estações que hospedam esses serviços para verificar se estão sendo vítimas de varredura de portas, de ataques de negação de serviço, entre outros detectáveis através da monitoração passiva.

A varredura de portas consiste em enviar pacotes para todas as portas de uma estação para saber quais são os serviços TCP e UDP oferecidos por ela. No caso do protocolo TCP, se a estação não oferecer o serviço em uma determinada porta, ela envia um pacote TCP de retorno com o bit `RST` ligado em resposta à tentativa de conexão. A figura 10 ilustra esse traço.

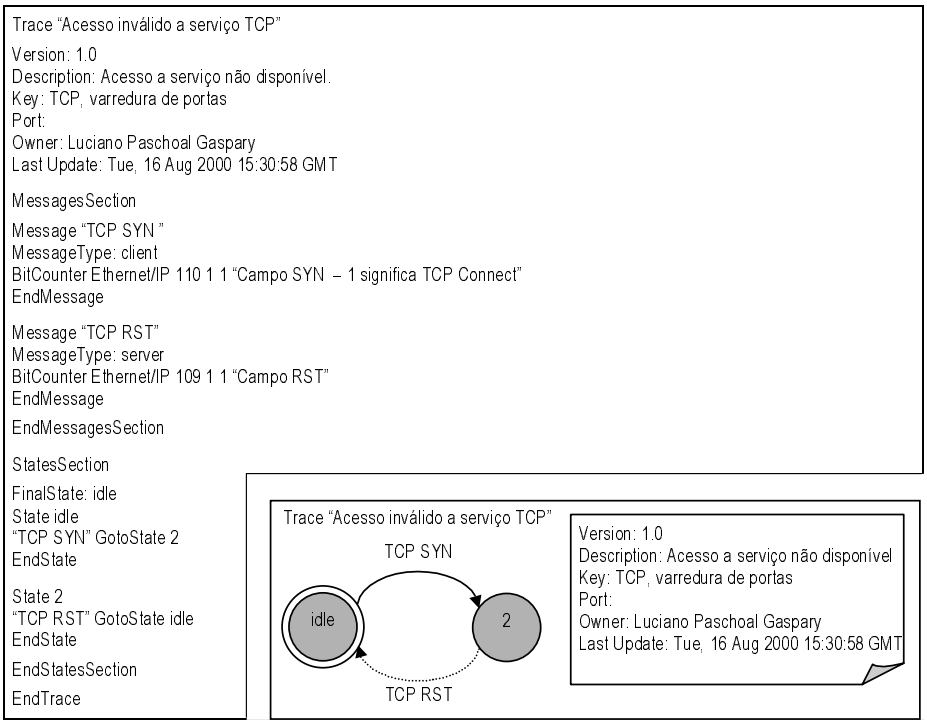

Figura 10: Traço para detectar varredura de portas

O gerente intermediário programa o agente de monitoração M1 para que passe a observar a ocorrência do traço. Além disso, ele consulta periodicamente a MIB RMON2 extendida onde os resultados da monitoração são armazenados (vide tabelas 1 e 2). Se em um intervalo de consulta o número de ocorrências superar um determinado valor, definido pelo gerente na especificação da tarefa de gerenciamento, o *script* gera uma notificação à estação central de gerenciamento.

Procedimento similar é realizado quando uma das estações sofre o ataque conhecido como *SYN Flood*. Este ataque consiste em enviar um grande número de pacotes de abertura de conexão (pacote com o flag `SYN` ligado) com um endereço de origem falso para uma determinada estação alvo. Esse endereço de origem falso deve ser inalcançável ou de uma estação inexistente (muitas vezes se usa um dos endereços reservados). A estação alvo, ao receber esses pacotes de abertura de conexão (`SYN`), cria uma entrada na fila de conexão e envia um pacote de resposta (`SYN/ACK`) para o endereço que solicitou a conexão. Após o envio do pacote de resposta, a estação alvo fica aguardando uma confirmação do solicitante da conexão. Como o endereço de origem dos pacotes é forjado, a estação alvo nunca receberá essa confirmação de conexão (vide figura 11a). Em um determinado momento, a fila de conexões da estação alvo fica lotada, e a partir daí todos os pedidos de abertura de conexão são descartados e o serviço indisponibilizado. Essa indisponibilização persiste durante alguns segundos, pois a estação alvo ao descobrir que a confirmação está demorando demais, remove a conexão aberta da lista.

A identificação desse ataque é realizada pelo traço ilustrado na figura 11b. Ao contrário dos exemplos anteriores, o ataque é identificado pela observação de vários insucessos de ocorrência do traço. Esta informação é armazenada pela MIB APM, conforme ilustrado na tabela 2.

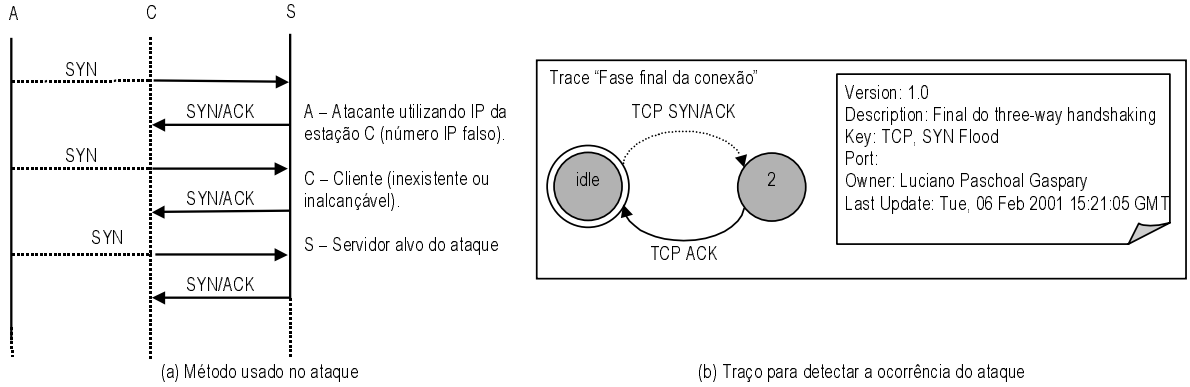

Figura 11: O ataque *SYN Flood*

6 Conclusões e Trabalhos Futuros

Esse trabalho apresentou uma arquitetura para gerenciamento de protocolos de alto nível e serviços de rede baseada na utilização de agentes programáveis. Motivado pela crescente necessidade das organizações em gerenciar protocolos de alto nível e, principalmente, as suas aplicações críticas, o trabalho propõe uma arquitetura flexível, capaz de acompanhar a rápida proliferação de protocolos e aplicações sobre redes (que precisam ser gerenciadas). Por ser baseada em infra-estrutura padronizada pelo IETF, não exige a ruptura com o que já existe implantado em boa parte das organizações (que demorou anos até ser consolidado).

A proposta da linguagem PTSL é uma das importantes contribuições desse trabalho. As abordagens estudadas e listadas na seção 2 se limitam a contabilizar pacotes enviados/recebidos entre determinados pares de estações classificando-os por protocolo [5] ou por fluxo [1, 8]. Nessas abordagens, as informações a que o gerente de rede tem acesso são, por exemplo, que entre as estações A e B foram observados n octetos/pacotes (a) do protocolo HTTP ou (b) de pacotes nos quais alguns de seus campos possuem valores pré-determinados. As inovações agregadas à PTSL aumentam a granularidade com que os protocolos podem ser monitorados, possibilitando analisar o comportamento de um protocolo ou parte dele ao propor a representação de traços de interesse. Isto instrumentaliza o gerente de rede com informações mais precisas, que lhe auxiliarão a operacionalizar o gerenciamento de falhas, configuração, contabilização, desempenho e segurança voltado a protocolos de alto nível e serviços de rede. Usando o mesmo exemplo anterior, o uso da linguagem permite observar e contabilizar, por exemplo, a ocorrência de acessos HTTP bem sucedidos, mal sucedidos e tentativas não autorizadas, bem como qualquer traço possível no protocolo HTTP. O poder de expressão de PTSL é outro ponto a seu favor. Enquanto boa parte das abordagens permite selecionar pacotes com base em alguns campos pré-determinados de protocolos até, no máximo, a camada de transporte [8], PTSL vai além ao possibilitar que a filtragem seja feita com base em quaisquer campos de quaisquer protocolos, incluindo os do nível de aplicação.

Com relação à arquitetura **Trace**, a contribuição mais evidente é a possibilidade de realizar o gerenciamento efetivo de protocolos de alto nível e serviços de rede, ao integrar a linguagem PTSL a agentes de monitoração programáveis e associar a ocorrência dos traços definidos pelo gerente a ações, também dinamicamente programáveis, viabilizando a automação de um conjunto de procedimentos de gerenciamento. A arquitetura não se limita à monitoração. Ao contrário, provê uma solução mais completa e abrangente que inclui a execução de ações, viabilizando tanto o gerenciamento reativo quanto o pró-ativo.

Outro aspecto positivo da arquitetura é o aumento significativo de escalabilidade com relação ao paradigma tradicional de gerenciamento SNMP porque possiblita delegar tarefas de gerenciamento, antes realizadas apenas pela estação central de gerenciamento, para gerentes

intermediários. A robustez agregada às tarefas de gerenciamento também representa uma contribuição importante. A arquitetura permite delegar funções de gerenciamento para gerentes intermediários que estejam bem perto dos agentes observados; assim, essas tarefas podem ser executadas mesmo que ocorram problemas de comunicação entre a estação central de gerenciamento e o gerente intermediário.

No entanto, por ser distribuída, a arquitetura exige mais trabalho para ser controlada. O gerenciamento dos componentes da arquitetura torna-se uma tarefa relativamente complicada. Nesse gerenciamento dos componentes inclui-se a distribuição e a atualização de scripts, a obtenção e a correlação de resultados. Um dos trabalhos a serem realizados é a criação de mecanismos que tornem mais transparente o uso da arquitetura.

Com relação à implementação da arquitetura é preciso destacar que o protótipo desenvolvido está sendo melhorado para oferecer a transparência supracitada. A interface do ambiente de gerenciamento, até agora deixada em segundo plano, está sendo remodelada. Testes relacionados ao desempenho da arquitetura ainda precisam ser realizados, mas Schönwälder, em [14], apresenta resultados favoráveis de testes de desempenho da implementação do Jasmin, implementação da MIB Script usada para implementar a arquitetura.

Referências

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

[2] L. R. Tarouco e A. C. B. Silva. "Uma Proposta para Gerência de Correio Eletrônico". Anais do *XII Simpósio Brasileiro de Redes de Computadores*, Curitiba, 1994.

[3] J. A. Carrilho. "Um Modelo para o Gerenciamento do Protocolo FTP Baseado em Domínios". *Dissertação de Mestrado*. Campinas: DCC da Unicamp, 1994.

[4] C. K. Silveira e E. R. M. Madeira. "Um Esquema para o Gerenciamento do Tráfego de Aplicações em Redes TCP/IP. Anais do *XIII Simpósio Brasileiro de Redes de Computadores*, Belo Horizonte, 1995, p. 567–585.

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

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

[7] N. Brownlee, C. Mills and G. Ruth. "Traffic Flow Measurement: Architecture". Request for Comments 2722, October 1999.

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

[9] N. Brownlee. "Traffic Flow Measurement: Meter MIB". Request for Comments 2720, October 1999.

[10] N. Brownlee. "SRL: A Language for Describing Traffic Flows and Specifying Actions for Flow Groups". Request for Comments 2723, October 1999.

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

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

[13] S. Waldbusser. "Application Performance Measurement MIB". Internet Draft, May 2000.

[14] 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.

[15] D. Levi and J. Schönwälder. "Definitions of Managed Objects for the Delegation of Management Scripts". Internet Draft, Nortel Networks, TU Braunschweig, July 2000.

[16] 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". To appear in *Proc. First IEEE/IEC Enterprise Networking Applications and Services Conference*, Atlanta, June 2001.

[17] TU Braunschweig, NEC C&C Research Laboratories. Jasmin - A Script MIB Implementation, 1999. `http://www.ibr.cu.tu-bs. de/projects/jasmin`.

[18] D. Levi, P. Meyer and B. Stewart."SNMP Applications". Request for Comments 2573, April 1999.

[19] R. Fielding et al. "Hypertext Transfer Protocol - HTTP/1.1". Request for Comments 2068, January 1997.