Monitorização e Visualização de Recursos num Ambiente Distribuído

Pedro Miguel Costa, Maurício Frade Domingues,
Joaquim Arnaldo Martins, José Luís Oliveira

***Resumo*- Esta apresentação reporta um conjunto de estudos efectuados sobre as funcionalidades de determinadas interfaces, ao nível físico e lógico para receber ou enviar informação de gestão sobre um meio de comunicação. Com esse intuito serão discutidas as potencialidades do *Packet Filter* e *Data Link Interface* (DLI).**

Apresenta-se igualmente um trabalho, com base no *Simple Network Management Protocol* (SNMP), relacionado com a *Remote Monitoring* MIB, no qual se irá descrever a implementaçao de dois dos seus grupos, nomeadamente os grupos *alarm* e *event.*

***Abstract*- This presentation reports a number of studies about the functionalities of some commercial interfaces, at physical and logical levels, in order to receive or send management information through a communication medium. With this purpose we will be discuss the potentialities of both de Packet Filter and Data Link Interface (DLI) .**

We will also present the Simple Network Management Protocol (SNMP) and a work related with the Remote Monitoring MIB, namely the implementation of two of its groups, the "alarm" amd "event" groups.

I. INTRODUÇÃO*

Começamos por fazer um estudo das potencialidades de duas interfaces existentes no sistema operativo ULTRIX, Packet Filter e Data Link Interface (DLI) tendo em vista a apresentação de um sistema capaz de recepção e transmissão de pacotes que possa dar resposta às necessidades impostas pelos vários grupos que compõem a RMON-MIB - *Remote Monitoring Management Information Base*.

O protocolo de comunicação Simple Network Management Protocol já deu provas de ser uma solução real para a troca de informação de gestão entre sistemas heterogéneos, isto é, de fabricantes diferentes. O protocolo SNMP, não só por estar disponível, mas também por ser muito simples, tal como o próprio nome indica, é de momento o que mais se aplica à criação de sistemas de gestão e monitorização de redes. Aqui irá ser feito um estudo sobre este protocolo, e em seguida descrever-se-á uma implementação de dois grupos da RMON-MIB: *alarm* e *event*, responsáveis, respectivamente, pela monitorização de variáveis e geração de eventos aquando da ocorrência de determinadas condições pré-definidas.

Finalmente tecer-se-ão algumas considerações respeitantes à implementação destes dois grupos.

* Trabalho realizado no âmbito da disciplina de projecto.

II. PACKET FILTER

O Packet Filter [1] é um pseudo-device driver, suportado pelo sistema operativo ULTRIX, para criação de canais de acesso a redes do tipo Ethernet e semelhantes.

A estrutura de software do *Packet Filter* é ilustrada na Fig. 1. Observando esta estrutura verificamos que as os componentes principais são o *device driver* e o *Packet Filter* que fazer parte do kernel do sistema operativo.

O protoloco RARP (`rarpd`) está aqui ilustrado como exemplo de utilização deste mecanismo. Utiliza a interface para ler e escrever pedidos RARP trabalhando directamente ao nível de rede. Um determinado filtro ao detectar pacotes RARP, passa-os para o *daemon* `rarpd` para posterior processamento.

A *Monitor aplication* diz respeito a outras aplicações definidas pelo utilizador, como por exemplo aplicações de monitorização do tráfego da rede.

A. *Configuração do Packet Filter*

Para que se possa criar uma aplicação com base no packet filter é necessário reconfigurar o kernel, da seguinte forma:

1. Adicionar as seguintes entradas na file de configuração do sistema, existente no directório `/sys/conf/{mips,vax}/HOSTNAME:`

```
optionsPACKETFILTER
.
.
.
pseudo-device packetfilter
```

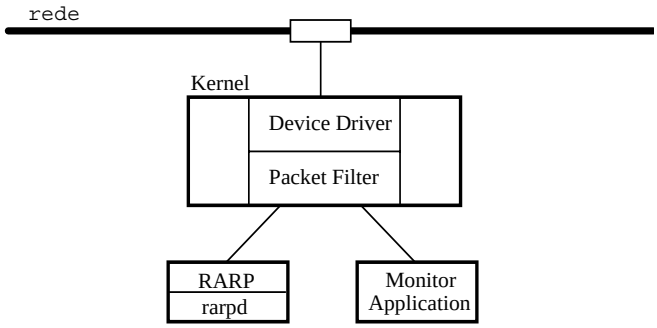


Fig.1 - Estrutura de software do Packet Filter

2. Reconstituir o kernel com o comando `/etc/doconfig -c`.

3. Verificar se `/dev` inclui o directório `pf`. Caso não contenha, terá de se executar o comando `MAKEDEV` com a opção `pfilt`. Este comando cria 64 *packet filter character special files* em `/dev/pf`, com os nomes `pfilt`**`n`**, em que **`n`** representa o número da file.

B. Programação do Packet Filter

Cada processo de aplicação do packet filter faz uso de uma ou mais *character special files* para obter acesso à rede.

A abertura de cada uma destas *special files* é feita usando a rotina `pfopen`. A este procedimento está associado um programa implementado pelo utilizador, que permite configurar o modo de recepção e leitura dos pacotes, bem como seleccionar o tipo de pacotes que se desejam receber, através de um mecanismo de filtragem.

Os pacotes capturados pela interface associada à aplicação em causa, permanecem numa fila de entrada de tamanho limitado, até que sejam lidos. Se esta fila de entrada se encontrar completa, então os pacotes são removidos.

A leitura de pacotes é feita por meio da *system call* `read` que lê os pacotes pela ordem em que se encontram na fila de entrada.

O envio de pacotes para a rede, por meio das *special files*, é feito utilizando a *system call* `write`, o qual envia exactamente um pacote, cada vez que é chamado. A construção do pacote é da responsabilidade da aplicação.

C. Filtragem de pacotes

A filtragem de pacotes é feita através de uma linguagem específica deste sistema. Cada lista de comandos do filtro especifica uma sequência de acções, que colocam um valor lógico no topo de um stack interno. Cada palavra da lista de comandos especifica uma acção e um operador binário.

As acções são: PUSHLIT, coloca a próxima palavra no stack; PUSHZERO, coloca um zero no stack; PUSWORD+N coloca a palavra N do pacote de entrada no stack. Os operadores põem o resultado da operação de dois elementos no topo do stack e são: EQ, LT, LE, GT, GE, AND, OR, XOR.

A associação de um dado filtro a uma *character special file* previamente aberta é feita por intermédio da *system call* `ioctl` da seguinte forma:

```
ioctl(fdes,EIOCSETF,filtro)
struct enfilter *filtro

struct enfilter
{
 u_char enf_Priority;
        // prioridade do filtro
 u_char enf_Filterlen;
        // comprimento do filtro
        // em shortwords
 u_short enf_Filter[ENMAXFILTERS];
        // lista de comandos do filtro
}
```

Um exemplo de um filtro destinado a aceitar pacotes do tipo IP provenientes da rede Ethernet 10Mb (Fig. 2) é apresentado de seguida.

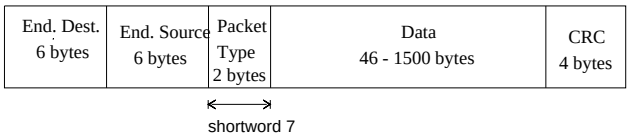


Fig. 2 - Pacote Ethernet 10Mb

```
struct enfilter filtro =
{
 10,1,// prioridade, número de
      // swortwords
      // utilizados na filtragem
 ENF_PUSHORD + 6,
      // Coloca a shortword 6 do
      // pacote de entrada no stack
 ENF_PUSHLIT, 0x0008,
      // Coloca a shortword 7 no
      // stack e compara-a
      // com o tipo IP (0x0800)
 ENF_EQ
      // Se for igual aceita
      // o pacote
}
```

D. Recepção em modo promíscuo

A recepção de pacotes em modo promíscuo possibilita ao utilizador a leitura de todo o tipo de pacotes circulantes na rede. Para isso é necessário realizar os seguintes passos:

1. Configurar o sistema com o comando `pfconfig` da seguinte forma:
   `#pfconfig +promisc [interface-name]`
2. No programa de aplicação utilizar o *system call* `ioctl` com os seguintes parâmetros:
   `ioctl(fdes,EIOCMBIS,ENPROMISC);`

Desta forma, se a *packet filter descriptor* (fdes) estiver aberta, a interface associada ficará programada para recepção em modo promíscuo.

E. Campos de informação

O packet filter possui ferramentas que nos permitem obter vários tipos de informação.

Através do pedido `ioctl(fdes,EIOCMBIS, ENTSTAMP)` cada pacote recebido vem precedido por um *header*, que está armazenado numa estrutura `enstamp` que contém os seguintes campos de informação:

- **`ens_stamplen`** - indica o comprimento da estrutura.

- **`ens_flags`** - contém informação relativa ao modo de recepção de cada pacote: modo promíscuo, *broadcast, multicast ou trailer encapsulation*.
- **`ens_count`** - Contém o comprimento do pacote em bytes, não incluindo o *header*.
- **`ens_dropped`** - Número de pacotes perdidos devido à fila de entrada se encontrar completa. Esta variável é um contador acumulativo desde que a `enstamp` anterior tenha sido lida da mesma *packet filter special file*.
- **`ens_ifoverflows`** - Número total de *overflows* de entrada desde que o sistema foi inicializado.
- **`ens_tstamp`** - Hora aproximada em que o pacote foi recebido.

Outros tipos de informação encontram-se nas seguintes estruturas:

- **`endevp`** - contém dados relativos á interface, que está ligada a *packet filter special file*, os quais podem ser acedidos através do comando `pfsat(8)`.
- **`eniocb`** - fornece-nos parâmetros sobre a *character special file* que está a ser utilizada.

F. Conclusões

O acesso à informação, aliado à possibilidade de recepção em modo promíscuo, faz com que o Packet Filter seja uma ferramenta bastante útil, no que diz respeito a análise global de tráfego da rede.

O facto do seu código residir no kernel confere-lhe um desempenho bastante aceitável mas simultaneamente insuficiente se pretendermos construir aplicações de monitorização em modo promíscuo. Com uma taxa de recepção de pacotes máxima de, aproximadamente, de 112 Kbytes/s (~1Mbit/seg) o *Packet Filter* revela-se com pouca capacidade para acompanhar taxas de utilização acima dos 10%. Contudo, recentes versões do *Packet Filter* têm surgido sendo de referir a *Network Interface Tap* (NIT) [2] e o *BSD Packet Filter* (BPF) [3] com desempenhos superiores à versão estudada mas mesmo assim incapazes de captar toda a informação se a taxa de utilização rondar os valores máximos admissiveis.

III. DATA LINK INTERFACE (DLI)

A DLI [4] é uma interface de nível lógico que permite o desenvolvimento de protocolos de comunicação entre diferentes sistemas.

A. Serviços suportados pela DLI

Ao nível da camada dois a DLI suporta os seguintes serviços:

- Tipo de transmissão: datagrama.
- Formato da trama: Ethernet ou IEEE 802-3.
- ISO 8802-2: Ao nível da subcamada superior do nível 2, LLC, permite serviços Classe I, Tipo I *(connectionless).*
- Endereços: Permite o uso de endereços *multicast.*

B. Serviços a suportar pela aplicação

A DLI, sendo uma interface que acede directamente à camada 2, não fornece serviços próprios de camadas mais elevadas. Deste modo, as aplicações desenvolvidas com base na DLI devem assegurar os seguintes serviços:

- Encaminhamento de pacotes
- Controlo do fluxo dos pacotes
- Recuperação de erros
- Segmentação da informação a transmitir

C. Desenvolvimento de aplicações

O desenvolvimento de aplicações exige um conhecimento profundo da system call *socket*, e de todas as que lhe estão associadas : `socket, bind` e `close`, relacionadas com o canal de transmissão; `write, send` e `sendto`, relacionadas com a transmissão de pacotes; `read, recv` e `recvfrom`, relacionadas com a recepção de pacotes; `setsockopt`, relacionada com as opções da socket.

A DLI fornece uma estrutura de dados através da qual é possível configurar os serviços necessários à comunicação ao nível da camada dois. A estrutura `sockaddr_dl` é usada para transferir informação para a DLI quando quando uma aplicação executa a primitiva `bind`, ou quando transmite um pacote para a rede. A DLI usa também esta estrutura para transferir informação para a aplicação quando esta recebe um pacote da rede. No programa do utilizador esta estrutura irá conter todos os dados necessários à execução do `bind()`, nomeadamente:

- Familia de endereços: AF_DLI.
- Device: Identificação do device utilizado (por exemplo ln0).
- Formato da trama: Ethernet ou 802.
  De acordo com o especificado neste campo, será utilizada a estrutura correspondente: `sockaddr_edl` (que será aqui considerada) ou `sockaddr_802`.
  Na estructura `sockaddr_edl` há a considerar os seguintes campos :

  - A Flag `(dli_ioctlflg)` que especificará o modo como os pacotes serão filtrados. Os modos possíveis são :
    NORMAL - Permite ao programa a troca de pacotes com apenas um nó Ethernet. Com esta *flag* activa o endereço da máquina destino deve ser especificado à partida, de modo a ser utilizado pela *system call* `bind()` na conecção à socket. A aplicação receberá todos os pacotes com o seu endereço, desde que o endereço da fonte e o protocolo sejam os especificados na `bind()`. Duas aplicações com esta *flag* activa comunicarão exclusivamente uma com a outra .

Neste modo de operaçao a aplicação poderá utilizar todas as *system calls* para transferência de informação referidas acima.

EXCLUSIVE - Dá ao programa uso exclusivo do protocolo especificado, permitindo-lhe comunicar com qualquer outro programa que utilize o mesmo protocolo. Ao contrário do modo NORMAL, o envio e recepção de mensagens só é possível através da utilização das *system calls* `recvfrom()` e `sendto()`. Como na chamada da `bind()` o endereço destino não é especificado (sendo preenchido com um NULL), dado que vários destinos e várias fontes são permitidos, o programa terá que usar `recvfrom()` para saber o endereço da fonte de cada pacote, usando depois esse endereço para enviar uma resposta através da *system call* `sendto()`.

DEFAULT - Quanto ao uso das *system calls* é similar ao caso anterior. Também como no caso anterior, só o protocolo é verificado. A diferença fundamental é que o programa só recebe o pacote se mais nenhum outro o aceitar, ou seja, se o protocolo não for exclusivo de outro programa (que tenha esta flag a EXCLUSIVE), ou o par endereço/protocolo não coincida com o de outra aplicação (*flag* a NORMAL).

- Tipo de protocolo utilizado.
- Endereço físico da máquina destino.

Quando não especificado, (`ioctlflg` EXCLUSIVE ou DEFAULT) o programa utiliza `recvfrom()` para determinar o endereço fonte de cada pacote, utilizando esse mesmo endereço para, através de `sendto()`, enviar uma resposta.

Para o desenvolvimento de aplicações é necessário que o utilizador tenha permissões de administrador do sistema.

Esta interface não é a mais adequada à análise de tráfego da rede por não permitir a recepção de pacotes em modo promíscuo.

IV. SIMPLE NETWORK MANAGEMENT PROTOCOL (SNMP)

O SNMP [5] é um protocolo de gestão de redes TCP\IP. Foi desenvolvido pela comunidade Internet e cedo revelou uma grande aceitação por parte dos fabricantes. A sua relativa simplicidade levou ao recente aparecimento de uma segunda versão conhecida por SNMPv2 que se espera vir colmatar algumas das lacunas da sua antecessora versão, nomeadamente ao nível da segurança.

Numa gestão baseada no protocolo SNMP existem basicamente quatro componentes:

- Os agentes que representam dispositivos ou equipamentos geridos.
- Os gestores - NMS (Network Management Stations) os quais são regra geral computadores centrais que fazem a gestão da rede.
- O protocolo que descreve a forma como os gestores e os agentes comunicam entre si.
- O tipo e o formato da informação trocada (MIB [6] e SMI [7]).

O SNMP faz uso do protocolo de transporte UDP sobre IP -*User Datagram Protocol/Internet Protocol* - o qual não requer o estabelecimento de uma associação lógica entre sistemas. Como consequência, é necessária a identificação (endereço IP + porta UDP) do emissor e do destinatário em cada pacote enviado. Embora apresente inconvenientes, como por exemplo ausência de confirmação de recepção, é utilizado na implementação de diferentes redes dada a sua simplicidade.

Para que uma NMS interactue com vários agentes, o protocolo SNMP suporta vários PDUS (Protocols Data Units) os quais permitem ao gestor, não só obter valores de variáveis (Get) mas também, alterar esses valores (Set). Por outro lado, os agentes além de responderem aos pedidos de informação vindas do gestor, informam-no da ocorrência de situações anómalas através de mensagens não solicitadas (traps).

A Fig. 3 representa a arquitectura de um agente e um gestor SNMP, na qual se pode visualizar o conjunto de serviços disponível [8].

O standard SNMP especifica como as mensagens SNMP são transportadas sobre UDP/IP. Contudo as mensagens SNMP têm a possibilidade de serem transportadas sobre qualquer mecanismo de transporte que suporte trocas bidireccionais e apresente capacidades de endereçamento. Como prova disso encontram-se implementações SNMP sobre TCP, Ethernet, LLC, Appletalk, etc.

Sob o ponto de vista de organização administrativa, o SNMP recorre ao conceito de comunidade para obter autenticação de acesso entre o gestor e o agente.

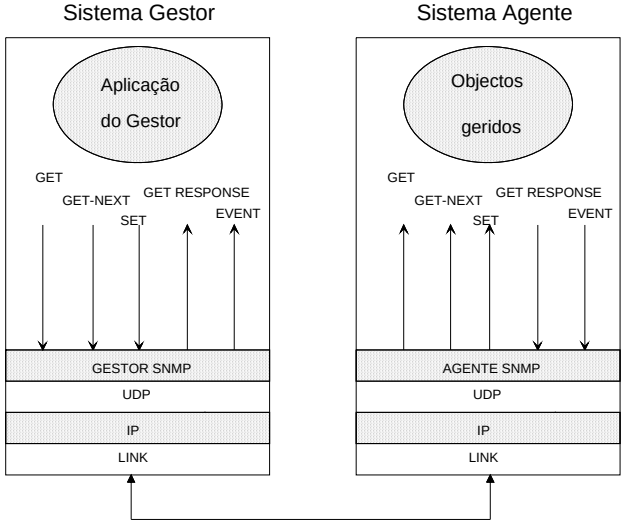


Fig. 3 - Comunicação Agente-Gestor SNMP

A. Management Information Base (MIB)

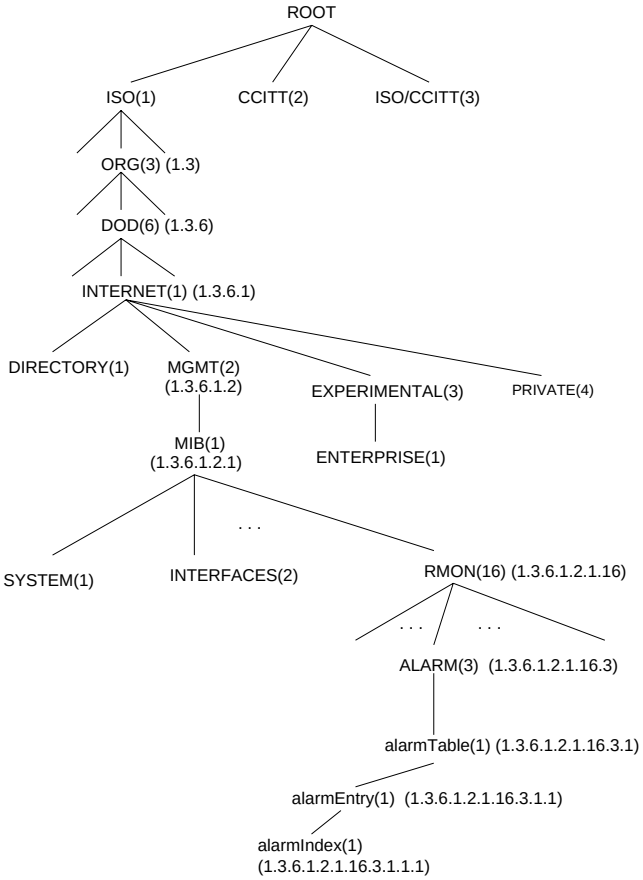
A MIB associada ao protocolo SNMP está organizada como uma base de dados hierárquica, surgindo assim, como uma árvore n-ária em que cada folha representa um objecto.

Cada agente contém uma estrutura de informação normalmente designada por MIB que, não é mais do que uma lista de variáveis.

A informação de gestão é estruturada em objectos, os quais apresentam determinados atributos. Cada objecto terá de possuir:

- Um nome que o identifica inequivocamente.
- Uma sintaxe, a qual é descrita pelo recurso à linguagem ASN.1.
- Uma codificação, isto é, o modo como o objecto é identificado quando da sua transmissão pela rede - identificador .

Como se pode ver na Fig. 4 a partir da raiz (root) da árvore há três nós administrativos distribuídos respectivamente pela ISO, pelo CCITT e um terceiro em simultâneo pela ISO e CCITT. Toda a subdivisão da árvore a partir daí, é da inteira responsabilidade do organismo que a originou. Como exemplo, é ilustrada a localização do objecto *alarmIndex* pertencente ao grupo *alarm* da RMON-MIB.



V. REMOTE MONITORING MANAGEMENT INFORMATION BASE (RMON MIB)

A RMON MIB [9] foi a primeira tentativa da comunidade Internet para normalizar os monitores de redes baseados em TCP/IP. Permite simplificar técnicas de diagnóstico de falhas na rede e de planeamento, assim como um apuramento do desempenho global do processo de monitorização.

A RMON MIB está organizada em nove grupos :

- **statistics** - armazena dados estatísticos com base no número de pacotes, bytes, *broadcasts*, *multicasts*, erros e colisões ocorridos num segmento local da rede Ethernet.
  Este grupo está organizado numa única tabela: *etherStatsTable*.
- **history** - este grupo faz periodicamente uma análise da rede. Os dados obtidos são armazenados para posterior utilização.
  Este grupo está organizado em duas tabelas: *historyControlTable* e *etherHistoryTable*.
- **alarm** - obtém amostras do valor de variáveis e compara-os com limiares pré-definidos, gerando eventos quando estes são ultrapassados. Requer a implementação do *event group.*
  Este grupo está organizado numa única tabela: *alarmTable*.
- **host** - contém dados relativos a todas as máquinas da rede (por exemplo pacotes enviados e recebidos, *multicasts*, *broadcasts* ou pacotes com erros enviados). Este grupo mantém actualizada uma lista das máquinas que fazem parte da rede através da leitura dos campos de endereços dos pacotes que circulam na rede (só os pacotes sem erros são considerados).
  O grupo está organizado em três tabelas: *hostControlTable*, *hostTable* e *hostTimeTable*.
- **hostTopN** - utilizando os dados estatísticos armazenados pelo grupo Host cria listas ordenadas de máquinas pertencentes à rede de acordo com parâmetros a definir pelo gestor, nomeadamente : interface, momentos inicial e final do período de amostragem, número de máquinas e o dado estatístico que funcionará como chave para a ordenação.
  Este grupo está organizado em duas tabelas: *hostTopNControlTable* e *hostTopNTable*. Requer a implementação do grupo Host.
- **matrix** - armazena informação relativa a comunicações entre pares de endereços. Sempre que uma nova comunicação é detectada, é criada uma nova entrada na respectiva tabela. A informação pode ser ordenada de acordo com os endereços fonte ou destino.
  Este grupo está organizado em três tabelas: *matrixControlTable*, *matrixSDTable* e *matrixDSTable*.
- **filter** - permite filtrar pacotes por meio dos seus bits constituintes e/ou do seu *status* . Um pacote que passe o teste de filtragem pode ser capturado ou gerar um evento.
  Este grupo está organizado em duas tabelas: *filterTable* e *channelTabel*.
- **packet capture** - permite capturar pacotes que tenham sido previamente aceites pelo grupo Filter e direccionados para um canal. A função deste grupo é controlar a saída de vários desses canais.
  Está organizado em duas tabelas: *bufferControlTable* e *captureBufferTable*. Requer a implementação do grupo Filter.
- **event** - controla a geração e notificação de eventos.

Está organizado em duas tabelas: *eventTable* e *LogTable*.

A adaptação da RMON MIB para outras redes, que não só a Ethernet, tem sido objecto de estudo. Recentemente surgiu um documento [10] que define já a TR RMON MIB, ou seja uma RMON para redes Token Ring esperando-se uma futura extensão a redes FDDI.

Dos nove grupos acima referidos, dois deles irão ser tratados em pormenor, nomeadamente os grupos *alarm* e *event.*

A. Grupo alarm

Este grupo destina-se à monitorização de variáveis de grupos pertencentes, ou não, à RMON MIB.

Para um conhecimento dos objectos que compõem este grupo apresenta-se de seguida, a tabela I que refere o nome, sintaxe e tipo de acesso correspondente a cada objecto.

TABELA I
OBJECTOS CONSTITUINTES DO GRUPO *ALARM*

OBJECT-TYPE	SYNTAX	ACCESS
alarmTable	SEQUENCE OF AlarmEntry	not-accessible
alarmEntry	AlarmEntry	not-accessible
alarmIndex	INTEGER (1..65535)	read-only
alarmInterval	INTEGER	read-write
alarmVariable	OBJECT IDENTIFIER	read-write
alarmSampleType	INTEGER	read-write
alarmValue	INTEGER	read-only
alarmStartupAlarm	INTEGER	read-write
alarmRisingThreshold	INTEGER	read-write
alarmFallingThreshold	INTEGER	read-write
alarmRisingEventIndex	INTEGER (1..65535)	read-write
alarmFallingEventIndex	INTEGER (1..65535)	read-write
alarmOwner	OwnerString	read-write
alarmStatus	EntryStatus	read-write

Durante a monitorização, o valor do objecto que está a ser monitorizado, definido em *alarmVariable,* é lido periodicamente de acordo com o valor do objecto *alarmInterval* e em seguida comparado com os limiares previamente definidos em *alarmRisingThrehold* e *alarmFallingThreshold*. Quando um desses limiares for ultrapassado desencadeia-se uma acção específica, cuja execução é da responsabilidade do grupo *event*, sendo por isso requerida a sua implementação.

É de referir que apenas poderão ser monitorizadas variáveis existentes na implementação do agente em causa e do tipo INTEGER, Gauge, Counter ou TimeTicks.

B. Grupo event

Os objectos pertencentes a este grupo encontram-se na tabela II, onde se indica a sintaxe e o tipo de acesso correspondentes a cada objecto. Note-se que este grupo é constituído por duas tabelas, nomeadamente *eventTable* e *logTable*.

TABELA II
OBJECTOS CONSTITUINTES DO GRUPO *EVENT*

OBJECT-TYPE	SYNTAX	ACCESS
eventTable	SEQUENCE OF EventEntry	not-accessible
eventEntry	EventEntry	not-accessible
eventIndex	INTEGER (1..65535)	read-only
eventDescription	DisplayString (Size (0..127))	read-write
eventType	INTEGER	read-write
eventCommunity	OCTET STRING (Size (0..127))	read-write
eventLastTimeSent	TimeTicks	read-only
eventOwner	OwnerString	read-write
eventStatus	EntryStatus	read-write
logTable	SEQUENCE OF LogEntry	not-accessible
logEntry	LogEntry	not-accessible
logEventIndex	INTEGER (1..65535)	read-only
logIndex	INTEGER	read-only
logTime	TimeTicks	read-only
logDescription	DisplayString (Size (0..255))	read-only

Este grupo controla a geração e notificação de eventos. A notificação pode ser efectuada de duas formas : logs e/ou SNMP traps. O registo (*log*) de ocorrências de um determinado evento é efectuado pelo agente criando uma entrada na tabela de logs a ele associada. Se o gestor pretender fazer o registo dos eventos ocorridos, à medida que eles forem sucedendo serão criadas novas entradas na tabela de logs correspondentes aos eventos. O número de entradas na tabela de logs relativas ao mesmo evento deverá ser controlado de acordo com a quantidade de memória disponível. Nesta implementação apenas as últimas cinquenta entradas correspondentes a um dado evento estarão disponíveis.

Este grupo está intimamente relacionado com os grupos *alarm* e *filter*. Sem a implementação de pelo menos um deles, a implementação do Event group não tem qualquer utilidade.

O envio de notificações é efectuado quando o objecto *eventType* possui o valor 3 ou 4.

VI. NOTAS REFERENTES À IMPLEMENTAÇÃO DOS GRUPOS *ALARM* E *EVENT*

A. *Descrição do funcionamento da monitorização*

O diagrama de blocos apresentado na Fig. 5 esquematiza as operações básicas executadas no processo de monitorização.

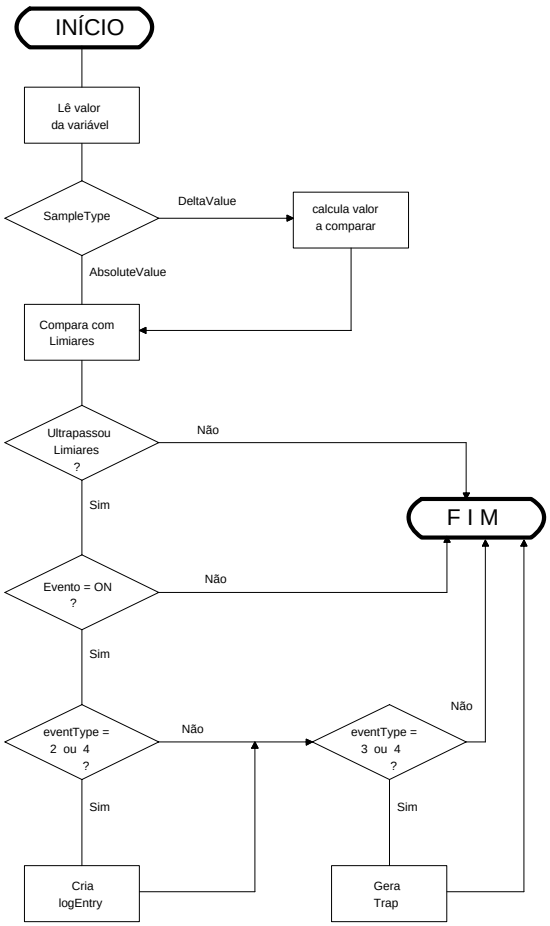


Fig. 5 - Fluxograma do processo de monitorização de uma variável

A rotina que implementa a monitorização é responsável pela realização das seguintes operações:

- Obter o valor da variável, através de um get interno, cujo identificador (OID) está contido em *alarmVariable.*
- Verificar se o mecanismo de histerese, regulador da geração de eventos, já foi inicializado. Este mecanismo consiste em não permitir que dois eventos consecutivos do mesmo tipo sejam gerados. A inicialização deste mecanismo é feito através do objecto *alarmStartUpAlarm*, cujo valor determina qual o tipo de evento que deverá ser gerado após uma entrada ter sido validada.
- Calcular o valor a comparar com os limiares. Para isso testa o valor de *alarmSampleType:* se for igual a *AbsoluteValue(1)* o valor lido é o valor a comparar, se for *DeltaValue(2)* serão feitas duas amostragens em cada período definido por *alarmInterval.* O valor a comparar será o módulo da diferença entre o valor actual e o valor lido na amostragem anterior.
- Comparar o valor obtido com os limiares pré-definidos. Se o valor a comparar for maior/menor ou igual a *alarmRisingThreshold/alarmFallingThreshold* e houver permissão para gerar eventos será gerado um evento cujo índice corresponde ao valor de *alarmRisingEventIndex/ alarmFallingEventIndex.*

B. Criação de entradas nas tabelas

É de todo o interesse que o gestor possa criar novas entradas nas tabelas relativas a alguns grupos. Um número fixo de entradas criadas pelo agente, que poderão ser oportunamente preenchidas pelo gestor, revela-se um processo pouco eficiente, pois é difícil prever as necessidades futuras. O esquema definido para a implementação de grupos na RMON MIB [11] possibilita ao gestor criar dinamicamente novas entradas em tabelas. Este esquema utiliza o valor 2 (under create) do objecto de status da respectiva entrada, o qual permite ao gestor requerer a criação de uma nova entrada.

Por razões de gestão de memória, o agente deverá estabelecer um limite máximo para o número de entradas. Se esse limite for atingido e, em seguida, for feito um pedido de criação de uma nova entrada, fazendo um *set* do status a dois, o agente deverá responder com noSuchName.

C. Eliminação de entradas nas tabelas

Uma entrada pode ser eliminada da tabela atribuindo ao objecto status respectivo o valor 4, através de uma operação de set.

Sempre que o status de uma dada entrada permanecer com o valor 3 (under create) durante um tempo que exceda um período previamente definido por quem implementou o agente, a entrada correspondente será apagada. Este procedimento evita que entradas estejam activas sem que delas se tire qualquer proveito, libertando assim recursos do sistema.

D. Inicialização dos objectos

O documento que especifica a RMON MIB define valores iniciais para apenas um número muito reduzido de objectos. Por esta razão, cabe ao implementador decidir qual o valor de certos objectos no momento em que são criados. Essa escolha tem como base apenas o respeito pela sintaxe dos objectos, já que a diversidade de comportamentos das variáveis que podem ser monitorizadas, afasta a hipótese da utilização de valores típicos para todos os objectos. A estratégia seguida nesta implementação consistiu em, por um lado atribuir um conjunto de valores que permitisse ao gestor efectuar o mínimo de operações de set, e por outro não lhe permitir validar a entrada sem antes configurar um determinado número de objectos. Por esta razão, serão feitos alguns comentários aos valores utilizados.

- **alarmInterval** - valor inicial: 2 (segundos)
  Como o número de entradas activas permitidas nesta implementação é de 50, um período de amostragem menor poderia provocar uma sobrecarga do agente.
- **alarmVariable** - valor inicial: 0.0

Esta variável contém o *Object Identifier* da variável a ser monitorizada. Como é óbvio, este valor não corresponde a nenhum objecto da RMON MIB. Este valor apenas pretende elucidar quanto ao tipo de sintaxe do objecto. Por não ser um valor válido, o agente só permite que o status desta entrada seja posto a *valid* depois deste objecto ser correctamente preenchido. O mesmo não é aplicável aos outros elementos da mesma entrada, já que todos eles são inicializados com valores válidos.

- **alarmSampleType** - valor inicial: 1 (absoluteValue)
  Este valor é utilizável para variáveis do tipo INTEGER e Gauge.
- **alarmValue** - valor inicial: 0 (read only)
- **alarmStartupAlarm** - valor inicial: 3
  Permite uma mais rápida inicialização do mecanismo de histerese na geração de eventos.
- **alarmRisingThreshold** e **alarmFallingThreshold** - valor inicial: 0
  Pela sua depêndencia da variável a monitorizar não é possível estimar um valor para a sua inicialização.
- **alarmRisingEventIndex** e **alarmFallingEventIndex** - valor inicial: 0
  Este valor implica a ausência de eventos associados.
- **alarmOwner** - valor inicial: string nula
  Deve ser preenchido pelo gestor.

E. Partilha de recursos disponíveis

Com a utilização de um objecto *owner*, existente em todos os grupos da RMON MIB, poder-se-ão evitar conflitos na partilha de recursos. O preenchimento deste objecto permite a outras estações gestoras reconhecer facilmente os recursos que estão na sua posse, bem como obter a informação necessária para contactar outras estações gestoras que possuam entradas activas, podendo assim negociar com elas a libertação de recursos que estejam nas sua posse e que sejam necessários para outras actividades.

VII. RESULTADOS

A implementação destes dois grupos permite-nos obter informação de variáveis implementadas no agente em questão. Desta forma, podemos verificar se estas ultrapassam valores não desejados, possibilitando-nos assim, estar a par da situação actual da rede.

O grupo *alarm* está implementado para suportar a monitorização de 50 variáveis ao mesmo tempo.

Nesta implementação o número de entradas na tabela de *logs* relativas ao mesmo evento é controlado de modo a que apenas as últimas 50 entradas sejam visualizadas, evitando assim uma sobrecarga da memória do sistema.

Contudo, na utilização máxima do sistema, ou seja com 50 entradas na tabela de alarmes activas, em que cada uma delas poderá ter a si associada 2 entradas na tabela de eventos e 100 entradas na tabela de logs activas, poderemos ter 100 entradas de eventos e 5000 entradas de logs activas, respeitantes ao grupo *alarm*, o que é perfeitamente possível passadas algumas horas de funcionamento do sistema. Nesta situação o sistema ficaria muito sobrecarregado, surgindo atrasos tanto na resposta a pedidos do gestor como na execução das funções de monitorização, pondo em questão a fiabilidade do sistema.

VIII. CONCLUSÕES

Estudamos e apresentamos algumas conclusões sobre a viabilidade de construção de aplicações de gestão, particularmente de monitorização de redes, com base em interfaces de baixo nível. Tanto a interface fornecida pelo *Packet Filter* como a fornecida pela DLI mostraram algumas limitações, em particular esta última pela impossibilidade de captar pacotes em modo promíscuo.

Apresentamos igualmente um estudo e uma implementação da *Remote Monitoring MIB*, uma interface standard, para monitores de redes, normalizada pela comunidade Internet para um modelo global de gestão baseado no protocolo SNMP.

AGRADECIMENTOS

Agradecemos ao Eng. Fernando Pipa pelo apoio na realização deste trabalho.

REFERÊNCIAS

[1] Digital Equipment Corporation, "The Packet Filter: An Efficient Mechanism for User-level Network Code", *Ultrix V4.1 Manual*.

[2] Sun Microsystems Inc. *NIT(4P); SunOS 4.1.1 Reference Manual*. Mountain View, CA, October 1990.

[3] S. McCanne and Van Jacobson, "The BSD Packet Filter: A New Architecture for User-level Packet Capture", *WInter USENIX Conference*, San Diego, CA, January 1993.

[4] Digital Equipment Corporation, "Guide to the Data Link Interface",*Ultrix V4.1 Manual*.

[5] J. Case, M. Fedor, M. Schffstall, J. Davin, "A Simple Network Management Protocol (SNMP)", *RFC 1157*, IAB, May 1990.

[6] K. McCloghrie and M. Rose (eds), "Management Information Base for Network Management of TCP/IP-based Internets: MIB-II", *RFC 1213*, Hughes LAN System, Performance Systems Internation, March 1991.

[7] M. Rose and K. McCloghrie, "Structure and Identification of Management Information for TCP/IP-based internets", *RFC 1155*, Performance Systems Internation, Hughes LAN System, March 1991

[8] A. P. Carvalho and N. P. da Rocha, "Plataforma SNMP para o Desenvolvimento de Objectos de Gestão", *Revista do DETUA, 1993.*

[9] S. Waldbusser, "Remote Network Monitoring Management Information Base", *RFC 1271*, Carnegie Mellon University, November 1991.

[10] S. Waldbusser, "Token Ring Extentions to the Remote Network Monitoring MIB", *RFC 1513*, Carnegie Mellon University, Setember 1993.

[11] M. Rose and K. McCloghrie (eds), "Concise MIB definitions", *RFC 1212*, Performance Systems Internation, Hughes LAN System, March 1991.