ANEMONA: Uma Linguagem de Configuração para Aplicações Práticas de Gerência Distribuída

Henrique Denes H. Fernandes, Elias Procópio Duarte Jr., Martin A. Musicante

Universidade Federal do Paraná, Departamento de Informática
Caixa Postal 19081 Curitiba PR 81531-990
denes@cce.ufpr.br, {elias, mam}@inf.ufpr.br

Resumo

Neste trabalho apresentamos a linguagem ANEMONA para configuração de aplicações de gerência distribuída. A linguagem permite a definição de expressões de objetos de gerência a serem monitorados, bem como a definição de condições relacionadas aos objetos, que podem sinalizar eventos associados, também definidos pela linguagem. Um tradutor da linguagem foi desenvolvido, gerando código para a configuração da *Expression MIB* e da *Event MIB*, dois padrões de gerência distribuída do IETF. Resultados práticos da utilização da ferramenta são apresentados. Em um primeiro experimento, uma aplicação ANEMONA detecta um ataque do tipo *TCP Syn Flooding*. Em seguida, é descrita outra aplicação que permite a monitoração de enlaces para a emissão de alarmes frente à ocorrência de um aumento brusco na sua taxa de utilização.

Abstract

This work presents the ANEMONA language for distributed network management applications programming. The language allows the definition of expressions of managed objects that are monitored, as well as triggers that when are fired may indicate the occurrence of associated events, which are also defined by the language. A translator for the language was developed. It generates code for configuring both the *Expression MIB* and *Event MIB*, which are IETF standards for distributed management. Practical results are presented. In the first case study, an ANEMONA application detects a *TCP Syn Flooding* attack. Another application is described which allows network link monitoring, generating alarms that indicate the occurrence of a steep increase on the link utilization rate.

Palavras Chave: Gerência de Redes, SNMP, Gerência Distribuída, Linguagens de Programação

1 Introdução

O paradigma distribuído para gerência de redes vem sendo adotado em sistemas baseados na arquitetura padrão para gerência da Internet, o SNMPv3 (*Simple Network Management Protocol version 3*) [1]. Um sistema baseado em SNMPv3 [2] é constituído por entidades de gerência que assumem o papel de gerentes, agentes ou *proxies*. Estes agentes se comunicam através do protocolo SNMPv3 propriamente dito. As entidades de gerência distribuem entre si as tarefas para a monitoração e o controle da rede.

O grupo de trabalho do IETF (*Internet Engineering Task Force*) DISMAN (*DIStributed MANagement*) tem como um dos seus objetivos a padronização da gerência distribuída no âmbito da Internet [3]. Entre os padrões publicados, estão as MIB's (*Management Information Bases*) *Expression* e *Event*. A *Expression MIB* [4] permite a construção e avaliação de expressões com valores obtidos a partir de objetos gerenciados, enquanto que a *Event MIB* [5] monitora um conjunto de objetos, podendo sinalizar um evento quando uma condição programada ocorrer.

Este trabalho descreve a linguagem ANEMONA (*A NEtwork MONitoring Application*) que permite a definição de expressões de objetos de gerência a serem monitorados, bem como a definição de condições relacionadas aos objetos, que podem sinalizar eventos associados, também definidos pela linguagem. Um tradutor da linguagem foi desenvolvido, gerando código para a configuração da *Expression MIB* e da *Event MIB*. Um conjunto de agentes distribuídos pela rede pode monitorar quaisquer objetos e expressões de objetos, bem como gerar alarmes frente à ocorrência de condições pré-definidas. Aplicações de monitoração são importantes para a gerência de segurança, desempenho, falhas e contabilização.

A linguagem ANEMONA oferece uma interface para o gerente humano definir aplicações de monitoração, apresentando como requisitos básicos simplicidade, concisão e precisão. O gerente humano informa as expressões e eventos a serem monitorados, gerando, em seguida, o código que permite a configuração dos agentes de monitoração. Uma das principais contribuições deste trabalho é oferecer uma opção para o uso das MIB's *Event* e *Expression* consideravelmente mais simples do que a configuração manual.

O restante deste trabalho está organizado da seguinte maneira. As MIB's *Event* e *Expression* são apresentadas na seção 2. A seção 3 descreve a linguagem ANEMONA, bem como a implementação de um tradutor para a mesma. A seção 4 descreve experimentos realizados com duas aplicações práticas do uso da linguagem: a primeira aplicação detecta uma invasão *TCP Syn Flooding* [6] iminente e a segunda aplicação permite a monitoração de enlaces para a detecção e emissão de alarmes frente à ocorrência de um aumento brusco na sua taxa de utilização. A seção 5 conclui o trabalho.

2 Gerência de Redes Distribuída com SNMP

Nesta seção são descritas a *Event MIB* e a *Expression MIB*, propostas pelo DISMAN, para aplicações de gerência de redes distribuída.

2.1 A Expression MIB

A *Expression MIB* [4] permite a construção e avaliação de expressões compostas por objetos de gerência e os valores atribuídos aos mesmos. Os resultados destas expressões ficam disponíveis como objetos MIB, podendo ser usados por uma aplicação de gerência de maneira direta ou serem referenciados por outra MIB, como a *Event MIB*. Estes objetos podem ainda ser usados pela própria *Expression MIB* formando expressões mais complexas.

A *Expression MIB* suporta três tipos de amostragem (coleta de valores) dos objetos que compõem a expressão: *valores absolutos*, *delta* e um *valor booleano* que é verdadeiro quando o valor do objeto monitorado tiver sido modificado.

Um valor absoluto é simplesmente o valor de um objeto no instante em que ele foi amostrado. Este tipo de amostragem não pode ser utilizado para a avaliação de expressões que envolvam contadores, sendo que estes devem ser manipulados como um delta (diferença) entre duas amostras. A amostragem delta define coletas periódicas dos valores amostrados, além de sempre armazenar o último valor a fim de que este possa ser utilizado no cálculo da diferença entre as duas amostras. Por fim, a amostragem por valores booleanos retornará verdadeiro se o valor da amostra atual for diferente do da amostra anterior, ou seja, se o valor do delta para as duas amostras for diferente de zero.

A *Expression MIB* possui duas seções principais: *Definition* e *Value*. A seção *Definition* contém as tabelas que definem as expressões e a seção *Value* contém os resultados das expressões avaliadas.

Na seção *Definition*, uma tabela de expressões contém, para cada entrada, uma expressão, além de outros parâmetros, como o tipo de dado retornado por esta expressão. Uma tabela de objetos contém os parâmetros que se aplicam a cada objeto, individualmente, como o identificador do objeto e tipo de amostragem. A expressão a ser avaliada é uma *string*. As variáveis envolvidas na construção de uma expressão são os próprios objetos de gerência e são expressas como um símbolo dólar (‘$’), seguido de um inteiro, que serve para indexar a entrada correspondente ao objeto referido, na tabela de objetos. Um exemplo de expressão válida é ($1 - $5) * 100, que corresponde à diferença dos valores do primeiro e do quinto objetos da tabela de objetos, multiplicada por 100.

As expressões não podem ser recursivas, porém uma expressão já em uso pode usar resultados de outra expressão, desde que a segunda não possua nenhuma variável que seja o resultado direto ou indireto da sua própria avaliação. Os operadores permitidos são

delimitadores de escopo, aritméticos, lógicos e relacionais cuja sintaxe e semântica é a mesma da linguagem C [7] e quanto aos tipos de dados suportados, estes são os mesmos do SNMP. Também são suportadas algumas funções, entre as quais *counter32*, que força qualquer valor para *Counter32* e a função *exists* que verifica se uma instância de objeto indicada existe, retornando um valor booleano com o resultado. A evocação de uma função se dá pelo seu nome, seguido de parâmetros.

A seção *Value* contém os valores das expressões avaliadas, disponíveis em uma tabela de resultados, que contém uma relação que corresponde aos tipos de dados possíveis para um resultado, dispostos um tipo por coluna. Para uma dada expressão, somente uma coluna será instanciada dependendo do tipo de valor de retorno da expressão, configurado na tabela de expressões.

2.2 A Event MIB

A *Event MIB* [5] permite a monitoração de objetos de gerência em um sistema local ou remoto e toma ações básicas quando uma condição de disparo for atingida, podendo ser complementada pela *Expression MIB*, a qual fornece resultados de expressões a serem monitorados.

A *Event MIB* possui duas seções principais: de gatilhos (*triggers*) e de eventos. Os gatilhos definem as condições que acionam os eventos. Na seção de gatilhos são definidos os objetos monitorados e como relacionar cada gatilho a um evento. Na seção de eventos é determinado o que acontece quando uma condição programada é verificada, que implica no envio de uma notificação, na alteração do valor de um objeto, ou ambos.

Na seção de gatilhos, uma tabela contém informações sobre os *triggers* programados. Para cada entrada é possível especificar o tipo de teste a ser executado, que pode ser *boolean* ou *existence*. Os tipos de teste são descritos abaixo.

O teste do tipo *existence* pode disparar eventos quando a instância do objeto monitorado se tornar presente, ausente ou ainda mudar de valor. Se um evento for acionado quando uma instância de um objeto se tornar presente, esta instância deverá deixar de existir antes que o evento possa ser evocado novamente e *vice versa*. Se um evento for acionado devido a uma mudança de valor do objeto, o mesmo evento será acionado no caso de futuras mudanças de valor, sem maiores restrições.

O teste do tipo *boolean* exige a seleção de um teste específico, além de requerer que o objeto monitorado possua um valor inteiro. Na seção de gatilhos são indicados um valor de

referência inteiro para ser comparado com o valor do objeto monitorado e a comparação a ser executada entre o valor amostrado e o valor de referência, podendo ser diferente, igual, menor, menor ou igual, maior ou maior ou igual. Se o resultado do teste for verdadeiro, um evento será acionado. Este mesmo evento só será disparado novamente quando o resultado do teste passar a ser falso e tornar a ser verdadeiro.

Cada gatilho contém um índice que aponta o evento, na seção de eventos, que deverá ser acionado. Se a ação evocada pelo evento programado for uma notificação, a seção de eventos, deverá possuir informações sobre a notificação correspondente. Quando a ação evocada pelo evento programado for a alteração do valor de um objeto, a seção de eventos deverá conter informações como o identificador do objeto cujo valor será alterado e o novo valor a ser atribuído ao objeto.

2.3 Uso Combinado da Expression MIB e da Event MIB

Conforme exposto anteriormente, a *Event MIB* pode ser usada em conjunto e complementada pela *Expression MIB*, onde os objetos com resultados das expressões definidas através da *Expression MIB* são monitorados pela *Event MIB*. Com este conjunto dispomos de uma ferramenta poderosa para a monitoração de objetos. No entanto, devido a algumas dificuldades de ordem prática, estas ferramentas têm sido pouco utilizadas pelos administradores de redes. Estas dificuldades consistem no complicado processo de configurar as expressões, os gatilhos e os eventos, exigindo uma série de operações *snmpset*, assim como conhecimento detalhado acerca da arquitetura das MIB's envolvidas [8]. O gerenciamento de todas as diversas tabelas das duas MIB's também é problemático, devido à necessidade de exercer um controle efetivo sobre suas entradas e os relacionamentos entre as tabelas. Além de este processo ser trabalhoso, ele está muito sujeito à incidência de erros.

3 Uma Linguagem para Gerência Distribuída de Objetos SNMP

Como alternativa para facilitar a configuração das MIB's *Event* e *Expression* é proposta a linguagem ANEMONA, cujos programas, ao serem traduzidos, geram ações para configurar as MIB's. Um programa da linguagem ANEMONA primeiramente define as entidades de gerência a serem monitoradas. Os *agentes monitorados* são quaisquer dispositivos que executem o software agente do SNMP e possuam em sua base de informações de gerência as MIB's *Event* e *Expression*.

O tradutor da linguagem ANEMONA pode ser executado em qualquer computador que possua acesso físico à rede, execute o software gerente SNMP e tenha permissões de escrita nos objetos de gerência dos agentes monitorados. O administrador especifica as condições de monitoração no programa. Quando um programa é traduzido, uma seqüência de operações *snmpget* e *snmpset* é gerada a fim de configurar as MIB's *Event* e *Expression*, residentes no agente monitorado especificado no programa. Os agentes monitorados, com suas MIB's devidamente programadas, passam a monitorar objetos ou expressões de objetos. Na ocorrência de um evento programado, o agente monitorado gera uma notificação (*trap*) para um gerente especificado ou, opcionalmente, pode executar uma operação *snmpset*, previamente definida sobre um objeto local.

3.1 Definições da Linguagem ANEMONA

Todo programa ANEMONA possui três seções obrigatórias. A primeira delas é o comando *watch*, que indica o *host* que executa o agente a ser configurado. A especificação da comunidade a ser utilizada é obrigatória. A segunda seção dos programas ANEMONA contém as *declarações* dos objetos que serão referenciados no corpo do programa. A última seção contém o corpo do programa propriamente dito, obrigatoriamente precedido de *begin* e terminado por *end*. A estrutura básica de um programa é ilustrada abaixo.

```
watch <monitor> using <comunidade>
<declarações>
begin
        <código>
end
```

3.1.1 Declarações

É necessário declarar os objetos para que a ferramenta possa administrar os seus tipos e ainda determinar como eles são amostrados: valores absolutos (*absolute*), valores delta (*delta*) ou valor booleano verdadeiro se modificado (*modified*). A declaração de um objeto é ilustrada abaixo.

```
<OID> is <tipo>: <tipo de amostragem>
```

Sempre que um objeto for referenciado, a referência a ele se dá pelo seu nome (*OBJECT IDENTIFIER*).

3.1.2 Tipos de Dados

Os tipos de dados suportados são *Integer*, *OctetString*, *OID* (*OBJECT IDENTIFIER*),

IPAddress, *Counter32*, *Unsigned*, *TimeTicks* e *Counter64*. Os tipos escolhidos foram definidos a partir de um subconjunto dos tipos definidos pela SMI [9], onde foram aproveitados aqueles que são usados com maior freqüência nas aplicações de gerência [8].

3.1.3 Operadores

A linguagem suporta operadores aritméticos, relacionais, lógicos booleanos, lógicos bit a bit, de concatenação e condicionais. Toda expressão é finalizada por um ponto e vírgula.

Os operadores aritméticos são: adição '+', subtração '-', multiplicação '*', divisão '/' e resto da divisão inteira '%'. Para operações aritméticas os tipos de operandos suportados são *Integer32*, *Counter32*, *Counter64*, *Unsigned* e *TimeTicks* e o tradutor ANEMONA deve decidir o tipo de retorno de cada operação de acordo com as seguintes regras, em ordem de precedência:

1. Para o operador '-' unário, o resultado será sempre do tipo *Integer32*.
2. Se os operadores esquerdo e direito forem de tipos iguais, o retorno será deste tipo.
3. Se um dos operadores for *Counter64*, o retorno será *Counter64*.
4. Se um dos operadores for *IPAdress*, o retorno será *IPAdress*.
5. Se um dos operadores for *TimeTicks*, o retorno será *TimeTicks*.
6. Se um dos operadores for *Counter32*, o retorno será *Counter32*.
7. Caso contrário, o retorno será *Unsigned*.

Os operadores relacionais comparativos suportados são: igualdade '==', desigualdade '!=', maior que '>', menor que '<', maior ou igual '>=' e menor ou igual '<='. Os operandos podem ser de qualquer tipo e os resultados sempre serão do tipo *Unsigned*, com valor igual a zero caso seja falso e diferente de zero, sendo verdadeiro.

Os operadores lógicos booleanos são a conjunção '*and*', disjunção '*or*' e negação '*not*', sempre grafados em caracteres minúsculos. Os operandos são expressões que resultem um valor *Unsigned* e o retorno sempre será *Unsigned*.

Os operadores lógicos bit a bit são conjunção bit a bit '*AND*', disjunção bit a bit '*OR*', negação bit a bit '*NOT*' e ou exclusivo bit a bit '*XOR*'. O que os diferencia sintaticamente dos operadores lógicos booleanos é o fato de serem escritos com caracteres maiúsculos. Os operandos, ambos do mesmo tipo, podem ser *IPAddress* ou OID e o tipo de valor de retorno possui o mesmo tipo dos operandos.

Para os tipos *OctetString* e OID é possível a concatenação, através do operador '+'. Ambos os operandos devem ser do mesmo tipo e o tipo do resultado será o mesmo dos

operandos.

É suportado um operador condicional, que atribui valores a objetos. Este operador avalia constantemente uma expressão com valor de retorno booleano e, enquanto esta for verdadeira, o objeto referido terá um primeiro valor. Se o resultado da expressão for falso e enquanto permanecer falso, será atribuído um segundo valor a este objeto.

if <expressão> then <valor 1> else <valor 2> rec by <OID / macro>

Por fim, podem ser usados parênteses '(' e ')' como delimitadores de escopo na avaliação de expressões.

3.1.4 Macros

Nessa linguagem, macros são usadas para atribuir um identificador a um nome de objeto ou ao resultado de uma expressão. Esse identificador reduz o trabalho do programador, que não precisará repetir referências longas a objetos, porém a sua função mais importante é identificar entradas da tabela de resultados da *Expression MIB*, para expressões que tenham sido programadas pela ferramenta. Ao encerrar a tradução de um programa, a ferramenta reportará o nome de objeto que está vinculado a cada identificador de macro, a fim de que o administrador possa coletar resultados de expressões intermediárias. Em ANEMONA macros são definidas através do comando *bind*.

bind <identificador> to <expressão>

3.1.5 Comandos Básicos

Os comandos que correspondem às ações básicas que serão tomadas na ocorrência de eventos são *set* e *notify*. *Set* atribui um valor a um determinado objeto. Neste caso, é obrigatório declarar a comunidade que será utilizada e o valor a ser atribuído deverá ser um inteiro, não sendo suportadas expressões, devido a restrições operacionais da *Event MIB* [5]. Já o comando *notify* envia uma *trap* a um dado gerente. Neste caso a comunidade utilizada também deverá ser explicitada além de um objeto que identifica a natureza da *trap*.

set <OID> at <IP> using <comunidade> to <valor inteiro>
notify <IP> <comunidade> <OID>

3.1.6 Triggers

Nesta linguagem, *triggers* são implementados pelo comando *when*. Ele recebe uma

expressão com resultado booleano e executa uma lista com comandos básicos dentre aqueles definidos em 3.1.5. A expressão será programada na *Expression MIB* e para cada comando da lista, será configurado, na *Event MIB*, um gatilho para monitorar o resultado da expressão e um evento para a cada ação a ser tomada. Quando o resultado da expressão for verdadeiro, as ações correspondentes são executadas.

```
when <expressão>
do
        <lista de ações>
end
```

3.1.7 Funções

Para serem avaliadas junto às expressões, da mesma maneira que os operadores definidos anteriormente, são suportadas as funções pré-definidas *counter32* e *exists*. A função *counter32*(<inteiro>) converte qualquer valor inteiro para *Counter32* e a função *exists*(<OID>) verifica se a instância do objeto informada existe e seu retorno é do tipo *Unsigned*. Caso a instância exista, um valor diferente de zero é retornado. Caso não exista é retornado zero.

3.2 Um Tradutor para ANEMONA

Com a finalidade de criar um ambiente de testes para a obtenção de resultados decorrentes do uso da linguagem proposta, foram implementados subconjuntos das MIB's *Event* e *Expression*, uma vez que não foi encontrada nenhuma implementação de uso público disponível destas MIB's. Os subconjuntos implementados estão definidos em [8]. Para a implementação do tradutor foram utilizadas técnicas padronizadas para construção de compiladores, descritas em [10]. As definições da gramática da linguagem ANEMONA estão disponíveis em [8].

4 Estudos de Caso

Durante os testes foram feitos estudos de caso envolvendo situações reais de gerência de redes de computadores nas quais a ferramenta ANEMONA foi aplicada. Estes estudos de caso tratam de programas ANEMONA que geram uma notificação ao administrador quando detectam um possível ataque do tipo *TCP Syn Flooding* [6] ou a saturação da capacidade de do enlace, com base no número de datagramas IP transmitidos e recebidos.

4.1 Detecção de um Ataque TCP Syn Flooding

O ataque conhecido como *TCP Syn Flooding* consiste em provocar uma falha no mecanismo de estabelecimento de conexão do protocolo TCP. Este estabelecimento de conexão é feito através de um *handshake* de três vias, o qual é iniciado com um cliente emitindo um segmento TCP para o servidor com o *flag Syn* setado em seu cabeçalho. Normalmente o servidor responde com um segmento com os *flags Syn* e *Ack* setados para o endereço do cliente especificado no cabeçalho IP. Por fim, o cliente envia um *Ack* para o servidor e a conexão está estabelecida.

O ataque *TCP Syn Flooding* [6] é caracterizado por inundar um dado servidor com segmentos TCP com o *flag Syn* setado em seu cabeçalho, porém com o endereço de origem no cabeçalho IP adulterado, correspondente ao de um *host* inacessível. Desta maneira, quando o servidor receber este segmento, enviará um segundo segmento com os *flags Syn* e *Ack* setados para o endereço constante no cabeçalho IP do primeiro segmento. O segmento transmitido pelo servidor não será respondido, até atingir o tempo limite, não permitindo ao TCP completar o *handshake* de três vias.

Durante um ataque, o *host* atacante envia diversas requisições *Syn* para uma ou mais portas TCP do servidor atacado. O número destas requisições deverá ser suficiente para inundar as filas de requisições, causando a indisponibilidade dos serviços executados nas portas atacadas.

Para a detecção de um provável ataque deste tipo, o objeto de gerência *tcp.tcpAttemptFails* computa o número de vezes que a máquina de estados TCP passa para o estado *CLOSED* a partir de um estado *SYN-SENT* ou *SYN-RCVD*, mais o número de vezes que passa do estado *SYN-RCVD* para o estado *LISTEN*.

Um servidor em funcionamento, aguardando conexões, permanece no estado *LISTEN* até receber uma requisição *Syn*. Ao receber esta requisição, que no caso de um ataque terá o endereço de origem adulterado para um endereço inacessível, o servidor enviará à origem um segmento com os *flags Syn* e *Ack* setados e passará ao estado *SYN-RCVD*, onde aguardará a confirmação do segmento transmitido. Como não haverá esta confirmação, o servidor permanecerá no estado *SYN-RCVD* até atingir o tempo limite, quando então, passará para o estado *CLOSED*, provocando um incremento de *tcp.tcpAttemptFails*.

Para este estudo de caso foi utilizado o programa *Neptune* [6], que gera um número determinado de segmentos adulterados, com endereços de origem e destino e porta fornecidos

pelo usuário.

O instrumental utilizado foi um computador com sistema operacional *Linux*, agente NET-SNMP [11], MIB's *Event* e *Expression*, tradutor ANEMONA e servidores HTTP, FTP e *telnetd* e outro computador com sistema operacional *Linux*, com o serviço *telnetd* e com *Neptune*. Ambos estavam conectados a uma rede *Ethernet* através de interfaces de 10 Mbps.

Como o objeto *tcp.tcpAttemptFails* é um contador, este foi amostrado como delta, com intervalo de 6 segundos, usando a *Expression* MIB e o tradutor ANEMONA.

Durante o funcionamento normal da rede, foram realizadas conexões para os serviços de FTP, HTTP e *telnet*, onde o *host* monitorado agiu como cliente e também como servidor, e o valor de *tcp.tcpAttemptFails.0* manteve-se em zero, constatando que os erros assinalados pelo objeto não são freqüentes.

Tempo de invasão	delta de *tcp.tcpAttemptFails* (intervalo de 6 s)
0 s	0
6 s	0
12 s	170
18 s	300
24 s	301
30 s	300

Figura 1: Ataque TCP Syn Flooding

Foi, então, feito o primeiro ataque, contra o serviço da porta 23 (*telnet*), usando 5000 segmentos e foram colhidos os valores delta constantes na tabela da figura 1, em intervalos de 6 segundos. Após 30 segundos, o valor delta de *tcp.tcpAttemptFails* se estabilizou em 300. Ao final do ataque, o valor absoluto de *tcp.tcpAttemptFails.0* era 4887.

Com base nos resultados destes testes, foi arbitrado como valor crítico 25 para o valor do delta de *tcp.tcpAttemptFails*, para um intervalo de 6 segundos entre as amostras, que seria equivalente a quatro falhas por segundo.

O programa ANEMONA a seguir implementa esta aplicação:

```
watch: denes.cce.ufpr.br using private
tcp.tcpAttemptFails.0 is Counter32: delta
begin
        when tcp.tcpAttemptFails.0 > 25
        do
                notify denes.cce.ufpr.br private tcp.tcpAttempFails.0
        end
end
```

Neste programa, o *host* denes.cce.ufpr.br é monitorado, tendo o objeto

tcp.tcpAttemptFails de sua MIB amostrado como delta. Quando o valor do delta deste objeto for superior a 25, será gerada uma notificação para denes.cce.ufpr.br contendo *tcp.tcpAttemptFails.0*. As ações geradas pela tradução deste programa estão disponíveis em [8].

Com o computador em condições normais de operação, isto é, sem nenhum ataque sendo executado, este programa foi traduzido e, em seguida foram gerados ataques idênticos ao anterior a partir da outra máquina. Em um resultado representativo, após 7 segundos foi recebida uma notificação no *host* monitorado.

O ataque foi detectado no tempo indicado acima, mas mesmo que a invasão fosse abortada logo que descoberta, isto não seria suficiente para impedir a interrupção da continuidade do serviço na porta 23. Isto se deve ao fato de que a máquina de estados TCP somente passará do estado SYN-RCVD para o estado CLOSED após o *timeout*, quando o serviço já terá sido interrompido.

4.2 Detecção da Saturação de um Enlace

Este experimento consiste na geração de uma notificação quando a utilização de um enlace estiver saturada. Para simular uma saturação, a utilização do enlace foi medida com base no número de datagramas IP, sendo utilizado um programa gerador de fluxo, especialmente projetado para este fim, descrito em [8].

Este programa transmite pacotes UDP, com um tamanho mínimo, em grande quantidade, com o objetivo de saturar o enlace. O protocolo escolhido foi o UDP porque, ao contrário do TCP, não possui algoritmos de controle de fluxo.

Os objetos de gerência *ip.ipInReceives* e *ip.ipOutRequests* contabilizam o número de datagramas IP recebidos e transmitidos, respectivamente. O número de datagramas do enlace é dado pela soma dos valores destes dois objetos.

Os recursos de *software* e *hardware* utilizados foram um computador com sistema operacional *Linux*, agente NET-SNMP, MIB's *Event* e *Expression*, tradutor ANEMONA e servidor e cliente do gerador de fluxo UDP e outro computador com sistema operacional *Linux*, com servidor e cliente do gerador de fluxo UDP, conectados a uma rede *Ethernet* através de interfaces de 10 Mbps.

Como os objetos *ip.ipInReceives* e *ip.ipOutRequests* são contadores, foram amostrados como deltas, utilizando intervalo de 6 segundos. O programa ANEMONA a seguir amostra os objetos mencionados como deltas, calcula a sua soma e atribui o resultado a uma instância da

tabela de resultados da *Expression MIB*, denotada por *utilization* e cujo endereço será reportado ao usuário após a tradução do programa.

```
watch: denes.cce.ufpr.br using private
ip.ipInReceives.0 is Counter32: delta
ip.ipOutRequests.0 is Counter32: delta
begin
        bind utilization to (ip.ipInReceives.0 + ip.ipOutRequests.0);
end
```

Os dados obtidos a partir do programa acima foram coletados em duas situações distintas: sem utilização dos recursos da rede e com utilização pesada dos recursos da rede.

Sem fazer uso de aplicações que requeiram recursos da rede, os valores representativos do número de datagramas na rede, dados pela soma dos deltas de *ipInReceives* e *ipOutRequests*, registrados em intervalos de 1 minuto foram 80, 88 e 84, respectivamente.

Para garantir a utilização massiva dos recursos da rede, foram estabelecidas conexões FTP ao servidor residente no computador monitorado. Embora a elevada taxa de utilização de um enlace durante um FTP esteja diretamente relacionada com o tamanho dos datagramas IP, esta aplicação garante um tráfego contínuo de pacotes no sentido servidor-cliente e é relevante para o experimento.

Primeiramente foi estabelecida uma sessão FTP, em seguida duas, três e quatro sessões. A cada nova sessão, foram feitas três leituras dos valores do número de datagramas na rede, dado pela soma dos deltas de *ipInReceives* e *ipOutRequests*, conforme mostrado na figura 2.

	Tempo Decorrente		
Número de Sessões FTP	1 minuto	2 minutos	3 minutos
1 Sessão	2055	2926	3003
2 Sessões	3052	3244	2762
3 Sessões	3686	3318	3837
4 Sessões	3796	3569	3230

Figura 2: Tráfego em sessões FTP

Em seguida, foi monitorada a quantidade de datagramas na rede durante a execução do gerador de fluxo UDP. Primeiramente, foi monitorado o fluxo num único sentido, ou seja, com o servidor rodando no *host* monitorado e cliente no outro computador da rede. Depois foi monitorado o fluxo em dois sentidos, ou seja, com servidores e clientes sendo executados em cada um dos computadores utilizados. A figura 3 mostra os valores representativos do número de datagramas na rede, utilizando intervalo de 6 segundos, coletados de 6 em 6 segundos, para execução do gerador de fluxo em um sentido e nos dois sentidos.

Tempo de Execução	1 sentido	2 sentidos
0 segundos	90	86
6 segundos	16091	96463
12 segundos	55483	57489
18 segundos	81360	62784
24 segundos	65859	125463
30 segundos	78524	124944

Figura 3: Número de Datagramas IP

Em vista dos dados obtidos acima, foi arbitrado 8000 como valor crítico do delta do número de datagramas na rede. O programa a seguir amostra *ip.ipInReceives.0* e *ip.ipOutRequests.0* como deltas, calcula a sua soma, atribuindo o resultado a uma entrada da tabela de resultados da *Expression MIB*, denotada por *utilization*. Quando o valor vinculado a *utilization* for superior ao valor crítico, no caso 8000, é gerada uma notificação para o *host* especificado com a instância de objeto vinculada a *utilization*.

```
watch: denes.cce.ufpr.br using private
ip.ipInReceives.0 is Counter32: delta
ip.ipOutRequests.0 is Counter32: delta
begin
        bind utilization to (ip.ipInReceives.0 + ip.ipOutRequests.0);
        when utilization > 8000
        do
                notify denes.cce.ufpr.br private utilization
        end
end
```

Com o computador em condições normais de operação, isto é, sem estar executando o gerador de fluxo UDP, este programa foi traduzido e, em seguida foi evocado o servidor do gerador de fluxo e, em um outro computador conectado na rede, o seu cliente. Em um experimento representativo, após 19 segundos foi recebida uma notificação no *host* monitorado.

As ações geradas pela tradução dos programas contidos nesta seção estão disponíveis em [8], onde o leitor pode comparar os comandos necessários para a programação manual da aplicação com os programas em ANEMONA.

5. Conclusão

A linguagem ANEMONA oferece ao usuário uma interface de alto nível para especificar o comportamento da *Expression MIB* e da *Event MIB*, permite a utilização das duas MIB's em conjunto e gera automaticamente ações para configura-las, a partir de uma mesma especificação.

A ferramenta proposta torna viável o uso, individual ou combinado, das MIB's *Event* e *Expression*, que até então possuíam elevada complexidade de configuração. Para demonstrar a viabilidade desta abordagem foi construída uma ferramenta prática, baseada no pacote NET-SNMP, cujas partes integrantes são um subconjunto da *Expression MIB*, um subconjunto da *Event MIB* e um tradutor para ANEMONA.

Foram ainda, realizados testes exaustivos, para os quais foram elaborados programas em ANEMONA. Foram também realizados estudos de caso com aplicações que podem detectar ataques iminentes e a saturação da capacidade de um enlace. Estes testes e estudos de caso mostraram não apenas a viabilidade da ferramenta como as vantagens no seu uso, se comparadas às ações executadas diretamente contra as MIB's *Event* e *Expression*.

A partir dos testes da linguagem, pode-se concluir que o seu uso reduz a incidência de erros (devido ao fato de minimizar a interação com o usuário, durante a fase de configuração), exonera o usuário de exercer o controle sobre as diversas tabelas das MIB's e o poupa do trabalho meramente repetitivo de atribuir valores a objetos de gerência. Ao invés disso, o usuário faz uso de uma interface de alto nível, fornecida pela linguagem ANEMONA.

Quanto ao desempenho da ferramenta, é possível afirmar que as ações geradas por ela são similares àquelas que seriam geradas pela configuração manual das MIB's, porém esperando-se uma menor incidência de erros e maior rapidez, aumentando a produtividade do administrador de redes.

Como trabalho futuro, devem ser agregadas ao conjunto as MIB's do DISMAN *Management Target* [12] e *Notification* [13], possibilitando que a *Event MIB* possa monitorar expressões de objetos em *hosts* remotos.

Referências Bibliográficas

[1] STALLINGS, W. *SNMP, SNMPv2, SNMPv3, RMON and RMON2: Pratical Network Management*. Addison-Wesley, third edition, 1998.

[2] CASE, J.; MUNDY, R.; PARTAIN, D. and STEWART, B. Introduction to Version 3 of the Internet-Standard Network Management Framework. *Request for Comments 2570*, 1999.

[3] Distributed Management Working Group (disman) Home Page. Disponível em: <http://www.ietf.org/html.charters/disman-charter.html>

[4] KAVASSERI, R.; STEWART, B. Distributed Management Expression MIB. Request for Comments 2982, October, 2000.

[5] KAVASSERI, R.; STEWART, B. Event MIB. Request for Comments 2981, October 2000.

[6] “Project Neptune”. Phrack Magazine.Volume Seven. Issue Forty-Eight. July, 1996.

[7] KERNIGHAN, B. W. and RITCHIE, D. M. The C Programming Language. Englewood Cliffs: Prentice Hall, 1978.

[8] FERNANDES, H. *ANEMONA: Uma Linguagem de Configuração para Aplicações de Monitoração de Redes*. Dissertação (Mestrado em Informática) – Departamento de Informática, Universidade Federal do Paraná, 2001.

[9] MCCLOGHRIE, K.; PERKINS, D. and SCHOENWAELDER, J. Structure of Management Information Version 2 (SMIv2). Request for Comments 2578, April 1999.

[10] AHO, A.; SETHI, R. e ULLMAN, J. Compiladores, princípios, técnicas e ferramentas. Editora LTC, 1995.

[11] The NET-SNMP Project Home Page. Disponível em: <http://net-snmp.sourceforge.net>

[12] LEVI, D.; MEYER, P. and STEWART, B. SNMP Applications. Request for Comment 2573, April 1999.

[13] KAVASSERI, R. Notification Log MIB. Request for Comments 3014, November 2000.