UNIVERSIDADE FEDERAL DO CEARÁ
CENTRO DE CIÊNCIAS
DEPARTAMENTO DE COMPUTAÇÃO
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

Dissertação de Mestrado

Mecanismo de Coleta de Dados no *Framework* Ativo ANTS para Subsídio em Gerenciamento de Redes

Victória Damasceno Matos

Profº José Neuman de Souza, Doutor
Orientador

Jorge Luiz de Castro e Silva, Doutorando
Co – Orientador

Fortaleza, Julho de 2003

UNIVERSIDADE FEDERAL DO CEARÁ
CENTRO DE CIÊNCIAS
DEPARTAMENTO DE COMPUTAÇÃO
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

Mecanismo de Coleta de Dados no *Framework* Ativo ANTS para Subsídio em Gerenciamento de Redes

Este exemplar corresponde à Dissertação de Victória Damasceno Matos

Profº José Neuman de Souza, Doutor
(Orientador)

Jorge Luiz de Castro e Silva, Doutorando
(Co-orientador)

Dissertação apresentada ao Mestrado de Ciência da Computação da Universidade Federal do Ceará (UFC), como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Aos meus pais, Francisco e Maria,
e aos meus irmãos, Victor e Verbena.

Agradecimentos

Ao longo dos meses de dedicação a este trabalho, algumas pessoas tiveram essencial participação, com seus incentivos e orientação. Desta forma, quero expressar de maneira particular a minha gratidão:

- À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), pelo incentivo financeiro;

- A todos os professores e funcionários do Mestrado em Ciências da Computação da Universidade Federal do Ceará, em especial, à professora Rossana Andrade, que sempre me incentivou e motivou;

- Aos professores José Neuman de Souza e Jorge Luiz de Castro e Silva pela competência com que desempenharam o papel de orientação;

- A todos os amigos que compartilharam comigo os momentos de tristeza e de alegria no decorrer dos últimos dois anos;

- Ao meu namorado, Francisco Alceles, por sua paciência, ajuda e fundamental apoio, se posicionando sempre ao meu lado, independentemente da dificuldade da situação;

- Aos meus queridos pais, Francisco Matos e Maria Damasceno, e aos meus irmãos, Victor e Verbena, que foram fundamentais em toda minha vida, incentivando e aconselhando da melhor forma possível;

- A Deus, principalmente, que permitiu, direcionou, protegeu e me deu forças em cada novo dia de minha vida.

Sumário

Lista de Abreviaturas

ANEP	(*Active Network Protocol*)
ANTS	(*Active Node Transfer System*)
AMP	(*Active Measurement Project*)
APCI	(*Active Processing Control Information*)
ATM	(*Asynchronous Transfer Mode*)
CANEs	(*Composable Active Network Elements*)
CMIP	(*Common Management Information Protocol*)
CMOT	(*CMIP over TCP/IP*)
DARPA	(*Defense Advanced Research Projects Agency*)
GPS	(*Global Positioning System*)
IETF	(*Internet Engineering Task Force*)
IP	(*Internet Protocol*)
ISO	(*International Organization for Standardization*)
Janos	(*Java-oriented Active Network Operating System*)
LAN	(*Local Area Network*)
MIB	(*Management Information Base*)
MINC	(*Multicasting-based Inference of Network-internal Characteristics*)
MIT	(*Massachusetts Institute of Technology*)
NIC	(*Network Interface Card*)
NIMI	(*National Internet Measurement Infrastructure*)
NLANR	(*National Laboratory for Applied Network Research*)
NMS	(*Network Management System*)
OID	(*Object Identifier*)
OSI	(*Open Systems Interconnection*)

PDU	(*Protocol Data Unit*)
PLAN	(*Programming Language for Active Networks*)
RFC	(*Request For Comments*)
RPC	(*Remote Procedure Call*)
SNMP	(*Simple Network Management Protocol*)
TCP/IP	(*Transmission Control Protocol / Internet Protocol*)
TTL	(*Time To Life*)
UDP	(*User Datagram Protocol*)
vBNS	(*Very High-speed Backbone Network Service*)
WAN	(*Wide Area Network*)

Lista de Figuras

Lista de Tabelas

Resumo

MATOS, Victória. **Mecanismo de Coleta de Dados no *Framework* Ativo ANTS para Subsídio em Gerenciamento de Redes**. Orientador: José Neuman de Souza e Co-orientador: Jorge Luiz de Castro e Silva. Fortaleza-Ce. UFC, 2003. Diss.

As redes ativas se apresentam como uma tecnologia inovadora para solucionar diversos problemas existentes nas redes tradicionais. Vários estudos aplicam os conceitos desse novo paradigma com o objetivo de melhorar a eficiência no gerenciamento de redes. Recentes pesquisas comprovam que essa tecnologia provê a flexibilidade e distribuição necessárias à gerência.

Este trabalho implementa um mecanismo de coleta de dados em um *framework* ativo desenvolvido com base no *kit* de ferramentas ANTS (*Active Node Transfer System*), que é uma pesquisa bastante relevante no ramo de tecnologia ativa, desenvolvido pelo *Massachusetts Institute of Technology* (MIT). A proposta implementada é uma alternativa para operações de coleta de informações, eliminando algumas deficiências dos mecanismos tradicionais, como grande consumo de largura de banda por gerar tráfego excessivo e sobrecarga do gerente.

Um protótipo foi desenvolvido com o objetivo de validar a viabilidade deste mecanismo de coleta. Após a sua implementação, foi realizada uma série de análises de desempenho e comparações com coletas nos modelos tradicionais. Estes dados permitiram o levantamento das vantagens e desvantagens da arquitetura proposta nesta dissertação.

Palavras-chave: gerenciamento de redes, coleta de dados, redes ativas, SNMP, ANTS.

Abstract

MATOS, Victória. **Mechanism of Data Collection in Active Framework ANTS for Subsidy in Networks Management**. Advisor: Jose Neuman de Souza and Co-advisor: Jorge Luiz de Castro e Silva. Fortaleza-Ce. UFC, 2003. Diss.

The active networks are presented as an innovative technology to solve diverse problems existing on the traditional networks. Several studies apply the concepts of this new paradigm with the objective to improve the efficiency in the network management. Recent researches prove that this technology provides flexibility and distribution necessary to the management.

This work implements a data collection mechanism in the active framework developed on the ANTS (Active Node Transfer System) toolkit, of the Massachusetts Institute of Technology (MIT), which it is a prominent research in the branch of active technology. The proposal implemented is an alternative for information collection operations, eliminating some deficiencies of the traditional mechanisms, as great consumption of bandwidth, by generating excess traffic and overloading the manager.

A prototype was developed as objective to validate the viability of this collection mechanism. After its implementation, a series of performance analyses and comparisons with collections in the traditional models was made. These data allow to the survey of the advantages and disadvantages of the architecture proposal in this dissertation.

Keywords: network management, data collection, active networks, SNMP, ANTS.

1 Introdução

1.1 Motivação

É notório que a Internet está crescendo de maneira exorbitante. Devido a isso, o monitoramento da rede torna-se essencial para se obter um entendimento claro de seu desempenho. Existem diversos estudos em medição e análise de desempenho de rede, nos quais percebe-se que as comunidades de pesquisa estão de comum acordo que uma infra-estrutura de medição de tráfego é indispensável para a otimização das operações em rede e para o crescimento futuro da Internet [1, 2].

Na gerência de redes, levando-se em consideração as suas áreas funcionais, o gerenciamento de desempenho é o maior motivador para os estudos de coleta eficiente de dados. O gerenciamento de desempenho está diretamente relacionado com a disponibilidade de informações de tráfego, de modo a poder determinar a carga da rede. Diferentemente do gerenciamento de falhas, ele é pró-ativo, envolvendo a coleta e interpretação de medições periódicas dos indicadores de desempenho, identificando gargalos, avaliando tendências e fazendo previsões do comportamento futuro da rede. Este tipo de gerenciamento preocupa-se com o desempenho atual da rede, incluindo parâmetros estatísticos, tais como: atraso, vazão, disponibilidade e número de retransmissões. Para isso, é necessário o acesso dinâmico a uma certa quantidade de dados da rede, que são obtidos através de algum método de coleta de dados.

Uma outra consideração importante para este trabalho é que o gerenciamento de redes é uma das aplicações que pode se beneficiar com o uso de tecnologia ativa.

Estudos em redes ativas [3, 4, 5, 6, 7], mostram que uma das maiores plataformas com esta tecnologia é o ANTS (*Active Node Transfer System*), sendo por isso, o *framework*

adotado para o mecanismo de coleta de dados proposto nesta dissertação. Desta forma, é possível que uma ferramenta de gerenciamento obtenha dados precisos e de maneira eficiente utilizando este mecanismo ativo.

A maior motivação do modelo de coleta ativa proposto é devido ao fato das abordagens tradicionais de gerenciamento, como o SNMP (*Simple Network Management Protocol*) e o CMIP (*Common Management Information Protocol*), gerarem tráfego excessivo, pois se tratam de mecanismos baseados no modelo cliente-servidor, no qual o gerente centraliza as informações e fornece a ordem para execução de ações corretivas, e o agente interage com a MIB (*Management Information Base*) e executa as ordens do gerente. Assim, os sistemas de gerenciamento mais complexos necessitam de uma volumosa troca de informações entre o gerente e o agente. Além disso, tais mecanismos normalmente apresentam retardos elevados para a detecção de uma situação anômala ou para o desencadeamento de ações corretivas. Estas ações, por sua vez, costumam ser executadas somente após a ocorrência de uma falha, ou seja, são reativas. Sugere-se que o gerenciamento seja descentralizado, visando à redução do tráfego e dos retardos, e, preferencialmente, pró-ativo.

1.2 Objetivos e Contribuições do Trabalho

Diante da motivação exposta, teve-se como objetivo desenvolver um mecanismo de coleta de informações que se mostrasse como uma melhor alternativa de coleta de dados perante os métodos tradicionais, que possuem deficiências, que serão descritas no decorrer desta dissertação, no seu modelo cliente-servidor.

Dessa forma, foi desenvolvida uma aplicação no ambiente de redes ativas ANTS, que possibilitou uma coleta descentralizada e sem trocas excessivas de mensagens entre o nó gerente e os nós monitorados. Assim, baseado no modelo de coleta ativa proposto, foram implementadas rotinas na plataforma ativa ANTS, onde, a partir de um nó ativo, se desencadeia um processo de inicialização da captura de dados, ou seja, um nó ativo envia cápsulas (correspondentes aos pacotes, nas redes tradicionais) que executam a tarefa de coleta localmente. Para isso, essas cápsulas possuem código Java, que é responsável pela coleta local nos nós ativos que estão sendo monitorados. Portanto, os nós em questão enviam apenas os resultados dos parâmetros de tráfego desejados para o gerenciamento.

Com esse modelo, que fornece dinamicamente parâmetros de tráfego da rede, pretende-se subsidiar métodos de gerenciamento, principalmente aqueles que usam tecnologia ativa. Salienta-se que, em uma gerência mais complexa, esta captura de informações apresenta melhores resultados, como será discutido nos capítulos posteriores. Estado da Arte

Algumas abordagens de medição e análise de desempenho da rede classificam a coleta de dados em intrusiva ou não-intrusiva [2, 8]. A medição não-intrusiva mede o comportamento observando as taxas de pacotes que chegam no sistema fim. Neste caso, nenhum tráfego é inserido na rede. Em contra-partida, os sistemas de medição intrusivos injetam pacotes nas redes com o intuito de que eles lhes tragam algum tipo de informação.

Pode-se citar como medição não-intrusiva: as medições utilizando o SNMP, o *tcpdump*, o *NetFlow* e o *CoralReef*. O SNMP é o protocolo de gerência de rede mais utilizado na Internet hoje [9]. Através de um modelo cliente-servidor, o SNMP faz uso das informações contidas na MIB de cada cliente. Outra ferramenta de monitoramento é o *tcpdump*, que coleta os pacotes transmitidos e recebidos pelos *hosts* com o sistema operacional UNIX [10]. O *NetFlow* é o sistema de monitoramento disponível nos roteadores Cisco e coleta as estatísticas de tráfego observadas em um ponto na rede [11]. O *CoralReef*, desenvolvido pela CAIDA, é um pacote de *software* destinado ao monitoramento de enlaces ATM (*Asynchronous Transfer Mode*) [12], que coleta pacotes *timestamped* (que é uma marcação de tempo bastante precisa feita pelo relógio interno das placas de captura, especificando o momento em que o pacote foi coletado).

Os sistemas de medições intrusivas incluem o NIMI, o MINC, o *Surveyor* e o AMP. O NIMI (*National Internet Measurement Infrastructure*) usa ferramentas como *ping*, *traceroutes*, *mtrace* e *treno* para desenvolver a medição [13]. O MINC (*Multicasting-based Inference of Network-internal Characteristics*) transmite mensagens de sondas para muitos destinos e conclui taxa de perda de *links* e *delay* [14]. A infra-estrutura do *Surveyor* emprega máquinas de medição (50, aproximadamente) com *links* de grande e pequena distância. O GPS (*Global Positioning System*) é usado para sincronizar todas essas máquinas e, dessa forma, possibilitar a medição dos *delays* nos caminhos da Internet [15]. O AMP (*Active Measurement Project*) do NLANR (*National Laboratory for Applied Network Research*) consiste de um conjunto de estações monitoramento que medem a performance do *backbone* vBNS[1] (*Very High-speed Backbone Network Service*) [16].

[1]vBNS é uma rede (sustentada pelo NSF e MCI) que oferece recursos de transmissão de altíssima velocidade (até vários Gbps) com gerenciamento de QoS.

Abordagens mais atuais vieram com os estudos de tecnologia ativa. A partir de 1995, após as primeiras publicações oriundas de pesquisas financiadas pelo DARPA (*Defense Advanced Research Projects Agency*), a distribuição do controle e da programabilidade dos nós de comutação de uma rede passaram a ser investigadas [3]. Atualmente existem dois paradigmas na área de tecnologia ativa: redes ativas e agentes móveis. Ambos oferecem suporte a modelos que utilizam recursos computacionais no interior e/ou nas bordas da rede para o carregamento e a execução de programas "sob demanda". Embora conceitos dessas tecnologias tenham sido originados em diferentes comunidades de pesquisas, visando resolver diferentes problemas, eles começam a se sobrepor em termos de foco e aplicabilidade [17].

Pelos estudos bibliográficos, notou-se que nos últimos anos a tecnologia ativa tem se apresentado, em algumas pesquisas, como recursos para o melhoramento da gerência de redes, com o foco centralizado na captura de indicadores de desempenho [5, 6, 18, 19, 20], substituindo métodos de medição tradicionais já citados. Porém, esta frente de pesquisa está apenas começando. Verificou-se que existem poucos estudos na área de aprimoramento da gerência com o uso desta tecnologia. Portanto, um dos objetivos deste trabalho foi que o modelo proposto se baseasse nos princípios de redes ativas, amenizando o tráfego injetado na rede, uma vez que a coleta de dados é realizada localmente, e também, por outro lado, como se trata de um mecanismo intrusivo, não apresenta o *delay* existente na coleta não-intrusiva.

1.3 Organização da Dissertação

Conforme a organização da dissertação, o próximo capítulo conceituará as redes ativas. Neste capítulo serão descritas implementações e aplicações que se beneficiam com a tecnologia ativa, pois se trata de uma inovação na área de redes de computadores por trazer características de descentralização e de grande adequação a novos protocolos. Também serão mostradas algumas arquiteturas existentes, seguido de um levantamento dos principais projetos nessa área.

O detalhamento da arquitetura da plataforma ativa ANTS, que foi o *framework* utilizado no mecanismo de coleta proposto, se encontrará no capítulo 3. Neste capítulo, além da arquitetura, serão levantados aspectos de segurança e do modelo de desenvolvimento do ANTS.

No capítulo 4, encontra-se o ponto de maior ênfase da dissertação. Nele serão expostas as idéias fundamentais do mecanismo de coleta de dados em um ambiente de redes ativas,

juntamente com o detalhamento de sua arquitetura e os princípios do modelo de implementação para a validação dos objetivos desta dissertação.

Os *benchmarks* que foram feitos na evolução do trabalho serão apresentados no capítulo 5. Uma análise comparativa entre o mecanismo de coleta ativo e a coleta com o protocolo SNMP devidamente comentada também fará parte do conteúdo deste capítulo, onde serão provados os reais ganhos com a coleta em questão. No capítulo 6 conterá as conclusões e os trabalhos futuros. E, por fim, o apêndice A traz a descrição das medidas estatísticas utilizadas, e o B mostra o código fonte da implementação do mecanismo proposto.

2 Redes Ativas

2.1 Introdução

As redes tradicionais garantem o compartilhamento de dados de forma que seus pacotes possam ser eficientemente transportados entre os sistemas conectados. Elas fazem somente o processamento necessário para enviar os pacotes ao seu destino. Esse tipo de rede é insensível aos *bits* que carrega e os dados dos usuários são transmitidos sem sofrer modificação. O papel da computação nesse tipo de rede é extremamente limitado [21].

Com o tempo, a tecnologia foi ficando mais barata, e mais funcionalidades foram sendo implementadas nas redes, em um esforço para oferecer melhores serviços para os usuários. Para garantir a criação e implementação desses novos serviços surgiu o conceito de *redes ativas*.

As redes ativas são uma nova representação de arquitetura de rede, que permitem a adição, pelos usuários, de programas customizados nos nós da rede, possibilitando que estes dispositivos realizem atividades de computação nas mensagens que fluem por eles [3, 21]. Este processamento pode ser individualizado por usuários ou por aplicação.

A Figura 1 mostra características das redes ativas de maneira comparativa às redes tradicionais. Nas redes tradicionais, um *host* tem a única função de retransmitir um pacote, sem alteração alguma em seu conteúdo. Em contrapartida, na arquitetura de redes ativas, verifica-se a existência de rotinas nos nós ativos e, nos pacotes ativos, códigos de programas. Desta forma, os nós e os pacotes atuam nas mensagens que fluem na rede, podendo modificar o comportamento de um nó ou pacote.

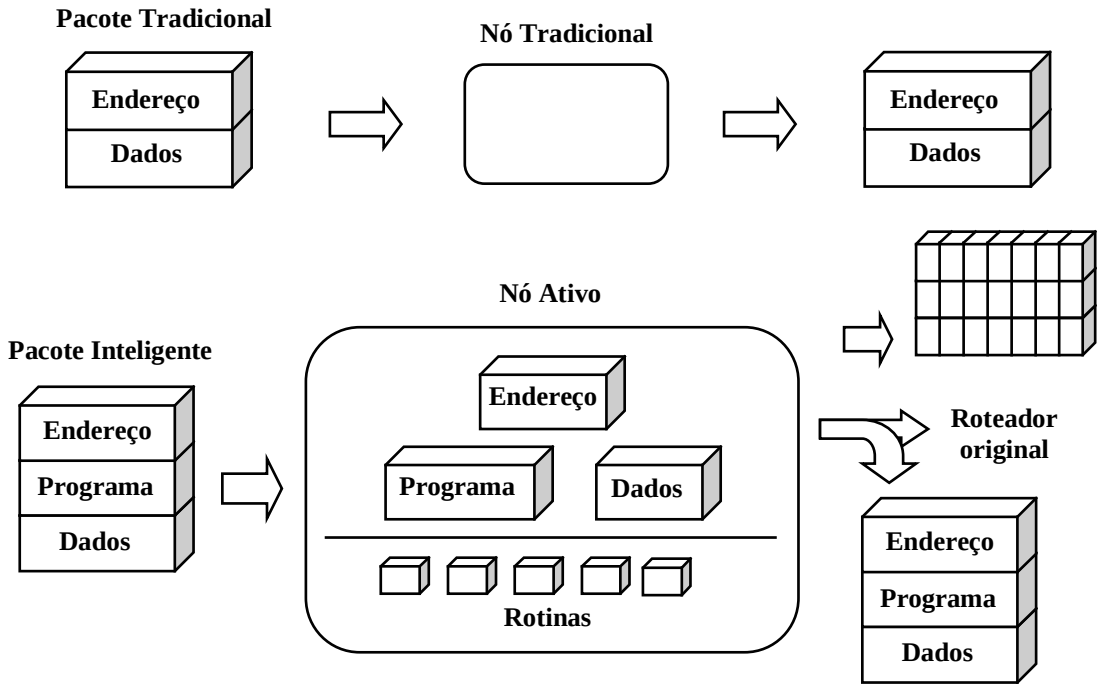

Figura 1. Comparação entre redes ativas e redes tradicionais.

Esta tecnologia se propõe a solucionar diversos problemas que existem nas redes passivas, tais como: dificuldade de integrar novas tecnologias e/ou padrões em uma infra-estrutura compartilhada de rede; baixo desempenho, devido às operações redundantes em várias camadas de protocolo; complexidade em acomodar novos serviços no modelo de arquitetura existente.

2.2 Aplicações da Tecnologia

Existem aplicações em que seria interessante, do ponto de vista do usuário, a capacidade de programar suas redes. Assim, elas aproveitariam as características básicas das redes ativas, permitindo a criação de novos protocolos e tecnologias inovadoras, que podem ser facilmente implementadas.

Entre as implementações e aplicações de redes que se beneficiam com a arquitetura de redes ativas encontram-se [21, 22]:

2.2.1 Automatização da atualização das regras de um *Firewall*

Os *firewalls* implementam filtros que determinam quais pacotes devem ser passados de forma transparente e quais devem ser bloqueados. Embora tenham uma relação de paridade com outros roteadores, *firewalls* implementam funções específicas de aplicações e de usuários, além do roteamento dos pacotes. A necessidade de atualizar o *firewall*, para habilitar o uso de novas aplicações, muitas vezes é um impedimento para a sua atualização. Em uma rede ativa, este processo poderia ser automatizado através da habilitação do acesso de aplicações de representantes autorizados, que se autenticariam no *firewall* e injetariam módulos apropriados.

2.2.2 Esquemas de *Web Proxies* e tecnologia inteligente de *Caching*

Os *web proxies* são um exemplo específico de implementação, sendo desenvolvido para servir e armazenar temporariamente páginas *web*. Uma fração significativa das páginas *web* é programada dinamicamente e não está sujeita ao armazenamento passivo. Isto sugere que o desenvolvimento de esquemas de *web proxies* suporte *caches* ativos que armazenem e executem programas que gerem páginas *web*.

Um outro argumento em favor da utilização de tecnologias ativas é a tecnologia inteligente de *caching*, pois se sabe que o *caching* é uma tecnologia utilizada para evitar grandes tráfegos de informação da rede. O ponto principal dessa tecnologia é saber “de que” e “onde” deve ser o *caching*.

Numa rede ativa, os nós sabem se devem ou não fazer *cache* de uma determinada informação, baseando-se nos pedidos dos clientes, bem como no *cache* dos nós vizinhos.

Um modelo proposto por Harvest [23] emprega um esquema hierárquico no qual os nós de *cache* são posicionados perto das extremidades da rede, isto é, dentro da organização do usuário. Este sistema é escalonável e pode ser estendido, permitindo que nós da hierarquia estejam posicionados em pontos estratégicos dentro da rede de provedores de acesso. Um problema interessante é o desenvolvimento de algoritmos e ferramentas que automaticamente realizem o balanceamento da hierarquia através do reposicionamento dos próprios nós de *cache*, e não apenas da informação do *cache*.

2.2.3 Melhoramento do tráfego *Multicasting*

As redes ativas podem ser usadas para melhorar o tráfego *multicasting*, uma vez que nós ativos internos podem evitar problemas, como o alto número de retransmissões inúteis, duplicações de pacotes, e imunidade de grupos às mudanças de membros. Os atuais esquemas passivos oferecem soluções parciais para esses problemas.

2.2.4 Controle de congestionamento

A comunidade científica acredita que problemas de congestionamento na rede não desaparecerão em um futuro próximo. Desta forma, é essencial encontrar uma maneira para que se possa conviver com eles. Congestionamento é um caso especial de gerenciamento de rede. Sendo assim, é também beneficiado com a utilização de redes ativas.

Entre algumas soluções para amenizar o congestionamento em redes ativas podem ser citadas:

- A utilização de *buffers* nos nós ativos, para monitorar a largura de banda e, assim, controlar o fluxo de dados;
- Se forem vários fluxos, há diferentes parâmetros de congestionamento. O nó ativo deve controlar as taxas de cada fluxo, além da taxa total. Ele também deve se adaptar às mudanças dinâmicas de parâmetros;
- Diretamente no local de congestionamento, e somente se for necessário, pode ser feita uma transformação de dados. Apesar de ser um recurso poderoso, pode ter um custo computacional elevado;
- Em caso de congestionamento, pode ser feito o descarte seletivo de unidades menos importantes (em relação à informação que carregam);
- Finalmente, pode haver uma interação *multistream*. Por exemplo: se na recepção de vídeo e áudio estiver havendo perda de vídeo, as unidades de áudio devem ter prioridade extra para garantir a recepção de alguma informação.

2.2.5 Gerenciamento de rede

São de maior interesse para este trabalho os benefícios trazidos pelas redes ativas em aplicações de gerenciamento de rede. Um vez que os nós internos da rede são ativos, o gerenciamento da rede deixa de ser central e passa a ser distribuído, diminuindo tanto o tempo

de resposta quanto a utilização da largura de banda. É possível também inserir códigos especiais nos pacotes, para que eles possam agir como “primeiros socorros” no caso de encontrarem um nó problemático. Dessa forma, o nó afetado mudaria seu estado automaticamente, sem esperar uma resposta do gerenciamento central. As políticas de gerenciamento podem ser mudadas facilmente graças à flexibilidade inerente à tecnologia de redes ativas.

2.3 Arquitetura

As arquiteturas são agrupadas de acordo com seus métodos básicos para a realização da rede ativa [22], são eles: método de pacotes ativos, método de nós ativos e método de pacotes e nós ativos.

A maioria das primeiras redes ativas utilizava o método de pacotes ativos. Nesse método, os nós também são ativos, porque permitem operações computacionais dentro da camada de aplicação, mas não possuem código ativo. Portanto, a razão de se chamar “tecnologia de pacotes ativos” é que o código ativo encontra-se nos pacotes, e é executado nos dados desse mesmo pacote, ou pode ser executado na ordem de mudança do estado ou do comportamento de um nó.

No método de nós ativos, os pacotes não possuem o código a ser executado, mas transmitem algumas identificações ou referências para as funções pré-definidas que se encontram nos nós ativos. Os nós são ativos no sentido que eles decidem quais funções são executadas em seus dados e fornecem os parâmetros para essas funções. Portanto, o código existente encontra-se nos nós ativos. A motivação para tal arquitetura é que o método de pacotes ativos sofre problemas relativos tanto à performance (por causa dos requerimentos de segurança), quanto à capacidade (pois a única maneira de minimizar a questão de segurança é pela restrição dos programas transmitidos nos pacotes).

Os pacotes ativos podem transmitir eficientemente código somente quando este é relativamente simples e restrito. Por outro lado, os nós ativos podem transportar eficientemente qualquer código, porém, esse código é pré-definido, pois tem que estar nos nós ativos ou, no mínimo, em um nó de onde possa ser baixado para outro nó. No método de pacotes e nós ativos, os pacotes ativos transmitem o código simples existente; outros códigos mais complexos encontram-se nos nós ativos. Usualmente, essa arquitetura permite ao usuário escolher entre os dois métodos, de acordo com a natureza de suas aplicações.

2.4 Levantamento de Estudos

As redes ativas contam com a habilidade de adicionar programas facilmente à sua infra-estrutura. A dificuldade de se escolher trabalhos atualmente em curso sobre este tema se dá devido à estrutura dinâmica que é inerente às próprias redes ativas. Como existe um número grande de projetos, serão comentados alguns, ressaltando suas arquiteturas e aplicações, como gerenciamento de rede e congestionamento. O próximo capítulo discorrerá com maior ênfase o projeto ANTS desenvolvido pelo MIT.

2.4.1 *Smart Packets*

O projeto *Smart Packets* (pacotes inteligentes), atualmente ainda em andamento [24], desenvolvido pela a *BBN Technologies* [5], foi projetado para demonstrar os benefícios de redes ativas em gerenciamento de redes. Sua arquitetura baseia-se no método de pacotes ativos, em que os pacotes tradicionais são substituídos por *Smart Packets*, que podem transmitir programas.

Existem quatro tipos de pacotes na estrutura do *Smart Packets*: pacotes de programas, de dados, de erros e de mensagens. Os pacotes de programas transmitem o código para ser executado no nó apropriado. Os pacotes de dados transmitem os resultados da execução de volta para o programa original de gerenciamento de rede. Os pacotes de mensagens transmitem as informações e os pacotes de erros retornam as condições de erro.

Os *Smart Packets* são encapsulados em pacotes ANEP (*Active Network Protocol*) (que é um protocolo especial desenvolvido pelo DARPA a fim de facilitar portabilidade e interoperabilidade entre os diferentes projetos de redes ativas), e que, por sua vez, são encapsulados em pacotes IP (*Internet Protocol*). Os *Smart Packets* são enviados para um *host* final ou para cada roteador. No primeiro caso, o conteúdo do *Smart Packet* é executado no *host* final e os resultados são enviados de volta, no segundo caso, o conteúdo é executado em todos os nós intermediários.

2.4.2 *Active Processing Control Information* (APCI)

O *Georgia Institute of Technology* subsidia o projeto CANEs (*Composable Active Network Elements*) que investiga arquiteturas e aplicações para redes ativas. Um dos resultados deste projeto foi o APCI [25] que é um exemplo de uma tecnologia experimental em curso que enfoca os benefícios de redes ativas com relação a congestionamento em redes.

O método usado é a definição pela rede de um conjunto finito de funções, as quais podem ser computadas em um nó ativo por um processador ativo. Existe também um cabeçalho de informações em cada pacote que especifica quais execuções podem ser feitas nele, chamadas de APCI (*Active Processing Control Information*).

Quando um pacote chega a um nó, os seguintes passos são realizados:

- O destino de um pacote é processado;
- Se houver um APCI, o pacote é então enviado para um processador ativo para processamento futuro. Se não, o pacote é transmitido;
- A função especificada no APCI é realizada;
- O cabeçalho do pacote e o APCI são recomputados se o resultado da função são os dados transformados. O estado dos nós também é atualizado como requerido pela função;
- O pacote é transmitido.

Os estudos sobre o APCI já foram finalizados, porém ainda existem outras pesquisas na área de redes ativas com o projeto CANEs [26].

2.4.3 *Active IP Option*

O *Active IP Option* [6] foi proposto pelo MIT e baseia-se na arquitetura de pacotes ativos, pois descreve uma extensão do mecanismo de opções de IP que suporta o encapsulamento de fragmentos de programas em datagramas (unidade de transmissão do protocolo IP) e a avaliação desses fragmentos à medida que eles atravessam a rede. Hoje, os pacotes passivos são substituídos pelas cápsulas ativas, que são programas em miniatura executados à medida que trafegam. Estas cápsulas podem chamar primitivas pré-definidas que interagem com o ambiente de nós locais, e deixam para trás informações em um nó que elas tenham visitado. As cápsulas seguintes podem transmitir código que depende dessas informações.

O *Active IP Option* pode fazer roteamento, cópias e incorporação de funções. O ambiente de processamento permite que condições da rede sejam examinadas, o datagrama atual seja remetido e datagramas adicionais sejam construídos e enviados. O estado de um nó também pode ser modificado.

Este projeto, após ter sido concluído, serviu como base para uma das grandes pesquisas na área de redes ativas. Com os resultados obtidos no projeto *Active IP Option*, o MIT passou a implementar o ANTS, que é a plataforma estudada nesta dissertação.

2.4.4 *SwitchWare*

O projeto *SwitchWare* desenvolvido pela *University of Pennsylvania* [7], tem uma arquitetura onde são definidas três camadas: pacotes ativos, *switchlets* e infra-estrutura de roteadores ativos. A primeira camada realiza o que se chama de método de pacotes ativos e a segunda, o método de nós ativos.

No *SwitchWare*, os pacotes ativos transmitem programas contendo código e dados e repassam ambos para o cabeçalho e campo de mensagem de um pacote convencional. A linguagem de programação utilizada é o PLAN (*Programming Linguage for Active Netwoks*), que é uma linguagem de alto nível, também desenvolvida pela *University of Pennsylvania*. Os programas PLAN chamam rotinas, chamadas *switchlets*, as quais podem autenticar ou usar outros mecanismos mais pesados proporcionar a segurança básica necessária.

A infra-estrutura de roteadores ativos é a base sólida sobre a qual os pacotes ativos e as *switchlets* estão construídos. A segurança da arquitetura *SwitchWare* é garantida nessa camada.

Os trabalhos do projeto *SwitchWare* referentes a redes ativas desencadearam várias frentes de pesquisa nesta área. Este projeto de pesquisa está em plena atividade e se encontra disponível em [27].

2.4.5 *NetScript*

Atualmente, sistemas de rede são difíceis de projetar, implementar, implantar e gerenciar. Nós intermediários (roteadores, *switches*) são construídos como caixas-pretas; por outro lado, nós fim (estações, servidores) podem executar diversas aplicações. Para programar novas funções nos nós intermediários deve-se fazer um complexo processo de padronização que pode durar uma década. Além disso, enquanto os nós fim podem ser facilmente programados e utilizados para aplicações multimídia, os nós intermediários devem esperar que apareçam protocolos padrão que suportem roteamento e garantam qualidade de serviços.

O projeto *NetScript* foi proposto pela *Columbia University* [21]. Ele é uma linguagem e ambiente para implementar protocolos ou serviços.

Uma rede *NetScript* consiste em uma coleção de nós (por exemplo: PCs, *switches*, roteadores) que executam um ou mais mecanismos *NetScript*. Um mecanismo *NetScript* é uma abstração de *software* para dispositivos programáveis de processamento de pacotes.

O sistema *NetScript* consiste de dois componentes: *NetScript* (uma linguagem de fluxo de dados para construir protocolos de processamento de pacotes); e um *kit* de ferramentas *NetScript* (um conjunto de classes Java compilado pela linguagem).

O *NetScript* proporciona: meios para a construção de elementos ativos de operações primitivas em pacotes em fluxo; alocação de recursos dos nós para suportar operações computacionais e comunicação pelos elementos ativos; e controle do fluxo de pacotes que trafega entre elementos ativos.

As pesquisas deste projeto já foram concluídas, com sua ultima publicação em 1999, porém seus resultados e implementação, que se encontram disponível em [28], serviram como base para vários estudos e desenvolvimento de soluções em redes ativas.

2.4.6 *Active Node Transfer System* (ANTS)

O ANTS (Sistema de Transferência de Nós Ativos), foi considerado neste estudo como uma das ferramentas de maior destaque na área de tecnologias ativas. Além disto, o acesso à documentação e ao pacote que implementa este ambiente é bastante facilitado. Devido a estas razões, adotou-se esta plataforma para desenvolvimento do modelo proposto. Assim, esta arquitetura de redes ativas será abordada com maior ênfase no próximo capítulo.

3 *Active Node Transfer System* (ANTS)

3.1 Introdução

O ANTS é um *kit* de ferramentas para redes ativas baseado em Java desenvolvido pelo MIT para construir e implementar dinamicamente protocolos de rede. Essa metodologia baseia-se em código móvel e técnicas de *cache*. A arquitetura do sistema permite que novos protocolos sejam implementados tanto nos roteadores quanto nos sistemas fim, sem a necessidade de coordenação e sem interação indesejada entre os protocolos coexistentes [4].

O ANTS pode ser visto como um sistema de programação distribuída para a rede. Ele também provê um modelo de programação para expressar novos protocolos como sendo operações nos nós. A ferramenta garante alta flexibilidade na inserção destes protocolos e aplicações em redes de computadores.

Após o seu desenvolvimento pelo MIT (versão 1.0), outras universidades atuaram em seus estudos. A versão 2.0.2 foi desenvolvida pela *University of Washington* e a versão 2.0.3 pela *University of Utah*, sendo esta última a responsável pela continuidade do projeto.

3.2 Arquitetura do Protocolo ANTS

Uma rede baseada em ANTS consiste em grupos de nós interconectados que executam o ANTS em tempo real. Os nós podem estar conectados em uma LAN (*Local Area Network*) ou WAN (*Metropolitan Area Network*). Os serviços garantidos por ANTS são flexíveis. Aplicações diferentes podem inserir novos protocolos na rede especificando as rotinas que são executadas nos nós que encaminham as mensagens. As redes podem se adequar às aplicações incorporando processamento normalmente feito em sistemas fins, ou mesmo novos tipos de processamento que só fazem sentido em redes ativas.

O projeto de ANTS se fundamenta em três objetivos [29]:

- Os nós da rede devem suportar uma variedade de diferentes protocolos de rede sendo utilizados simultaneamente;
- Novos protocolos podem ser aceitos apenas pelo consentimento mútuo entre as partes interessadas, ao invés de precisar de um registro centralizado ou o consentimento de todas as partes;
- Novos protocolos devem ser implementados dinamicamente e sem a necessidade de colocar toda ou parte da rede *off-line*.

São utilizados três componentes-chave na arquitetura para se conseguir esses objetivos: (1) Os pacotes das redes tradicionais são substituídos por ***cápsulas,*** que se referem ao processamento a ser feito para elas. (2) Roteadores e nós fins são substituídos por ***nós ativos,*** que executam as rotinas de processamento das cápsulas e mantêm seu estado associado. (3) Um mecanismo de ***distribuição de código*** garante que as rotinas de processamento sejam transferidas automaticamente e dinamicamente para os nós onde são necessárias [4].

3.2.1 Protocolos e Cápsulas

A utilização dos elementos programáveis da rede é feita através de um modelo que combina as rotinas de encaminhamento nos nós individuais dentro de um padrão de comportamento. Estes padrões definem o processamento que ocorre na rede como um todo e devem ser separados uns dos outros. Para que esse objetivo seja alcançado são utilizados cápsulas, grupos de códigos e protocolos, que estão representados na Figura 2.

- Cápsulas: têm como principal função incluir uma referência à rotina de encaminhamento que é processada em cada nó ativo. Existem rotinas de encaminhamento que são encontradas em todos nós ativos. Outras, porém, são específicas da aplicação, não constam em todos os nós, mas são transferidas a um nó, se for necessário;
- Grupos de códigos: conjunto de tipos de cápsulas relacionadas cujas rotinas de encaminhamento são transferidas como uma unidade pelo sistema de distribuição de código;

- Protocolos: é um conjunto de grupos de códigos relacionados que são tratados pelos nós ativos como uma única unidade de proteção.

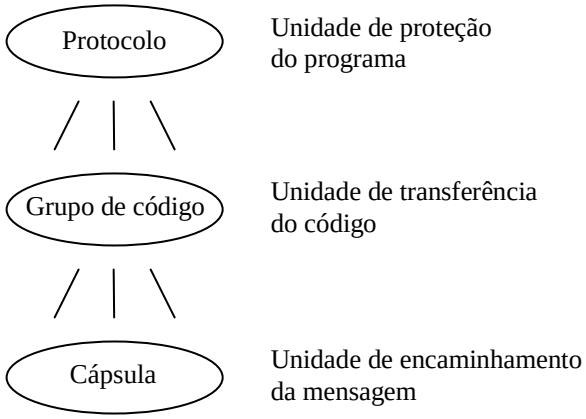

Figura 2. Hierarquia de composição de cápsulas.

3.2.1.1. Formato da Cápsula

A Figura 3 descreve o formato da cápsula. Cada cápsula tem um identificador único para seu protocolo e tipo particular de cápsula naquele protocolo.

Este identificador é derivado da descrição do código do protocolo e serve para duas razões básicas: reduzir enormemente o perigo de protocolos fraudados e permitir que protocolos e tipos de cápsulas sejam alocados rapidamente e de forma descentralizada.

O restante do formato da cápsula é composto de um cabeçalho compartilhado, que contém campos em comum a todas as cápsulas; um cabeçalho tipo-dependente e que deve ser atualizado à medida que a cápsula atravessa a rede; e um campo de mensagem, que é onde se encontra a informação a ser transmitida.

Protocolo / Cápsula	Cabeçalho Compartilhado	Resto do cabeçalho ... Mensagem

Figura 3. Formato da cápsula.

3.2.2 Nós Ativos

Os nós ativos são os responsáveis por implementar protocolos dentro de um ambiente restrito, que limita o acesso a recursos compartilhados. Eles exportam um conjunto de primitivas que são usadas nas rotinas de processamento definidas na aplicação, fornecendo os recursos compartilhados entre os protocolos e estabelecendo restrições para utilização destes recursos.

3.2.2.1. Primitivas do Nó

É o conjunto de primitivas disponíveis nos nós ativos que determina os tipos de rotinas de processamento que podem ser implementadas pelas aplicações:

- Acesso ao ambiente: serve para controlar a localização do nó, o estado dos *links*, as tabelas de roteamento, hora local e assim por diante;
- Manipulação da cápsula: para acessar tanto os campos de cabeçalho quanto o campo de mensagem;
- Controle de operações: para permitir que cápsulas criem outras cápsulas e encaminhem, copiem e descartem a si mesmas;
- Armazenamento de nós: para manipular objetos definidos na aplicação que são mantidos por um curto intervalo de tempo.

3.2.2.2. Modelo de Execução

Suporta uma forma geral de encaminhamento de pacotes, além da atividade computacional em geral. Possui as seguintes características:

- A rotina de encaminhamento da cápsula é configurada no remetente e não pode mudar durante o percurso pela rede; cápsulas pertencentes a um protocolo não podem criar cápsulas pertencentes a um outro protocolo;
- Nem todos os nós da rede precisam executar uma rotina de encaminhamento em particular. Rotinas de encaminhamento podem auto-selecionar os nós nos quais é útil realizar processamento especializado dependendo da localização do nó e de suas capacidades;
- Uma vez que rotinas de encaminhamento podem ser definidas por usuários não confiáveis, elas têm capacidade limitada;

- Os dados que uma cápsula pode acessar enquanto na rede são determinados pelo protocolo a que ela pertence. Normalmente, somente cápsulas pertencentes ao mesmo protocolo devem compartilhar o mesmo estado.

Quando uma cápsula chega em um nó, sua rotina de processamento é executada totalmente (a menos que exceda seu limite de recursos). A rotina processa o campo de mensagem da cápsula e inicia qualquer outra ação necessária, como encaminhamento.

Durante o processamento da cápsula, os nós ativos são responsáveis pela integridade da rede e por tratar qualquer erro que apareça. Para garantir que a rede seja robusta e seus recursos estejam alocados de forma autenticada, um limitador de recursos é incluído no cabeçalho compartilhado da cápsula e funciona como um campo TTL (*Time To Life*) genérico. Este limite é decrementado pelos nós à medida que os recursos são consumidos, e os nós descartam cápsulas quando os seus recursos chegarem a zero.

3.2.3 Distribuição de Código

Para se ter uma infra-estrutura programável, é necessário possuir um mecanismo de propagação dos códigos para onde eles são necessários. Um bom esquema deve ser eficiente, adaptar-se às mudanças nos nós conectados e limitar sua atividade para que a rede permaneça robusta.

Muitos mecanismos são possíveis. Em um extremo, o programa pode ser transmitido para todas as cápsulas. Esse esquema somente é bem sucedido em transferências de programas extremamente curtos e quando a largura de banda não é escassa. No outro extremo, os programas podem ser pré-carregados em todos os nós que podem necessitar deles, o que seria inviável em um cenário com muitos dispositivos. Nenhuma dessas abordagens serve ao objetivo de implementação rápida e descentralizada.

A abordagem utilizada é de associar a transferência de códigos com a de dados, como em uma função dentro da rede. Sabe-se que tal mecanismo é vantajoso, uma vez que ele limita a distribuição de código somente para onde é necessário [4].

O esquema construído transmite os códigos sob demanda e armazena-os em um *cache* para melhorar a performance nos casos comuns de fluxo (seqüências de cápsulas que seguem o mesmo caminho e precisam do mesmo processamento). As cápsulas de um novo tipo podem ser injetadas na rede. Elas viajam através dos nós da rede. Um protocolo leve (que não

sobrecarrega a rede) é usado para transmitir os programas das cápsulas de um nó para o seguinte, onde elas são armazenadas para um uso futuro.

A seqüência dos eventos desse protocolo de transmissão sob demanda está listada abaixo e representada na Figura 4:

- As cápsulas identificam seus tipos e o protocolo aos quais elas pertencem. Essa informação é inalterável por instância da cápsula;

- Quando uma cápsula chega a um nó, o *cache* do código de protocolo é checado. Se o código requerido não está presente, é enviado para o nó anterior um pedido de carga, baseado no tipo de cápsula e protocolo. A execução da cápsula é suspensa por um certo tempo, aguardando o código;

- Quando um nó recebe um pedido de carga que pode responder, ele o faz imediatamente. É enviada a mensagem que contém todo grupo de código necessário;

- Quando o nó recebe a resposta do pedido de carga, ele incorpora o código em seu *cache*. Se os códigos requeridos estão todos presentes, a cápsula que estava na espera é "acordada". Se a resposta do pedido não foi bem sucedida, a cápsula é descartada.

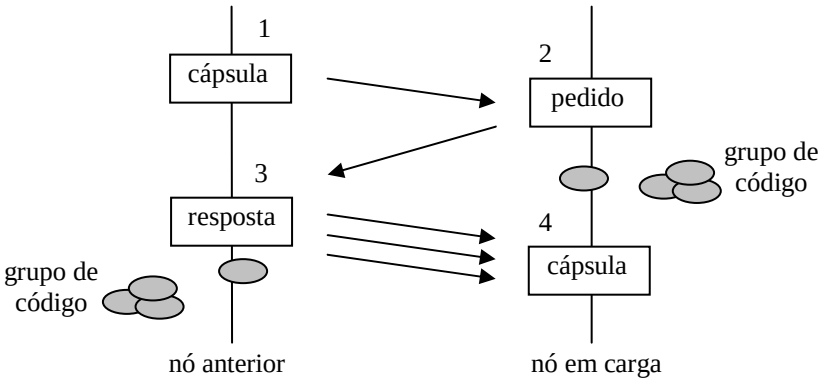

Figura 4. Distribuição do código.

3.3 Visão do Modelo de Programação

O pacote ANTS foi desenvolvido em Java e é executado como um processo a nível de usuário no sistema operacional Linux [5]. O protocolo de distribuição de código transfere as rotinas de processamento em arquivos de classes Java. Esta linguagem foi escolhida pelo suporte para segurança e mobilidade (através de *bytecodes* e sua verificação). Todas as implementações

desenvolvidas devem transferir as definições do protocolo através da rede utilizando tecnologia de código móvel. No protótipo do ANTS, inclusive as aplicações e os nós ativos são também escritos em Java [29].

O ANTS utiliza o projeto Janos (*Java-oriented Active Network Operating System*), da *University of Utah*, para acessar o ambiente de rede, criar *threads*, etc. [30]. O objetivo do projeto Janos é desenvolver um sistema operacional local para nós de redes ativas, baseado na linguagem Java. O foco principal são as pesquisas de gerenciamento e controle de rede. O projeto possui objetivos secundários nas áreas de segurança da informação, análise de desempenho e tecnologia de transferência de componentes de código. Devido suas características, o projeto Janos também é conhecido por Janos Java NodeOs. A Figura 5 mostra a arquitetura de camadas do processamento de aplicações desenvolvidas neste *framework* ativo, onde é necessário uma máquina virtual Java (JVM) e o Janos para a implantação do ambiente ANTS.

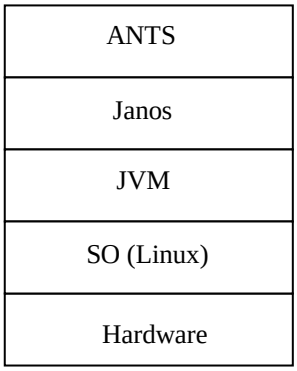

Figura 5. Camadas de processamento existentes na arquitetura proposta.

Uma rede ANTS é customizada por seus usuários nos termos do protocolo. Novos protocolos (e aplicações que os usam) devem ser desenvolvidos pelos usuários através de heranças das classes virtuais providas especificamente para este propósito. Os novos protocolos (e aplicações) são então usados pelas instâncias manipuladas por estas novas subclasses juntamente com um ambiente ANTS em tempo de execução. Deste modo, tem-se:

- Um novo protocolo é desenvolvido pela extensão da classe virtual *Protocol*. Isto requer todas as identificações dos diferentes tipos de pacotes que entram na rede por

suas diferentes rotinas de encaminhamento. Cada tipo de pacote e suas rotinas de encaminhamento são especificados pela subclasse da classe virtual *Capsule*;

- Uma nova aplicação é desenvolvida pela extensão da classe virtual *Application*;

- Uma nova instância da classe *Node* representa um local em tempo de execução do ANTS;

- Um novo protocolo e aplicação são usados pelas instâncias criadas por suas classes e incorporadas ao nó. A aplicação é conectada em um nó na ordem de enviar e receber cápsulas da rede. O protocolo é registrado com o nó para que a rede seja capaz de obter seu código quando for necessário.

Esse modelo permite a customização pelos usuários da rede, assumindo que a rede de nós ativos já exista e se encontra em tempo execução.

3.4 Níveis de Segurança

Se não for bem projetada, uma rede ativa pode sofrer problemas de segurança, pois tais redes podem dar permissão a um código não confiável ser executado. Para avaliar o ANTS neste aspecto, foram descritas as ameaças que os novos serviços apresentavam, como eles eram tratados, e como eles se comportam comparados aos protocolos padrões da Internet [35]. Através deste processo de avaliação, constatou-se que os conceitos de segurança permaneceram semelhantes ao da Internet. A maioria das ameaças pode ser tratada pelos nós ativos localmente, através das primitivas citadas no item 3.2.2.

Na implementação do ANTS, o primeiro nível de segurança é baseado na linguagem de programação usada (ou seja, Java), a qual define as regras de acesso. As cápsulas e serviços ANTS podem ser manipulados somente através de uma interface de API pública. Desta maneira, métodos essenciais podem ser declarados de forma que subclasses não possam reimplementá-los. Além disso, a máquina virtual Java desenvolve verificação de *bytecode* para checar se o código vem compilado conforme a especificação da linguagem [36].

O ANTS adiciona um segundo nível de segurança para prevenir que cápsulas invoquem códigos pertencentes a um protocolo não associado [36]. Conseqüentemente, toda cápsula transmite um identificador encriptado, referente ao protocolo, para assegurar um alto nível de segurança.

Neste capítulo foram descritos a arquitetura e todo funcionamento do *framework* com o pacote ANTS, salientando inclusive os níveis de segurança e a visão de programação. Após este estudo ter sido feito, a implementação propriamente dita da coleta ativa foi executada. Portanto, no próximo capítulo, será detalhado o mecanismo de coleta proposto e implementado nesta dissertação.

4 Mecanismo de Coleta Ativo

4.1 Introdução

Uma vez que o ANTS se trata de um pacote de classes Java que traz funcionalidade de redes ativas, como a que permite que os usuários insiram nos nós ativos rotinas para realizar atividades computacionais através de cápsulas (pacotes tradicionais), de forma flexível e descentralizada, utilizou-se desta ferramenta para a evolução de um processo de coleta que visa a descentralização e a quebra do paradigma dos modelos cliente-servidor de protocolos tradicionais de gerência, cujos cenários favorecem o aumento no tráfego da rede em questão.

Para a criação de aplicações utilizando o *kit* de ferramentas do ANTS, devem ser empregadas as classes Java deste pacote para facilitar a adição de propriedades fundamentais em um ambiente ativo. Isto é feito através das características de portabilidade de uma linguagem orientada a objetos. Como foi visto na seção 3.3, pode-se utilizar os conceitos de herança desta linguagem para a criação das classes de uma aplicação e de um novo protocolo.

Desta maneira, fez-se um protótipo que apresenta vantagens significativas no processo de obtenção de informação de objetos MIB utilizando o protocolo SNMP para a coleta local, com o objetivo de minimizar os tempos de gerência. Nas seções seguintes será descrito com detalhes o mecanismo de coleta ativo aqui defendido e como foi implementado este protótipo.

4.2 Arquitetura do Mecanismo

O *framework* ativo deste trabalho é executado em máquinas com o sistema operacional Linux, ou equivalente, com o pacote ANTS instalado, que está na versão 2.0.3 e disponível em [37]. Vale salientar que este ambiente de redes ativas é totalmente voltado para a linguagem de programação Java. O módulo implementado é composto de *scripts* (que são responsáveis pela

configuração dos nós que são monitorados e pelo roteamento das cápsulas da aplicação) e de classes Java (que são responsáveis pela estruturação das cápsulas e pelo processo de coleta de dados propriamente dito).

A aplicação proposta interage com os agentes SNMP para a obtenção de informações das MIBs nos nós analisados. A Figura 6 mostra como funciona a coleta inicializada por uma estação de gerência. Ela envia cápsulas com rotinas que devem ser executadas no nó monitorado para obter as informações desejadas.

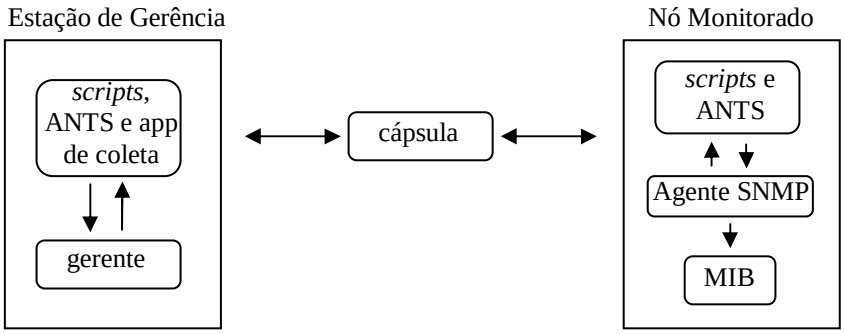

Figura 6. Processo de coleta de dados.

Essa arquitetura teve como motivação a diminuição da troca de mensagens entre o gerente e os agentes, que normalmente existe em um modelo tradicional de gerência. Estas excessivas trocas ocorrem porque os protocolos utilizados possuem limitadas operações para obtenção de dados. Estas limitações serão mais bem exploradas no capítulo 5, onde será feito um breve estudo sobre o protocolo SNMP e suas operações, de formar a subsidiar os parâmetros para os estudos de desempenho do protótipo exposto.

Além da vantagem da arquitetura de possibilitar a obtenção de vários objetos de uma MIB com o envio de apenas uma cápsula, a dissertação em questão defende a idéia de aproveitamento das características de distribuição de redes ativas para, assim, conseguir alcançar uma melhor desempenho na atividade de captura de dados necessária para a gerência da rede.

A Figura 7 descreve um modelo centralizado de gerência, geralmente encontrado nas redes atuais, que aplicam protocolos tradicionais de gerência como o CMIP e, com maior freqüência, o SNMP.

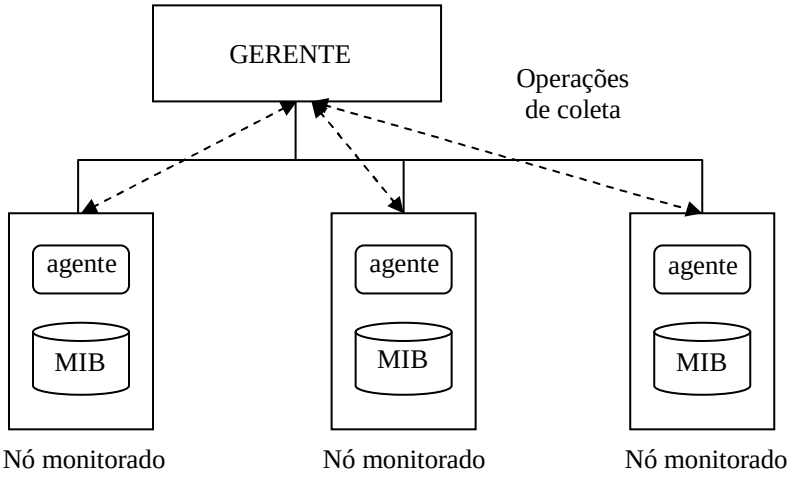

Figura 7. *Framework* tradicional de rede utilizando protocolos centralizados.

Neste ambiente, que está baseado no modelo cliente-servidor, há um grande número de troca de mensagens, pois o gerente deve interagir com todos os dispositivos gerenciados visando obter as informações de gerenciamento da MIB destes agentes. Esta troca de mensagens pode resultar em vários problemas de desempenho, uma vez que a centralização da gerência sobrecarrega o próprio gerente e as excessivas mensagens entre o gerente e o agente aumentam de forma substancial o tráfego da rede.

Pode-se colocar a descentralização do processo de gerenciamento como a maior vantagem deste modelo, pois as gerências tradicionais são ineficientes para implementar um cenário eficaz de gerência distribuída. Devido a isso, foi explorada a capacidade de flexibilidade e distribuição da tecnologia ativa e desenvolveu-se uma ferramenta onde pudesse ser utilizada a gerência descentralizada.

Como exposto anteriormente, a forma tradicional de se obter dados de uma MIB é centralizada, com o gerente tendo que capturar informações de todos os dispositivos por ele monitorado. No caso da implementação em questão, um nó ativo, não obrigatoriamente o gerente, pode enviar uma cápsula para efetuar a coleta em um outro nó ativo e, dessa forma, uma única cápsula pode efetuar a operação de coleta em vários dispositivos, como está representado na Figura 8.

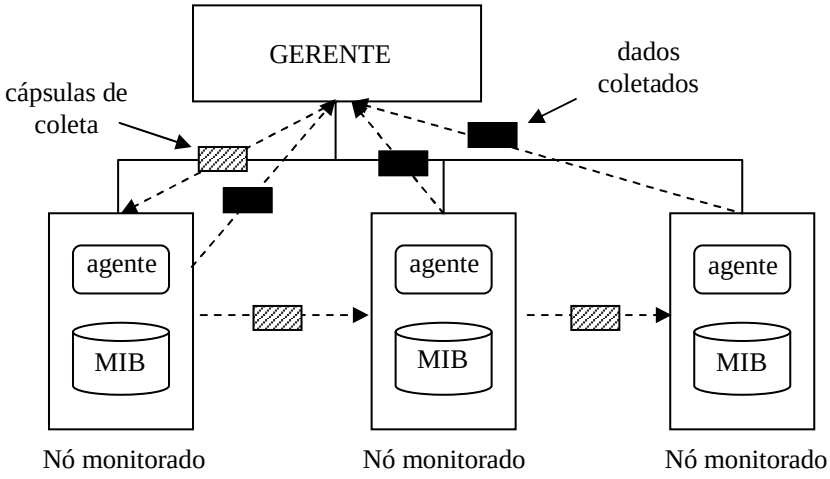

Figura 8. *Framework* de coleta de dados distribuída.

Quando o processo é inicializado, uma única cápsula, a ser enviada a um dos agentes, pode adquirir informações de todos os nós monitorados. Isto é possível porque um agente também tem a capacidade de enviar cápsulas de coletas para os outros agentes. Os resultados obtidos retornam à estação de gerência, também através de cápsulas. No momento que a captura dos dados é finalizada em um nó, uma cápsula de retorno é criada e enviada ao nó gerente.

Por conseguinte, o mecanismo de coleta apresentado aumenta o desempenho de captura de informações em uma MIB, tendo basicamente dois objetivos: diminuir o número de trocas de mensagens de gerência e descentralizar as coletas.

4.3 Alterações Necessárias

O pacote ANTS é composto por uma estrutura de diretórios, mostrada na Figura 9. Os diretórios mais relevantes desta estrutura e seu conteúdo estão descritos a seguir:

- *autoconf*: contém arquivos usados pelo *script configurate*;
- *doc*: contém a documentação do projeto ANTS;
- *lib:* contém arquivos *.jar* e *.zip* para a biblioteca de simulação do nó ativo;
- *scripts:* contém uma variedade de *shell scripts* usados para executar e configurar as aplicações de exemplo;

- *src*: contém o código fonte Java das aplicações ANTS e dos nós ativos.

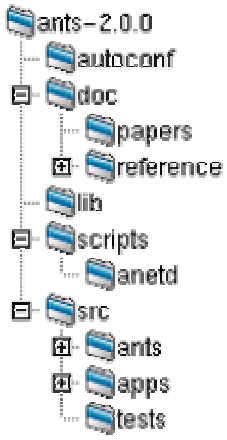

Figura 9. Estrutura de diretório do ANTS

Para o desenvolvimento de uma aplicação/protocolo, algumas alterações básicas foram feitas nos arquivos contidos na estrutura de diretórios mostrada na Figura 9.

A primeira alteração feita foi no arquivo *makefile.in* (encontrado em /ants-2.0.2/src), pois neste arquivo são especificados os diretórios de todos os arquivos fonte *.java* das aplicações e do pacote ANTS. A partir deste arquivo foi criado um outro (*makefile*), para ser utilizado na hora da compilação do pacote. Portanto, foi preciso editar este arquivo para acrescentar a nova aplicação de coleta desenvolvida.

Dentro da pasta /ants-2.02/src/ants se encontram todos os pacotes disponibilizados pelo *kit* de ferramentas ANTS. Não consta originalmente neste pacote o suporte às operações do protocolo SNMP. Então, a segunda modificação que foi preciso ser feita foi a adição deste pacote [38], o que possibilitou a execução da coleta SNMP local pelas cápsulas construídas através de uma aplicação do ANTS.

Decorrente da adição do pacote SNMP, houve a necessidade de uma outra modificação no código fonte da classe *PrimordialNode* do programa *PrimordialNode.java* (situada em /ants-2.0.2/src/snts/core). Esta alteração foi necessária pelo fato de ser esta a classe responsável pela exportação de outras classes do ANTS que precisam ser executadas por uma rotina (contida em uma cápsula) em um determinado nó remoto.

4.4 Implementação da Coleta Ativa

O primeiro passo da implementação foi estabelecer um ambiente ativo ANTS, pois, para que uma aplicação ativa execute as rotinas desejadas em um nó, utilizando este *framework*, é necessário que o nó citado encontre-se ativo. Este processo de ativação do nó é feito através da execução do *script antsvm* (em /ants-2.0.2/scripts) que pode ser descrito como um simples empacotador, atuando sobre uma JVM (*Java Virtual Machine*), que adiciona as classes ANTS no *classpath* e define as variáveis do ambiente Java. Este *script* usa como entrada uma das principais classes do pacote ANTS: a *ConfigurationManager* (cujo código fonte está no arquivo *ConfigurationManager.java,* situado em /ants-2.0.2/src/snts/core). A operação de ativar um nó compreende a associação de um IP virtual e de uma porta a uma determinada aplicação ativa neste nó. Essa configuração é passada como parâmetro através do script *coleta.config* (situado em /ants-2.0.2/scripts) para a classe *ConfigurationManager*. Assim sendo, quando a cápsula de uma determinada aplicação chega em um nó que está ativo, ela executa as rotinas desejadas.

Um outro fator a ser destacado são as rotas que devem ser seguidas por uma cápsula. O roteamento deve ser pré-estabelecido em um *script* chamado *coleta.routes* (situado em /ants-2.0.2/ scripts), que pode ser gerado automaticamente pelo ANTS através de parâmetros também encontrados no *coleta.config*. A geração automática do *coleta.routes* é feita utilizando o executável *makeroutes.sh* (situado em /ants-2.0.2/ scripts).

Após este cenário ter sido construído, com a geração de *scripts* e o pacote ANTS, trabalhou-se na aplicação propriamente dita. Ela é composta essencialmente por três classes distintas: *ColetaApplication, ColetaCapsule* e *ColetaReturnCapsule,* contidas nos códigos fontes *ColetaApplication.java*, *ColetaCapsule.java* e *ColetaReturnCapsule.java*, respectivamente, que, por sua vez, estão situadas em /ants-2.0.2/src/apps/coleta.

A classe *ColetaApplication* tem como funções: a inicialização da captura de dados, a gerência do envio e recebimento das cápsulas e a finalização de todo o processo no nó gerente. A *ColetaCapsule* é responsável por criar as cápsulas contendo o código de coleta local que são enviadas aos nós monitorados, sendo possível o envio de novas cápsulas de coleta por estes próprios nós, caracterizando o modelo distribuído. Por fim, há a *ColetaReturnCapsule*, que deve criar as cápsulas que retornam os resultados e tratar os valores retornados para o nó gerente. Além das classes citadas acima, existem três outras

classes que compõem a aplicação de coleta de dados ativa: *ColetaProtocol*, *ColetaCapsuleCacheKey* e *ColetaCapsuleCacheValue*.

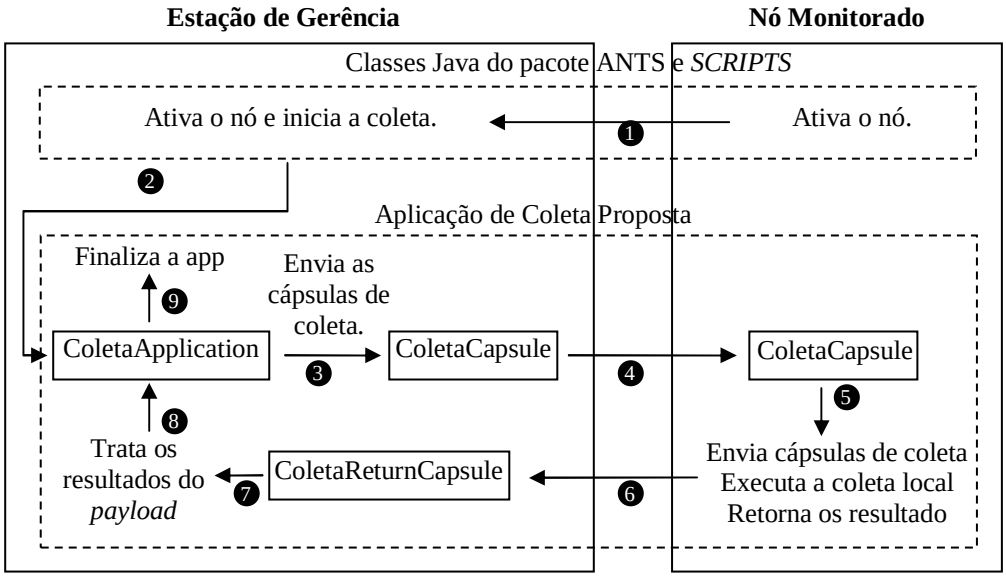

Figura 10. Modelo de execução do mecanismo de coleta proposto.

Os passos seguintes se referem ao modelo de execução deste mecanismo de coleta ativo, representado na Figura 10:

- O primeiro passo é ativar todos os nós. Para isso, é necessária a utilização da classe *ConfigurationManager* do pacote ANTS Essa classe recebe como parâmetros o IP virtual vinculado ao nó a ser ativado e o arquivo de configuração;

- O último nó a ser ativado deve ser o responsável pela coleta (gerente). Através da configuração do *script coleta.config*, o ANTS sabe quando uma aplicação de coleta deve ser inicializada em um determinado nó.

- Uma vez que todos os nós estiverem ativos e a aplicação inicializada, as cápsulas de coleta são criadas e enviadas (pela classe *ColetaCapsule*) a todos os nós conectados ao nó gerente que foram especificados no *script* de configuração *coleta.config*;

- Quando uma cápsula chega em um nó, a classe *ColetaCapsule*, usando o *script* de rotas *coleta.routes*, verifica se este nó possui vizinhos. Caso afirmativo, novas cápsulas de coleta são criadas e enviadas a estes a partir dos nós que estão sendo

monitorados (agentes), da mesma forma como foi feito no passo anterior. Durante este processo, é feito um controle através *caches* que será melhor descrito na seção 4.4.7;

- Após a verificação de vizinhos, a coleta local é realizada pela classe *ColetaCapsule*. A coleta é referente às OIDs (*Object Identifier*) especificadas no arquivo de configuração *coleta.config*;

- Os resultados são enviados ao nó gerente, também através de cápsulas. Quando a classe *ColetaCapsule* termina a coleta local, é feita a chamada da classe *ColetaReturnCapsule*, que cria as cápsulas para o retorno dos resultados e trata estas informações para a exibição ao usuário;

- Após terminado este processo, a classe *ColetaApplication* finaliza a aplicação no nó gerente.

Nas subseções seguintes, serão especificados todos os *scripts* e classes da aplicação de coleta ativa, e, a Figura 11 mostra a estrutura das classes desenvolvidas no decorrer deste trabalho.

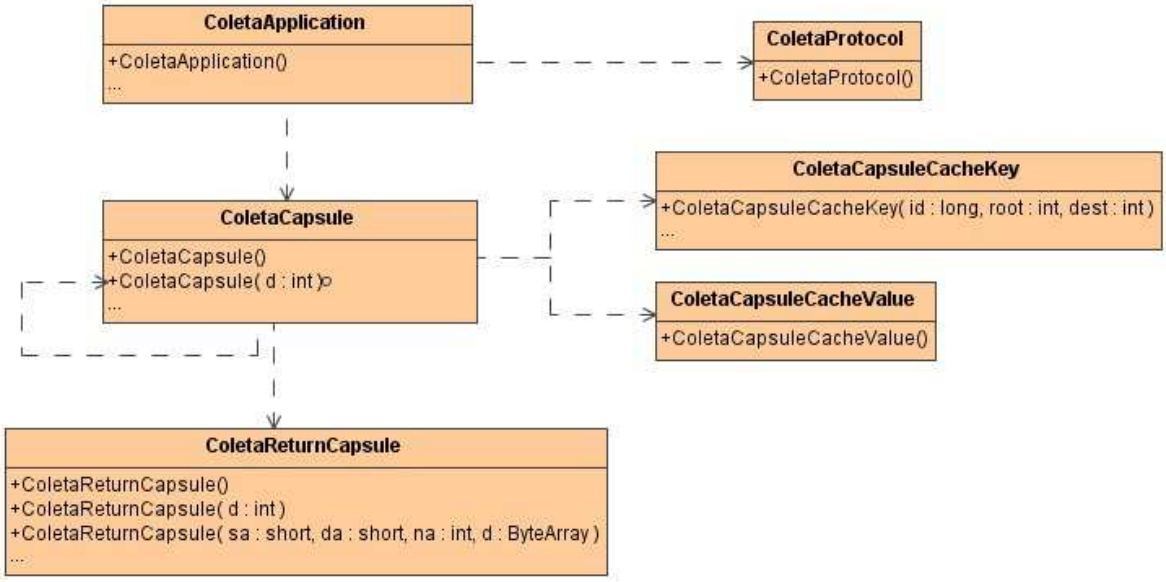

Figura 11. Estrutura de classes.

4.4.1 O Arquivo de Configuração (*coleta.config*)

De acordo com o padrão de nomenclatura estabelecido pelo protótipo do ANTS, um *script* (**.config*) é responsável por toda a configuração de uma aplicação. A partir dele, a classe *ConfigurationManager* recebe os parâmetros necessários para a execução de uma aplicação ativa.

A Figura 12 refere-se a um arquivo de configuração da aplicação de coleta proposta. Pode-se observar que neste arquivo (linhas 5, 8 e 10) estão associados ao IP real um IP virtual e uma porta, para cada nó. Nas linhas 4, 7 e 9, é passado como parâmetro o arquivo que contém as rotas necessárias para o encaminhamento das cápsulas, no caso descrito, o *coleta.routes*.

Durante a ativação do nó pelo *ConfigurationManager*, é feita a verificação a qual IP virtual este nó está associado. Caso, nesta verificação, perceba-se que o nó que está sendo ativado é o nó que possui o IP virtual que inicia um processo de coleta de dados, a aplicação é inicializada. Pode-se observar na linha 6 da Figura 12, a especificação de que na ativação do nó de IP virtual 18.31.12.1 é executada a aplicação *ColetaApplication*, do pacote *apps.coleta*, encontrada em /ants-2.0.2/src/apps/coleta. Durante a chamada da aplicação através desta linha, as OIDs a serem coletadas são especificadas. O arquivo de configuração da Figura 12, por exemplo, passa as OIDs 1.3.6.1.2.1.1.1.0 (correspondente a mib2.system.sysDescr.0) e 1.3.6.1.2.1.1.2.0 (mib2.system.sysObjectID.0).

O arquivo de rotas estabelece todo o processo de distribuição das cápsulas. Desta forma, através da lógica da aplicação de coleta, pode-se estabelecer um processo de captura de dados de forma centralizada ou distribuída, dependendo tão somente das rotas definidas neste arquivo.

Como citado anteriormente, este arquivo de rotas é criado pela execução do *script makeroutes* provido pelo pacote ANTS (/ants-2.0.2/scripts). Os parâmetros requeridos por este executável, que são fornecidos pelo arquivo de configuração (*coleta.config*), equivalem às linhas 11 e 12 da Figura 12.

```
1.  #
2.  # coleta.config criado pelo aplicativo CriarConfig em 10/02/2003
3.  #

4.  node 18.31.12.1 -routes coleta.routes -log 255 -consoleport 2050
5.  channel 18.31.12.1 200.17.37.34:8004 -log 255
6.  application 18.31.12.1 apps.coleta.ColetaApplication -port 5000
    -oids 1.3.6.1.2.1.1.1.0,1.3.6.1.2.1.1.2.0,

7.  node 18.31.12.2 -routes coleta.routes -log 255 -consoleport 2051
8.  channel 18.31.12.2 200.17.37.35:8005 -log 255

9.  node 18.31.12.3 -routes coleta.routes -log 255 -consoleport 2052
10. channel 18.31.12.3 200.17.37.36:8006 -log 255

11. connect 18.31.12.1 18.31.12.2
12. connect 18.31.12.2 18.31.12.3
```

Figura 12. *Script* de configuração (*coleta.config*).

4.4.2 O Arquivo de Rotas (*coleta.routes*)

```
1.  #
2.  # shortest routes, automatically generated
3.  # by ./makeroutes on 02/10/03
4.  #

5.  # source      destination       next        addr

6.  18.31.12.1    18.31.12.2        18.31.12.2  200.17.37.35:8005
7.  18.31.12.1    18.31.12.3        18.31.12.2  200.17.37.35:8005
8.  18.31.12.2    18.31.12.1        18.31.12.1  200.17.37.34:8004
9.  18.31.12.2    18.31.12.3        18.31.12.3  200.17.37.36:8006
10. 18.31.12.3    18.31.12.1        18.31.12.2  200.17.37.35:8005
11. 18.31.12.3    18.31.12.2        18.31.12.2  200.17.37.35:8005
```

Figura 13. *Script* de rotas (*coleta.routes*)

O arquivo *coleta.routes*, mostrado na Figura 13, foi criado pelo executável *makeroutes* que se encontra no diretório de *scripts* do ANTS. Outra forma de se obter o arquivo é escrevendo-o manualmente. Para a criação do arquivo de rotas através do *makeroutes* é preciso ter como entrada de dados o *script coleta.config*, detalhado na seção anterior. No caso, as rotas criadas no arquivo da Figura 13, foram obtidas a partir das linhas 11 e 12 da Figura 12.

Uma das grandes vantagens de uma plataforma ativa é a possibilidade de se descentralizar qualquer protocolo. Para tanto, tal característica é dada no ANTS através da

especificação destas rotas. Isto ocorre por tratar-se de um ambiente programável, onde pode haver um tratamento para o envio de cápsulas, subsidiado pelas especificações das conexões estabelecidas entre os nós ativos. Este tratamento será melhor detalhado na seção 4.4.5.

4.4.3 Classe *ColetaProtocol*

A classe *ColetaProtocol* agrupa os tipos de cápsulas: *ColetaCapsule*, *ColetaReturnCapsule*, *ColetaCapsuleCacheKey* e *ColetaCapsuleCacheValue*, relacionadas em grupo de código. Esta é a forma básica para a propagação dos códigos das cápsulas: pela difusão nos vizinhos da rede na qual um grupo de código é usado.

Assim sendo, basta que as classes da aplicação sejam encontradas nos nós que iniciam o mecanismo de coleta, pois as cápsulas que estão em um grupo de código são distribuídas pelos nós vizinhos da rede onde suas rotinas devem ser executadas. Este processo de distribuição ocorre como o já discutido na seção 3.2.3.

É importante salientar que, apesar de ser desnecessário que as classes da aplicação encontrem-se em todos os nós participantes do monitoramento, há a necessidade da presença dos arquivos de configuração e de rotas (*coleta.config* e *coleta.routes* respectivamente) nos nós a serem ativados pelo *ConfigurationManager*.

4.4.4 Classe *ColetaApplication*

A classe *ColetaApplication* desencadeia todo processo de coleta. Quando a classe *ConfigurationManager* é executada em um nó (que está associado à chamada da aplicação de coleta) é inicializado o monitoramento das OIDs passadas como argumento a esta aplicação. Estas OIDs estão em um único parâmetro *string*, separadas umas das outras por vírgulas. É feito o tratamento deste parâmetro, separando as OIDs. Após isso, a aplicação cria as cápsulas de coleta, através da classe *ColetaCapsule*, com campos que contém as OIDs a serem monitoradas. Esta é a classe responsável pela coleta local. As cápsulas de coleta são então enviadas para todos os nós que estão conectados ao gerente, de acordo com as conexões especificadas no arquivo de configuração.

4.4.5 Classe *ColetaCapsule*

Como se trata de uma classe estendida da classe *Capsule* do pacote ANTS, a *ColetaCapsule* herdou as características de criação de cápsulas e de manipulação de seus campos. Quando

esta classe é inicializada pela *ColetaApplication*, é criada uma cápsula contendo os campos referentes ao controle de encaminhamento das cápsulas (por exemplo: o endereço do nó gerente para onde é encaminhado os resultados solicitados; referências do protocolo; campos para a inspeção de *timeouts* e do tempo de vida da cápsula) e campos que possuem particularidades da coleta (como o número de objetos a serem coletados e as OIDs destes objetos).

Esta classe também cuida do processo de distribuição de cápsulas aos nós vizinhos de um agente. Para isso, é feito um controle de *cache*, com a ajuda das classes *ColetaCapsuleCacheKey* e *ColetaCapsuleCacheValue* (como será melhor esclarecido na seção 4.4.7). Após este controle, são criadas cápsulas idênticas às recebidas pelo nó e enviadas aos seus vizinhos que ainda não tenham recebido solicitação de coleta por esse tipo de cápsula.

O código Java que realiza a coleta local também se encontra nesta classe, que usufrui da distribuição de código fornecida pelo ambiente ANTS para chegar aos nós onde o gerente deseja executar as operações SNMP localmente.

4.4.6 Classe *ColetaReturnCapsule*

Da mesma forma que a *ColetaCapsule*, a classe *ColetaReturnCapsule* possui as propriedades herdadas da classe *Capsule* do pacote ANTS. Esta classe é chamada quando a coleta local for concluída e é necessário que os resultados obtidos sejam devolvidos ao nó gerente.

Então, a *ColetaReturnCapsule* cria cápsulas, com campos de controle e campos que contêm os valores resultantes dos objetos monitorados, e as envia para o nó gerente, onde sofrem um tratamento para que as informações possam ser exibidas adequadamente ao usuário. A rotina responsável por estruturar a forma de apresentação dos resultados também se encontra nesta classe.

4.4.7 Classe *ColetaCapsuleCacheKey* e Classe *ColetaCapsuleCacheValue*

Quando a classe *ColetaCapsule* entra na fase de retransmitir cápsulas aos vizinhos de um agente, é de extrema importância para os níveis de desempenho que um mesmo nó não receba mais de uma vez pacotes contendo pedidos de informações. Para isso, é feito um controle de *cache* através das classes *ColetaCapsuleCacheKey* e *ColetaCapsuleCacheValue*, que retornam as características do nó e o tempo de *cache*, respectivamente. Portanto, quanto a

ColetaCapsule vai retransmitir uma cápsula, verifica-se se já foi enviado esse tipo de cápsula ao nó de destino vizinho. Caso afirmativo, o envio é cancelado e passa-se para a verificação de um novo nó vizinho.

4.4.8 Exemplos de Saídas

Para um melhor entendimento do processamento dos scripts e classes citados acima, é válida a observação das saídas geradas em um gerente (nó de origem das cápsulas) e em um dos agentes (nó de destino das cápsulas) durante a execução de uma coleta de exemplo. As Figuras 14 e 15 trazem, respectivamente, estas saídas.

```
netvista3:/ants-2.0.2/scripts # ./antsvm edu.utah.janos.nodeos.Main
ants.core.ConfigurationManager coleta.config 18.31.12.1
Janos Java NodeOS v1.1.7
ANTS v2.0
Cannot export class groups in emulation.
*** Enviando cápsulas de coleta
*** Enviando o pedido de coleta para 18.31.12.2
*** Recebida a cápsula de retorno de 18.31.12.2
    netvista2
    3477
    3620
*** Recebida a cápsula de retorno de 18.31.12.3
    netvista3
    2747
    2963
*** Coleta finalizada com sucesso.
Shutting down.
```

Figura 14. Saídas de um gerente em uma coleta a dois agentes.

```
netvista2:/ants-2.0.2/scripts # ./antsvm edu.utah.janos.nodeos.Main
ants.core.ConfigurationManager coleta.config 18.31.12.2
Janos Java NodeOS v1.1.7
ANTS v2.0
Cannot export class groups in emulation.
*** ColetaCapsule chega do 18.31.12.1
*** Checando 18.31.12.1
*** Checando 18.31.12.3
*** Enviando pedido de coleta para 18.31.12.3
*** Obtendo o valor correspondente a OID 1.3.6.1.2.1.1.5.0
*** Obtendo o valor correspondente a OID 1.3.6.1.2.1.2.2.1.10.1
*** Obtendo o valor correspondente a OID 1.3.6.1.2.1.2.2.1.16.1
*** Retornando a cápsula de coleta para 18.31.12.1
```

Figura 15. Saídas de um dos agentes.

4.5 Automatização do Arquivo de Configuração

Durante o desenvolvimento do protótipo proposto, notou-se que a maior dificuldade deste mecanismo seria a elaboração do *script* de configuração (*.*config*). Essa dificuldade deve-se à necessidade de conhecimentos da estrutura do *config*. Além disto, por não possuir uma interface amigável, o usuário precisa fazer toda parametrização de forma textual, através de comandos específicos para a criação dos *scripts* ANTS, requerendo um domínio sobre esta pseudo-linguagem.

Em uma rede que esta sendo monitorada, pode haver diversos modos de coletar informações através deste protótipo. As variáveis de diferenciação encontram-se no *script* de configuração. Entre elas, estão as OIDs desejadas, os nós a serem monitorados e o modelo da rotas estabelecidas.

É conveniente, portanto, que o usuário possua a capacidade de ter vários arquivos de coleta (*.*config*), entre os quais ele pode escolher um para especificar qual tipo de informação ele deseja receber e que roteamento o processo deve seguir em um determinado momento. Com esse objetivo, foi implementada uma ferramenta (também utilizando a linguagem Java) que, através da entrada destas variáveis, pudesse ser gerado automaticamente um arquivo "*.*config"* como saída, dispensando assim, conhecimentos técnicos do usuário para a parametrização do arquivo *config*.

Após a criação deste arquivo, basta copiá-lo para todos os nós participantes da coleta, juntamente com o pacote alterado do ANTS, para que possa ser feita a ativação dos nós e a execução da captura dos dados.

A aplicação de criação automática do arquivo *config* é simples e consiste basicamente de três telas, apresentadas com uma interface limpa e intuitiva.

A Figura 16 mostra a primeira tela da ferramenta. Ela contém uma arvore estática com a representação dos objetos da MIB-II, entre os quais os usuários podem selecionar as OIDs desejadas no monitoramento. Após escolher a OID desejada no painel do lado esquerdo da tela, clica-se o botão adicionar ("+") e o código referente vai para o painel da direita. Análogo é o processo de retirada da OID. É importante salientar que, diante da existência de uma outra MIB em um dispositivo, os objetos referentes a esta MIB podem ser capturados, devendo apenas serem especificados manualmente no arquivo de configuração, pois a ferramenta de automatização limita-se a uma árvore estática da MIB-II.

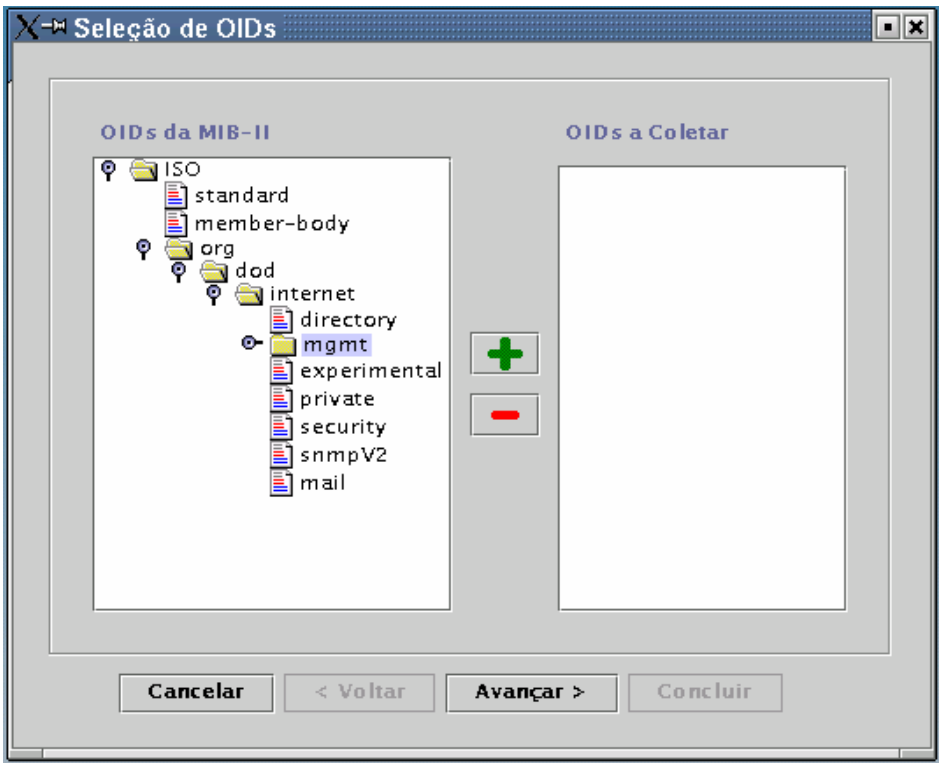

Figura 16. Escolha de OIDs através da ferramenta de automatização.

Um segundo parâmetro fundamental no arquivo de configuração é a configuração dos nós que integram o ambiente de monitoramento. É feita a distinção do gerente e do(s) agente(s) nesta configuração para informar à classe *ConfigurationManager*, na hora da ativação, se aquele nó deve ou não inicializar a execução do mecanismo de coleta. Os demais nós (agentes), têm como resultado da ativação, um estado de *standby,* com a alocação da porta especificada para a espera de cápsulas pertencentes ao protocolo de coleta.

Portanto, a próxima tela do aplicativo é composta para informar as características do nó gerente, que é automaticamente associado à chamada do processo de captura de dados. Nesta fase, também é requerido o estabelecimento das propriedades dos nós agentes, que são as mesmas necessárias para definir o gerente, ou seja, o IP real, o IP virtual e porta de cada agente. A aplicação critica a especificação de dois nós utilizando um mesmo IP virtual e/ou uma mesma porta, pois, no caso de uma simulação, estes dados precisam ser distintos. Pode-se permitir apenas que o nó possua o mesmo IP real, caracterizando um ambiente com NICs virtuais. É visualizada esta tela na Figura 17.

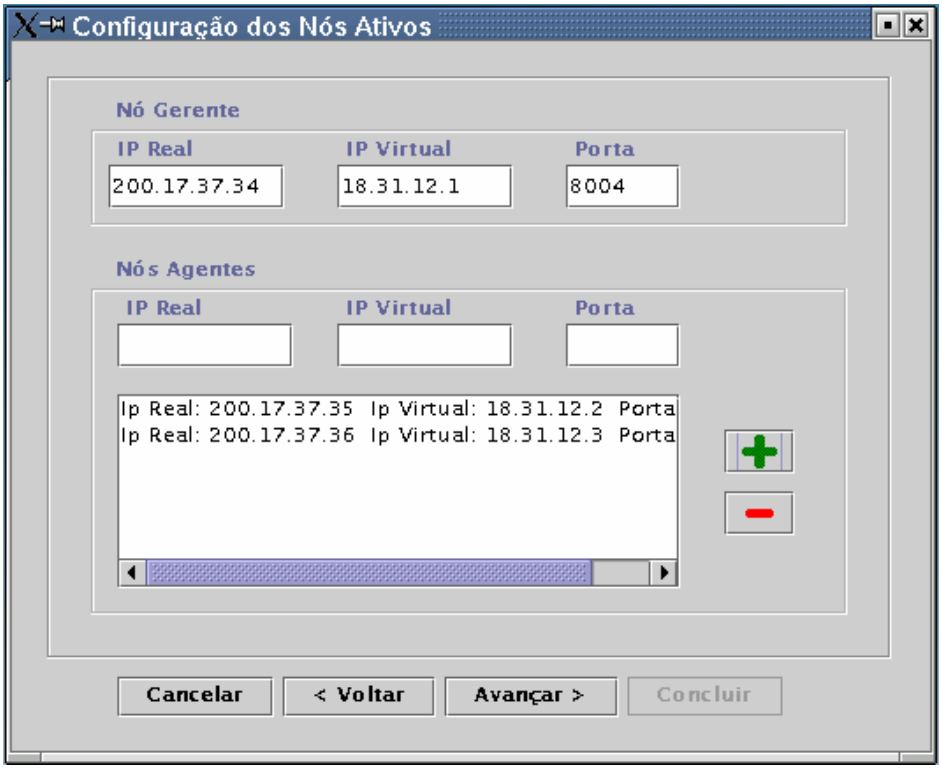

Figura 17. Configuração dos nós ativos através da ferramenta de automatização.

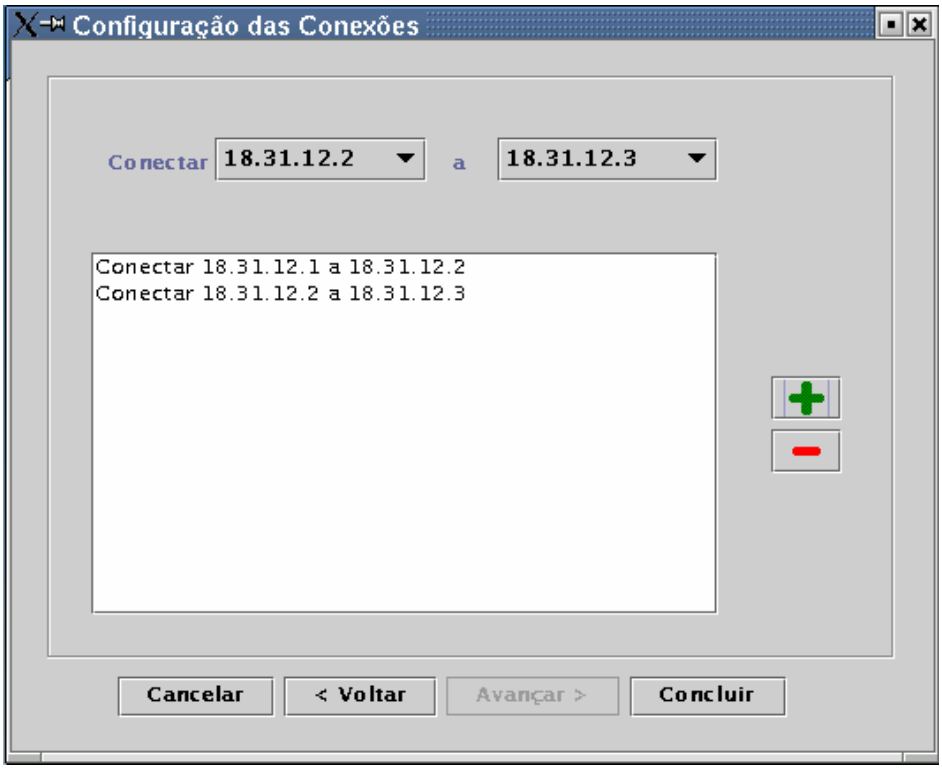

Figura 18. Estabelecimento de rotas e conexões através da ferramenta de automatização.

Como já foi descrito, o arquivo de configuração serve como entrada de dados para o *script makeroutes* gerar as rotas que são seguidas pelas cápsulas nos nós. Trata-se de um importante passo para a criação do arquivo **.config* porque possibilita a utilização da característica de um modelo descentralizado de obtenção de informações de gerência. O estabelecimento da distribuição das cápsulas está relacionado com a estipulação das conexões existentes em um nó. Ressalta-se que estas conexões são meramente atribuídas e não correspondem a meios físicos, embora seja essencial haver um meio de transmissão entre todos os nós abrangidos pelo processo.

O sistema estabelece estas conexões empregando a terceira tela da ferramenta desenvolvida. A referida tela corresponde à Figura 18. No lado esquerdo, o usuário escolhe um nó de origem (já definido na tela anterior). Da mesma forma, escolhe o nó de destino no *ComboBox* da direita e clica em "+" para adicionar o par ao *ListBox*.

Esta tela finaliza o processo. O botão "Concluir" chama o *FileDialogBox* para indicar nome e diretório onde é criado o arquivo de configuração.

4.6 Resumo do Processo de Utilização do Mecanismo de Coleta Ativa

Para a utilização do mecanismo de coleta ativa desenvolvido nesta dissertação, basta que seja montado um cenário usando a plataforma ativa ANTS já alterada para os devidos fins. Portanto, o usuário deste mecanismo coloca, nos *hosts* de sua rede, o pacote ANTS (com as modificações necessárias, como a adição do pacote SNMP e da aplicação de coleta) e o configura e compila através da execução do *script* ANTS: *configure*, e do comando *linux*: *make all*, respectivamente.

Uma observação a ser feita é que, apesar de ter sido considerado que a aplicação faz parte do novo pacote ANTS, ela não é parte essencial em todos os nós participantes do monitoramento, pois, com a existência de um método de distribuição de código (já detalhado na seção 3.2.3), é necessário apenas que a aplicação esteja no nó gerente, uma vez que o código necessário é propagado para os nós onde deve ser executado.

Após a compilação do pacote, os *scripts* de configuração e de rotas devem ser criados. Com a utilização da ferramenta que faz parte do escopo deste trabalho, o processo de edição do *script* de configuração fica mais simplificado. Desta forma, o usuário pode usufruir das

facilidades providas pela ferramenta e criar seus arquivos **.config* referentes às coletas desejadas.

A partir do arquivo **.config*, o usuário deve criar o arquivo **.routes*, que contém as rotas pré-estabelecidas das cápsulas de coleta. O arquivo de rotas pode ser escrito manualmente ou por meio do *script makeroutes* do pacote ANTS, como mencionado anteriormente. Para isto, basta que seja executado o comando: `./makeroutes –route <coleta.config >coleta.routes`. No comando anterior, *coleta.config* é o arquivo de entrada deste *script* e *coleta.routes* o arquivo de saída. Salientando que o nome "coleta" pode ser substituído por qualquer outro da preferência do usuário, basta apenas que, para cada nome *x* do arquivo de configuração, haja um arquivo *x* de rotas referente a ele (*x.config e x.routes*).

Para o bom funcionamento do mecanismo de coleta ativo, estes *scripts* de configuração e de rotas devem estar presentes em todos os nós que participam da coleta. Conseqüentemente, o usuário tem que disponibilizar estes arquivos nos referentes nós.

O usuário deve se assegurar que os nós a serem monitorados possuem um agente SNMP e que o serviço está ativo nestes dispositivos. Esta condição é necessária para que as operações locais SNMP possam ser realizadas.

Feita a disposição correta dos arquivos de *scripts*, das classes da aplicação e do pacote ANTS; o usuário entra no processo de execução da coleta. A partir daí, ele ativa os nós, com o comando: `./antsvm edu.utah.janos.nodeos.Main ants.core.ConfigurationManager <arquivo_de_configuração> <IP_virtual>`. Observando que o último nó a ser ativado é, obrigatoriamente, o gerente, como está especificado no arquivo de rotas. Caso contrário, os agentes ativados após o gerente não farão parte da rede ativa, não podendo receber e/ou enviar as solicitações de coleta.

Em resumo, os passos que um usuário deve seguir para que a captura de informações seja executada com sucesso são:

- Configurar e compilar o pacote ANTS, com as devidas modificações;
- Colocar as classes da aplicação de coleta ativa no nó gerente;
- Criar os arquivos de *script* de configuração e de rotas;
- Distribuir estes *scripts* em todos os nós participantes da monitoração;

- Assegurar que os nós a serem monitorados, possuam um agente SNMP ativo;
- Ativar primeiramente os nós agentes e, por fim, o nó gerente.

5 Cenários e Análise Comparativa

5.1 Cenários e Simulações

Durante os estudos desta dissertação, utilizou-se como auxílio para o trabalho as simulações suportadas por este *framework*. A aplicação de coleta pôde ser executada em uma única máquina simulando a existência de um cenário da aplicação com várias máquinas. Então, a aplicação de coleta foi representada como uma simples redes de três (ou mais) máquinas com endereços virtuais e portas distintas. Exemplo de topologia destas máquinas é apresentado pela Figura 19.

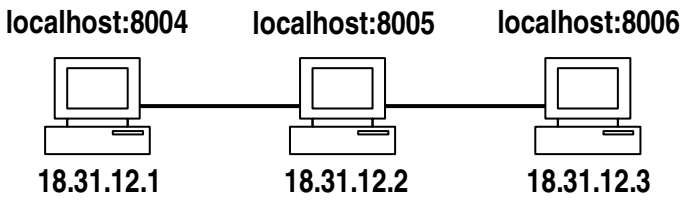

Figura 19. Topologia simulada para a aplicação de coleta.

Os endereços IP são todos virtuais. Toda comunicação é atualmente desenvolvida com o protocolo de transporte UDP (*User Datagram Protocol*) para o endereço *localhost* (127.0.0.1) e uma de suas portas (8004, 8005 ou 8006), como mostrado. Estas portas representam NICs (*Network Interface Card*) virtuais para as máquinas simuladas.

A estrutura dos arquivos que participam das aplicações de exemplo do pacote ANTS encontra-se na Figura 20. A simulação é possível através da classe *ConfigurationManager* (classe que ativa os nós) permitindo que o ANTS e a aplicação sejam executados em único nó. Os exemplos vindos no pacote ANTS utilizam um *script* chamado *nodesetup.sh* que inicia

o *ConfigurationManager* múltiplas vezes. Como entrada deste *script*, tem-se um outro chamado *DefaultEnv*, que monta o ambiente definindo a máquina virtual Java. Nas aplicações exemplo, há os *scripts* exigidos também por aplicações não simuladas como: o arquivo de configuração (**.config*, que passa o *localhost* como parâmetro de IP real dos nós ativos, e fazendo a distinção entre os nós apenas pelo IP virtual e porta) e o arquivo de rotas. Também existe um *script* que inicializa a aplicação (**.start*), passando como parâmetro o nome do arquivo de configuração e de rotas a ser utilizado; indicando o IP dos nós a serem monitorados e executando o *nodesetup.sh*.

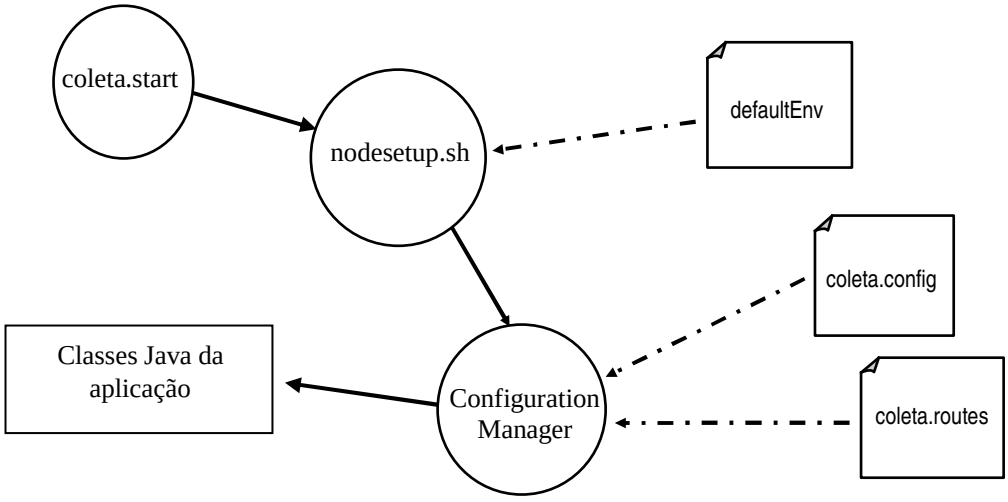

Figura 20. Estrutura de *scripts* dos exemplos com simulação.

5.1.1.1. Cenário da Aplicação para os *Benchmarks*

Para a realização dos *benchmarks*, foi necessária a utilização de um cenário real que, porém, limitou-se ao uso de apenas três estações. Como dito nas seções anteriores, no ambiente de redes ativas empregado há a necessidade de que todas as máquinas envolvidas no monitoramento estejam com o nó ANTS ativo em tempo de execução. Entretanto, existe um mecanismo de distribuição de código que dispensa a presença da aplicação em todos os nós, sendo requerida apenas nos nós que iniciam o processo, ou seja, nos nós gerentes. Neste caso, foi montado o cenário para os *benchmarks* como representado pela Figura 21.

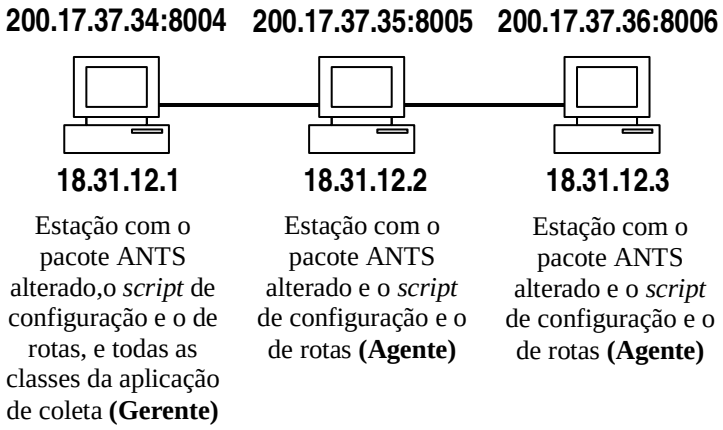

Figura 21. Topologia real da aplicação de coleta.

5.2 Protocolos de Gerência Tradicionais

Esta seção descreve o principal modelo padronizado existente para a gerência de rede, que é o modelo da Internet com o uso do protocolo SNMP. Esta abordagem é feita com o intuito de ser realizada uma análise comparativa dos modelos tradicionais com o mecanismo proposto na próxima seção.

5.2.1 O Modelo de Gerenciamento da Internet

O modelo da Internet de gerenciamento tem sua estrutura baseada no protocolo SNMP, que foi lançado em 1998 para atender à necessidade cada vez maior de um padrão para gerenciar os dispositivos IP [36]. Na sua concepção, este modelo englobou conceitos do próprio modelo OSI, tais como os da estrutura e da base de informações de gerenciamento.

Como o próprio nome sugere, o SNMP define um modelo simples para a gerência de rede de computadores. Esta simplicidade de implementação tem determinado sua dominação do mercado. Uma implementação SNMP consome poucos recursos de programação, armazenamento e processamento, o que permite a sua inclusão numa ampla faixa de equipamentos. Podemos gerenciar, via SNMP, dispositivos como impressoras, *no-breaks, switches, hubs*, entre outros. Além disto, por apresentar uma estrutura bem dirigida, facilmente se alcança a interoperabilidade entre estações de gerenciamento e sistemas de agentes de diferentes fornecedores.

No mundo do SNMP, existem dois tipos de entidade, o gerente e o agente. Um gerente pode ser definido como um servidor executando de sistema de *software* que pode lidar com tarefas de gerenciamento de uma rede.

A segunda entidade, o agente, é o *software* executado nos dispositivos da rede que são monitorados. Desta forma, estes dispositivos são capazes de responder a consultas do gerente (*polling*) e de enviar *traps*.

O relacionamento destas duas entidades baseia-se em um modelo cliente servidor, como mencionado na seção 4.2. As interações entre o gerente e o agente estão representadas na Figura 22.

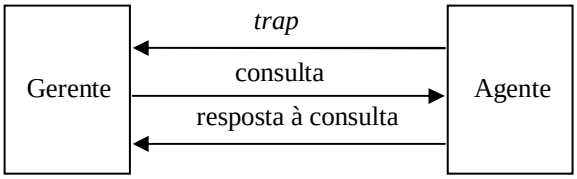

Figura 22. Relacionamento entre um gerente e um agente.

Neste contexto do modelo cliente-servidor, as coletas de dados são todas originadas a partir do gerente. Caso o escopo de abrangência do gerente tenha dimensões muito grandes, estas mensagens de consulta se apresentam em um número bastante elevado.

5.2.1.1. Operações SNMP

Assumindo que seja conhecida a forma do SNMP organizar as informações através do SMI e da MIB, será abordado como realmente poderemos obter informações de gerenciamento.

O protocolo SNMP possui uma variedade de operações, porém serão examinadas apenas as que se apresentam relevantes para o escopo do trabalho, ou seja, as referentes à consulta e coleta de informações:

- **Operação *get***

A solicitação de *get* é iniciada pelo gerente, que a envia para o agente. O agente recebe a solicitação e a analisa, possibilitando o diagnóstico do estado da rede e, conseqüentemente, a execução de uma ação corretiva, caso não haja conformidade nos dados coletados. Alguns dispositivos com carga excessiva, como roteadores, talvez não consigam responder a

solicitação e precisam excluí-la. Se o agente conseguir obter as informações solicitadas, retorna um *get-response* para o gerente. Esse processo é ilustrado na Figura 23.

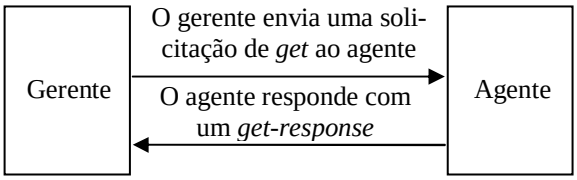

Figura 23. Seqüência de solicitações *get*.

Para que o agente saiba o que um gerente está procurando, a OID do objeto desejado na MIB do agente tem que estar especificada em um dos campos do PDU do *get*. O comando *get* serve para recuperar um único objeto da MIB de cada vez. Entretanto, esta limitação pode ser superada através da operação *get-next*.

- **Operação *get-next***

A operação *get-next* permite emitir uma seqüência de comandos para recuperar um grupo de valores de uma MIB. Em outras palavras, para cada objeto MIB a ser recuperado, são gerados separadamente uma solicitação de *get-next* e um *get-response*.

O comando *get-next* atravessa uma sub-árvore em ordem lexicográfica. Como uma OID é uma seqüência de inteiros, é fácil para um gerente iniciar na raiz da árvore de objetos da respectiva SMI e descer os galhos da árvore até encontrar a OID que está procurando. Quando o gerente receber uma resposta do agente ao comando *get-next* recém emitido, ele envia outro comando *get-next* e continua repetindo esse comando até que o agente retorne um erro, significando que o fim da MIB foi alcançado e não há mais objetos a serem retornados.

Esta operação tornou transparente ao usuário a obtenção de várias informações vinculadas aos objetos, pois, diferentemente do *get*, o usuário não precisa emitir vários comandos para obter mais de um valor de OID, basta enviar um único *get-next* que esta operação trata de obter os próximos valores de uma sub-árvore. Mesmo assim, a troca de mensagem continua sendo excessiva, pois, para se coletar *n* informações diferentes, tem que ser enviado *n get-next* pelo sistema, para que se possa receber *n get-response*.

- **Operação *get-bulk***

Uma das maiores otimizações providas pelo SNMPv2 é a operação de *get-bulk*. Sua proposta é minimizar o número de trocas requeridas para recuperar uma grande quantidade de informações de gerenciamento. O *get-bulk* permite a um gerente SNMPv2 solicitar uma resposta que seja a maior possível.

A operação *get-bulk* utiliza o mesmo princípio de seleção da operação de *get-next*, isto é, permite a recuperação de próximo objeto na ordem lexicográfica. A diferença é que, com o *get-bulk*, é possível especificar que múltiplos sucessores lexicográficos sejam selecionados.

Sua maior vantagem está em permitir que um aplicativo de gerenciamento recupere, de uma só vez, uma grande seção de uma tabela. A operação *get-bulk* transpôs uma das maiores limitações do SNMP, que era sua incapacidade de recuperar eficientemente um grande bloco de dados. Para a coleta de vários objetos de uma MIB, tanto com a operação *get* quanto com a *get-next*, o gerente necessitava enviar vários PDUs para a solicitação destes dados e recebia de volta vários *get-response*. Já o PDU do *get-bulk*, suporta uma maior quantidade de dados.

5.3 Comparação dos Mecanismos de Coleta

Tendo como motivação os problemas existentes no SNMP, o mecanismo de coleta em questão visou melhorar o desempenho de captura de informações em uma MIB, tendo basicamente dois objetivos: diminuir o número de troca de mensagens de gerência e descentralizar as coletas de informações.

Comparando esse modelo com o tradicional SNMP, chega-se às seguintes conclusões: com a utilização do SNMPv1 (SNMP versão 1), quando era desejado obter mais de um valor de OID da MIB, o gerente tinha que enviar uma operação SNMPv1 (*get*) para cada valor de OID desejado. Mesmo utilizando as versões 2 e 3 do SNMP (SNMPv2 e SNMPv3), uma operação que suporta o retorno de um maior número de OIDs (*get-bulk*), é de difícil aplicação em gerência descentralizada. Com o *framework* proposto, é possível se obter um número bem maior de valores retornados em uma única cápsula, pois nela está apenas a especificação das rotinas que são executadas localmente e, deste modo, o número de interações entre o gerente e o agente diminui, uma vez que a operação disparada pelo SNMP atua apenas no dispositivo gerenciado.

Comparações dos parâmetros relevantes para esta dissertação podem ser expressas por meio da Tabela 1 [20], onde são feitas relações entre as versões do SNMP e a coleta ativa, nos critérios de número de trocas de mensagem para a solicitação de muitos objetos, suporte a gerenciamento descentralizado e segurança.

Tabela 1. Comparação entre as versões do SNMP e coleta ativa.

	SNMPv1	SNMPv2	SNMPv3	Coleta Ativa
Número de trocas de mensagens	elevado	baixo (com *get-bulk*)	baixo (com *get-bulk*)	baixo
Suporte ao gerenciamento distribuído	não há	difícil	difícil	fácil
Segurança	baixa	baixa	normal	normal

Na análise de desempenho, ficou patente que o maior ganho, em tempo de resposta, obteve-se com a característica de distribuição do modelo proposto, pois, quando se deseja executar operações SNMP (seja operação de *get*, *get-next* ou *get-bulk*), o gerente tem que direcionar um PDU referente a esta operação a todo agente que ele deseja monitorar. Para isso, é necessário um gasto adicional de tempo, pelo fato de todos os comandos precisarem sair a partir do gerente. Na próxima subseção, será feita uma comparação mais detalhada entre gerenciamento de redes centralizado e distribuído.

Neste mecanismo de coleta descentralizada, o próprio agente pode transmitir as cápsulas de coleta, diminuindo a sobrecarga no gerente e, conseqüentemente, o tempo de resposta. Este método de distribuição está melhor exposto na seção 4.2.

5.3.1 Gerência Centralizada *versus* Gerência Distribuída

A abordagem clássica de gerenciamento (centralizada) utilizada, por exemplo, nas implementações iniciais das arquiteturas SNMP [36] e CMIP [37] é baseada em um paradigma cliente-servidor, onde este último, instalado em um elemento da rede, na forma de um agente, sofre *pollings* freqüentes para enviar dados a uma estação de gerenciamento (a NMS – *Network Management Station*), que nesse caso faz o papel de cliente. A NMS constrói então uma visão do estado da rede e gera alarmes quando problemas são detectados. Normalmente, a intervenção do operador ou administrador da rede é requerida. A NMS pode também enviar comandos aos agentes para que estes desencadeiem ações visando resolver

esses problemas. A funcionalidade dessas ações, no entanto, é limitada ou demorada. Situações de exceção podem ser configuradas nos agentes para que estes enviem, por iniciativa própria, avisos à NMS como, por exemplo, as *traps*.

Essa abordagem gera um grande tráfego na rede, que, na maioria das vezes, após ser analisado, serve apenas para deduzir-se que a situação é normal, não existem falhas. Esse grande tráfego não só pode prejudicar o desempenho da rede como também pode fazer com que o tempo decorrido entre a detecção de uma situação anormal e a intervenção apropriada para resolvê-la seja muito grande, podendo torná-la até mesmo inútil. À medida que o número de elementos gerenciados cresce, a complexidade, os requisitos de poder de computação da estação de gerenciamento e de consumo de banda da rede gerenciada também aumentam. Ainda, em redes com grande abrangência geográfica, a NMS pode estar distante dos elementos gerenciados e, conseqüentemente, os retardos na comunicação para o envio de dados e comandos podem ser excessivamente grandes.

Abordagens alternativas (distribuídas) para o gerenciamento vêm sendo pesquisadas nos últimos anos. Desde o trabalho de Yemini [38], sobre gerenciamento distribuído por delegação, até o uso de agentes móveis [17], diversas são as opções sendo pesquisadas ou já em uso comercial para a implementação da distribuição no gerenciamento.

As abordagens distribuídas mais simplificadas dividem a rede em domínios hierárquicos de gerenciamento, cada qual com sua NMS. Atividades de monitoramento e certas capacitadas de filtragem são delegadas aos próprios elementos gerenciados, que enviam apenas alarmes ou relatórios resumidos às NMS. Exemplos dessas abordagens são as funções de monitoramento de métricas e sumarização do modelo OSI de gerenciamento [35] e RMON (*Remote Monitoring*) no modelo SNMP [9, 35]. Essas abordagens, no entanto, apesar de reduzirem o tráfego na rede e o tempo entre a detecção de uma anomalia e a emissão do correspondente alarme ou ação, não oferecem flexibilidade para atualização ou troca de *softwares* de forma rápida.

Opções mais avançadas de gerenciamento distribuído utilizam paradigmas de objetos distribuídos, como CORBA (*Common Object Broker Request Architecture*) [39] e Java-RMI (*Java - Remote Method Invocation*) [40], ou da chamada remota de procedimentos (RPC - *Remote Procedure Call*)) [41]. O uso desse paradigma, embora vantajoso sob os pontos de vista de projeto e de reuso, abstrai-se dos detalhes de implementação. Exemplos como

CORBA, DCOM (*Distibuted Component Object Mode*) e RMI são muito úteis no projeto e construção de sistemas distribuídos, mas escondem os verdadeiros custos de suas implementações. O tamanho e a complexidade da infra-estrutura que tem que estar instalada na plataforma normalmente são grandes. Como resultado, o uso dos recursos da rede pode se dar de forma ineficiente, principalmente no que diz respeito ao consumo de capacidade de transmissão dos enlaces, dada a grande quantidade de dados a ser trocada entre os objetos.

Outra desvantagem dessas abordagens é não oferecer o suporte a ambientes verdadeiramente distribuídos, onde agentes (*softwares*) podem se comunicar com vizinhos para se encarregarem, em conjunto, de executar tarefas de forma distribuída e eficiente.

As pesquisas mais avançadas para a distribuição do gerenciamento usam o que é chamado de tecnologia ativa [17]. Essa tecnologia integra as pesquisas atuais em redes ativas, redes programáveis e em agentes móveis. De forma resumida, pode-se dizer que programas podem ser instalados em nós de comutação da rede que oferecem suporte a execução de programas ou em estações. Esses programas são dotados de certa capacidade de processamento, sendo capazes de agir localmente com o objetivo de resolver problemas ou consolidar informações de gerenciamento. Dessa forma, além de reduzirem o tráfego na rede e a demora para o desencadeamento de ações de gerenciamento, a atualização desses programas pode ser realizada de forma flexível e rápida. Essa abordagem não deve ser confundida com aquelas que pregam a distribuição de objetos. Com o uso de tecnologia ativa, programas inteiros, autônomos, e não apenas procedimentos ou objetos são instalados nos elementos a serem gerenciados, além de poderem migrar.

5.4 Ganhos Obtidos

Comparando os tempos gastos no modelo de coleta proposto com o modelo tradicional do SNMP, utilizando a ferramenta *ucd-snmp* [42], foi verificado que a obtenção dos valores vinculados de algumas OIDs, em apenas um nó, resulta em um maior tempo com o mecanismo proposto. Porém, à medida que foram adicionados nós ao processo de coleta e utilizada a característica de distribuição do mecanismo, inerente ao paradigma de redes ativas, o desempenho melhorou. Chegou-se a esta conclusão através dos resultados obtidos dos tempos médios, em milisegundos (*ms*), que estão apresentados na Tabela 2, onde, em um cenário, foram utilizados dois nós (gerente e agente), e em um outro cenário se utilizaram três

nós (um gerente e dois agentes). Os *benchmarks* foram feitos com e sem o roteamento distribuído e o número de OIDs coletadas foi variável.

O ganho mais significativo foi encontrado quando se modificou o número de OIDs a serem consultadas. Utilizando o cenário com três máquinas, conclui-se que o aumento do número de OIDs a serem obtidas tem mais impacto nos tempos gastos quando o ambiente é o do SNMP tradicional. Isso se explica pelo fato de que, no método proposto, o gerente envia uma única cápsula contendo todas as OIDs a serem coletadas, enquanto no SNMP tradicional são enviados vários PDUs das operações SNMP para a obtenção de dados. Com o mecanismo de coleta ativo proposto não se altera, ficando basicamente constante, devido à quantidade de envio de cápsulas ser indiferente ao número de OIDs a serem coletadas.

Tabela 2. Resultados obtidos (em ms).

Nº de OIDs	Coleta Ativa (1 agente)	Coleta Ativa (2 agentes, c/ distribuição)	Coleta Ativa (2 agentes, s/ distribuição)	SNMP (1 agente)	SNMP (2 agentes)
1	857	877	1389	100	170
2	853	862	1376	140	310
3	860	870	1375	250	460
4	855	872	1372	340	620
5	857	867	1372	360	790
6	854	864	1389	530	1010
7	855	859	1381	670	1070
8	850	865	1385	670	1240
9	852	879	1391	750	1520
10	853	867	1385	740	1650
11	855	880	1392	900	1720
12	858	874	1382	880	1960
13	860	871	1393	980	2040
14	854	880	1384	1080	2250
15	857	870	1385	1170	2400
16	854	879	1383	1280	2580
17	856	880	1386	1400	2710
18	858	883	1383	1460	2900
19	856	885	1383	1580	2940
20	876	883	1376	1580	3370

No entanto, há de se ressaltar que, de acordo com o método utilizado para a medição dos tempos (comando *time* do *linux*), ocorre um *delay* de, em média, um segundo, decorrente

do processo de ativar os nós envolvidos. Porém, a operação SNMP também possui um tempo de processamento adicional, que é em torno de meio segundo, decorrido da chamada da aplicação pelo sistema operacional. Assim, pode-se verificar que o modelo de coleta ativa é mais eficiente quando o cenário do ambiente a ser monitorado é composto por vários dispositivos e a quantidade de variáveis a serem coletadas (OIDs) for elevada. Desta forma, mesmo com tempo imposto pela ativação dos nós, a utilização deste processo é vantajosa. Independentemente do desempenho, os objetivos básicos estabelecidos pelo projeto (diminuir troca de mensagens de gerência e descentralizar as coletas) foram alcançados.

No estudo prático, verificou-se que a configuração das rotas também é um fator relevante para a obtenção de melhores tempos. Isso se explica porque, em um ambiente com muitas máquinas, não basta ter uma gerência distribuída sem avaliar em que pontos seria vantajoso fazer a descentralização. Constatou-se que no monitoramento de poucas máquinas, o sistema de coleta deve ser totalmente distribuído, ou seja, um nó monitorado deve repassar a cápsula de coleta para um outro nó a ser monitorado. Porém, à medida que o número de nós aumenta, pode-se ter um atraso na coleta, com esse modelo de descentralização. Portanto, é de grande valia o estudo e a implementação da solução ótima para o estabelecimento das rotas. Os resultados expressados nesta seção são melhor visualizados no gráfico da Figura 24.

Verificando o gráfico da Figura 24, percebe-se que, quando temos um cenário com o *framework* de redes ativas, os tempos permanecem quase constantes com a variação do número de OIDs. Neste gráfico, existem três cenários diferentes em que isto se aplica: um gerente monitorando apenas um agente; um gerente monitorando dois agentes, porém, sem aplicar a característica de distribuição do mecanismo; e um gerente monitorando dois agentes e aplicando a característica de distribuição do mecanismo.

Entre estes três cenários, verificou-se que a coleta descentralizada melhorou o tempo de resposta. Nos exemplos com três *hosts* envolvidos no monitoramento, notou-se um ganho de, em média, 500*ms* no tempo de resposta da coleta distribuída. Comparando o cenário com dois *hosts*, em que não cabe a distribuição, com o cenário de três equipamentos usando a distribuição, o aumento do tempo de resposta decorrido pelo recurso a mais a ser monitorado é quase insignificante.

Porém, as maiores diferenças nos tempos foram obtidas na comparação dos cenários que utilizavam o protocolo SNMP tradicional, com outros usando o mecanismo com redes

ativas. De acordo com o gráfico exposto, quando um gerente faz a coleta de um agente, através de operações SNMP, para a obtenção de uma quantidade pequena de OIDs, o tempo de resposta obtido é bem melhor do que quando se é aplicado o modelo ativo. À medida que esta quantidade for se aproximando de onze OIDs, este ganho foi se diluindo, conforme pode ser visto no gráfico da Figura 24, que foi gerado com a ferramenta Gnuplot [45]. A partir deste número, a coleta tradicional mostrou-se com tempos cada vez maiores.

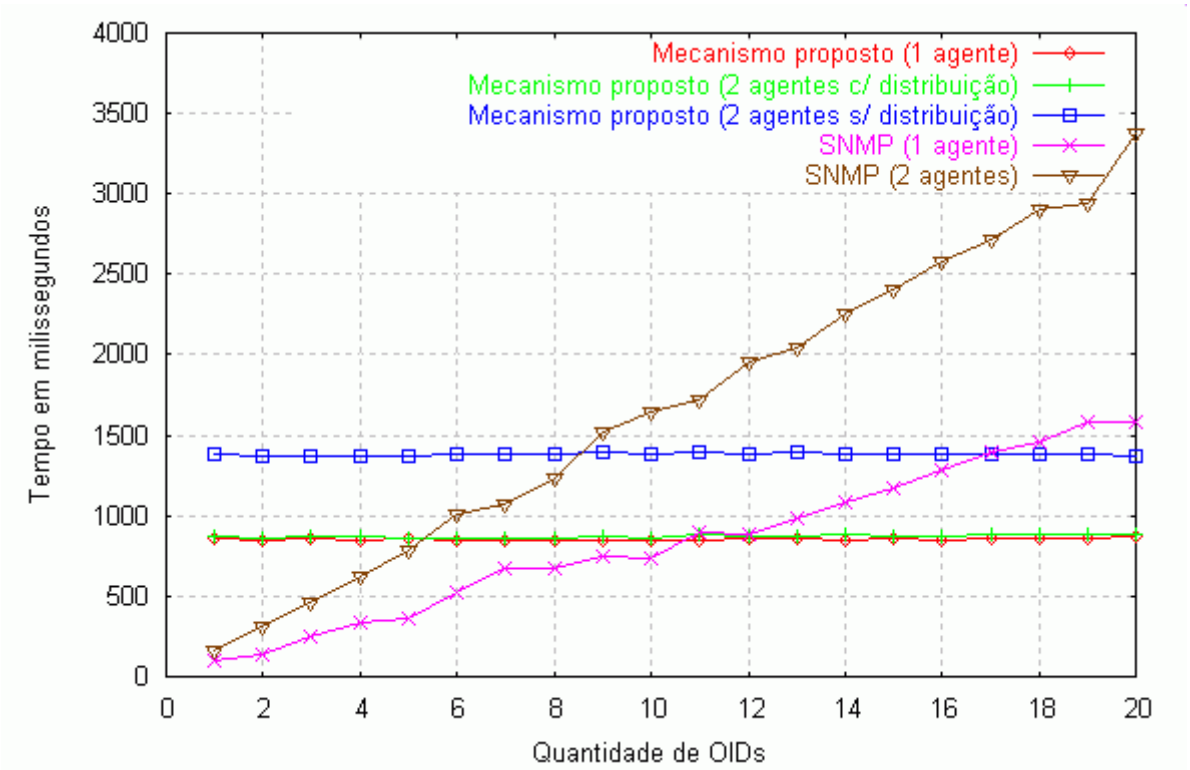

Figura 24. Benchmarks obtidos.

Analogamente a esta comparação, houve a comparação de cenários tendo um gerente e dois agentes, sendo que em um cenário foi aplicado o protocolo SNMP e no segundo, o mecanismo proposto com a característica de distribuição. Neste caso, os ganhos foram obtidos, usando o mecanismo ativo, a partir da coleta de seis OIDs, pois havia dois atenuantes para estes resultados: (1) o SNMP era sem distribuição e (2) um maior número de OIDs prejudica a coleta tradicional.

5.4.1 Levantamento Estatístico

Esta seção descreve as estatísticas dos resultados obtidos que foram apresentados na Tabela 2. A Tabela 3 traz os resultados dos cálculos de medidas que representam os dados levantados. Estas estatísticas foram obtidas com o auxílio da ferramenta Minitab [46], e, no apêndice A, encontram-se as definições das medidas utilizadas.

Tabela 3. Estatísticas dos resultados obtidos (em *ms*).

Cenário / Medidas estatísticas	Coleta Ativa (1 agente)	Coleta Ativa (2 agentes, c/ distribuição)	Coleta Ativa (2 agentes, s/ distribuição)	SNMP (1 agente)	SNMP (2 agentes)
Contagem	20	20	20	20	20
Média	856,50	873,35	1383,1	843	1686
Mediana	855,50	873,00	1383,5	815	1685
Desvio padrão	5,25	7,73	6,3	471	959
Erro Padrão	1,17	1,73	1,4	105	214
Mínimo	850	859	1372	100	170
Máximo	876	885	1393	1580	3370
Amplitude	26	26	21	1480	3200
Variância da amostra	27,53	59,71	39,35	221895,79	918815,53

Através da Tabela 3, verifica-se que as médias ($\overline{x}$) dos cenários obtidas com o método proposto usando um agente, com o método proposto com dois agentes com distribuição e com o método SNMP usando um agente, apresentam poucas diferenças, porém, quando se tem um cenário de coleta SNMP com dois agentes, a coleta proposta, sem utilizar a característica de distribuição, apresenta uma média de aproximadamente 20% menor que a média da coleta tradicional. Quando a coleta SNMP em dois agentes é comparada com o mecanismo proposto utilizando a distribuição, observa-se um diminuição da média dos tempos do mecanismo proposto ainda maior, em torno de 50%.

Um outro ponto a ser salientado é que os desvios padrões (s) do método desta dissertação são relativamente pequenos, enquanto os desvios padrões do método SNMP se mostram mais elevados comparados com o do mecanismo proposto, como pode ser observado na Tabela 3. O mesmo pode se afirmar com relação à variância da amostra. Isto é decorrente do fato de haver uma breve constância nos tempos gastos pelo método proposto, mesmo com a variação do número de OIDs. Isto é reforçado com a observação dos mínimos e máximos dos cenários utilizando este método, que resultam em uma amplitude pequena.

Em contrapartida, a coleta SNMP apresenta uma variância mais elevada que o mecanismo de coleta ativo, sendo comprovado através Tabela 3.

Com este levantamento, é possível concluir que o mecanismo proposto se mostra mais vantajoso que o método tradicional para a coleta de grande quantidade de dados, pois o número de OIDs não interfere na variação dos tempos gastos durante a coleta, ao contrário do SNMP, em que essa variação torna-se relevante.

As representações gráficas das distribuições dos dados dos cenários avaliados se encontram nas Figuras 25 a 29.

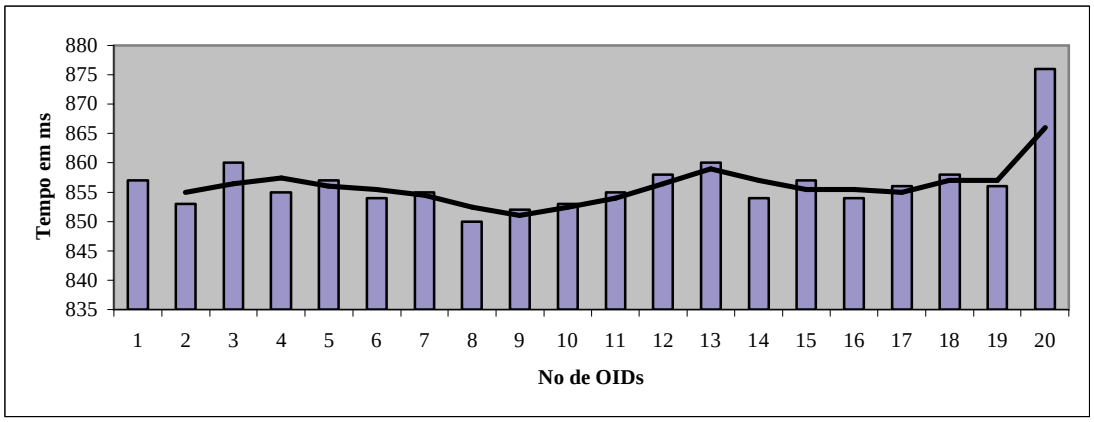

Figura 25. Coleta ativa (1 agente).

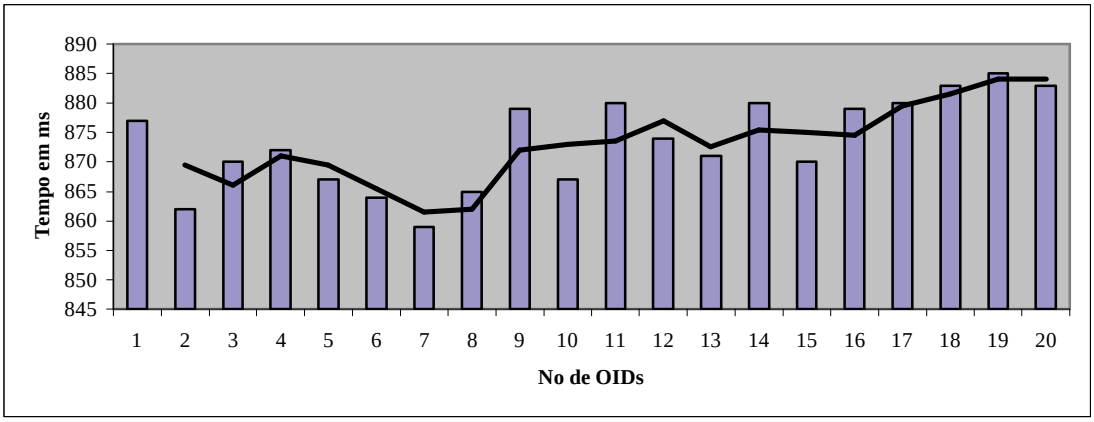

Figura 26. Coleta ativa (2 agentes com distribuição).

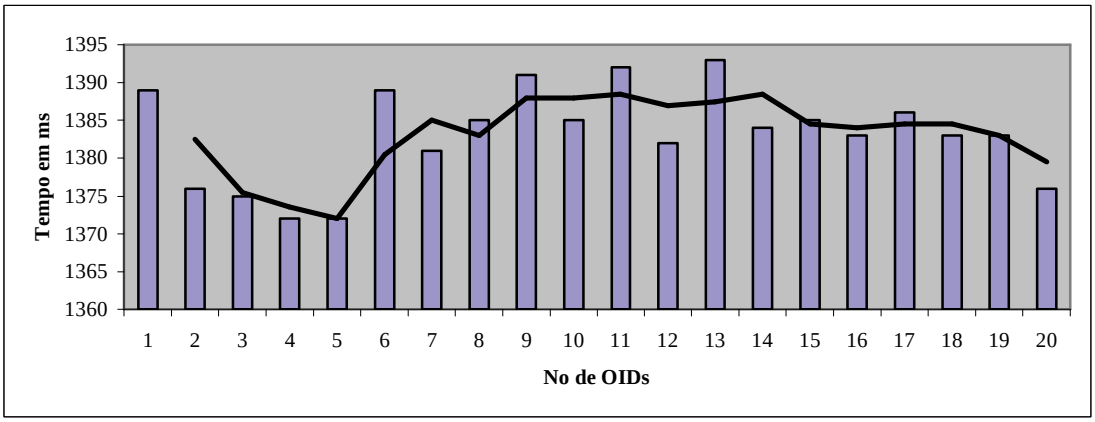

Figura 27. Coleta ativa (2 agentes sem distribuição).

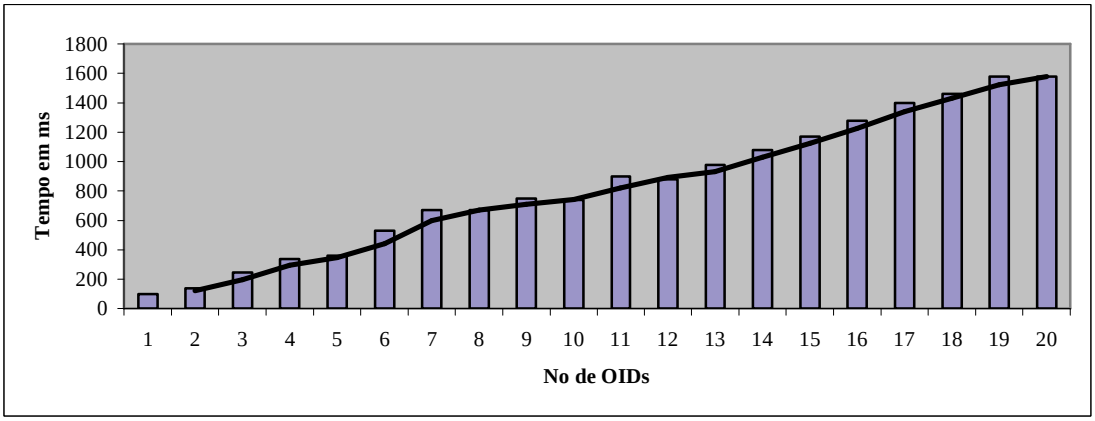

Figura 28. Coleta SNMP (1 agente).

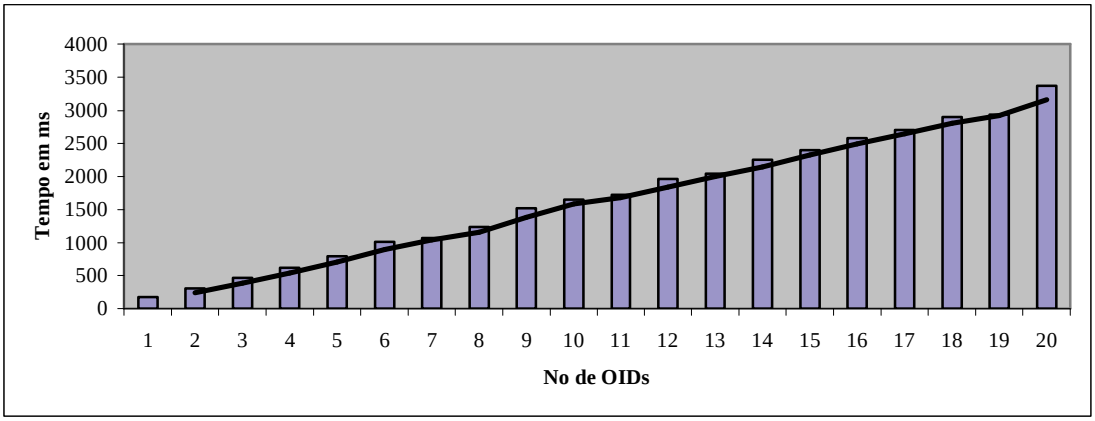

Figura 29. Coleta SNMP (2 agentes).

6 Conclusão

6.1 Considerações Sobre o Mecanismo Proposto

A motivação deste trabalho se deve às deficiências apresentadas nos protocolos de gerência tradicionais, baseados no modelo cliente-servidor, que são modelos de gerenciamento centralizados, com excessivas trocas de mensagens entre um gerente e um ou mais agentes, desperdiçando largura de banda e obtendo atrasos no monitoramento da rede.

Para a implementação deste novo modelo de coleta, foram utilizados os conceitos de redes ativas, uma vez que o gerenciamento de redes é uma das aplicações que pode se beneficiar com o uso desta tecnologia. Neste ambiente de redes ativas há mais flexibilidade, o que possibilita o gerenciamento descentralizado, diminuindo tanto o tempo de resposta quanto a utilização da banda passante.

Nos estudos feitos, foi constatado que entre as plataformas ativas mais importantes está o ANTS, que é um *kit* de ferramentas baseado em Java, desenvolvido pelo MIT para construir e implementar dinamicamente protocolos de rede. Portanto, este *framework* foi utilizado para o desenvolvimento do módulo proposto.

Os objetivos de otimizar o uso da banda passante e de diminuir os tempos gastos em uma coleta foram alcançados por se tratar de um mecanismo de coleta de dados baseado em gerência distribuída e pelo fato de ser injetado na rede apenas as cápsulas que indicam a inicialização de todo o processo de coleta local, que é através de operações SNMP locais, evitando o excesso de troca de mensagens entre o cliente e o servidor. Foram feitos *benchmarks* que comprovaram estas vantagens trazidas pela coleta de dados ativa.

Com o intuito de validar o modelo e mostrar as vantagens deste novo mecanismo de coleta, foi implementado um aplicativo, com uma interface amigável, para geração de *scripts*

de configuração. Utilizando esta aplicação, os responsáveis pelo gerenciamento da rede podem configurar suas consultas através da escolha de variáveis das MIBs, ou módulos de MIBs, e as rotas das cápsulas de coleta. As configurações dos nós da rede monitorada podem se dar também através deste aplicativo, ficando bem intuitiva a representação do monitoramento a ser realizado.

Este módulo de coleta de dados é uma contribuição para que as ferramentas de gerenciamento possam melhorar o seu desempenho através de características da tecnologia de redes ativas, como, principalmente, descentralização e flexibilidade. Desta forma, ferramentas de gerenciamento podem obter um ganho significativo de tempo em suas coletas de informações com o uso do mecanismo proposto ao invés das operações tradicionais do protocolo SNMP.

6.2 Trabalhos Futuros

Um dos trabalhos futuros desencadeado por esta dissertação será fazer uma análise de desempenho mais detalhada usando os dados levantados nos *benchmarks*.

O desenvolvimento de uma arquitetura de gerenciamento que se integre com modelo de coleta ativa pode ser apontado como um outro trabalho futuro. Salientando que, como o *framework* utilizado é baseado em redes ativas, a plataforma de gerência que utilizar o mecanismo proposto, pode usufruir de tecnologia ativa. Neste caso, a gerência em questão pode ser ainda mais eficiente.

Como houve bons resultados nos testes feitos com o protótipo, pretende-se desenvolver um gerenciamento pró-ativo utilizando o mecanismo proposto nesta dissertação. Desta forma, pode-se ter um gerenciamento descentralizado e com bons resultados na obtenção de dados.

6.3 Considerações Finais

O trabalho integrou áreas de conhecimento relativamente novas na computação e que estão sendo utilizadas em muitas pesquisas, como redes ativas e gerenciamento distribuído, com a finalidade de diminuir a troca de mensagens de gerência e descentralizar o mecanismo de coleta.

O protótipo implementado mostrou que a construção de um sistema baseado na arquitetura já pode ser considerada viável e também que, com a melhoria dos tempos de respostas das coletas tradicionais, como a do protocolo SNMP, um gerenciamento de rede mais eficaz pode ser prestado aos usuários com o uso do mecanismo.

Referências Bibliográficas

[1] ANAGNOSTAKIS, K. G., et al. Efficient Packet Monitoring for network Management. In: *IFIP/IEEE Network Operations and Management Symposium (NOMS) 2002*, p.423-436, Abr., 2002. Disponível em: <http://www.cis.upenn.edu/~anagnost/papers/flame.pdf>. Acesso em: 03 maio 2003.

[2] HOU, Y.T.; DONG, Y.; ZHANG, Z. Network Performance Measurement and Analysis – Part1: A Server-Based Measurement Infrastructure. Disponível em: <http://citeseer.nj.nec.com/249251.html>. Acesso em: 03 agosto 2002.

[3] TENNENHOUSE, D. L. A Survey of Active Network Research. *IEEE Communications Magazine*, v.35, n.1, p.80-86, Jan., 1997. Disponível em: <www.cs.brown.edu/courses/-cs295-1/Papers/active.pdf >. Acesso em: 03 maio 2003.

[4] WETHERALL, D.J.; GUTTAG, J.V.; TENNENHOUSE, D.L. ANTS: "A Toolkit for Building and Dynamically Deploying Network Protocols. In: IEEE OPENARCH'98, San Francisco, CA, Abr., 1998. Disponível em: <http://citeseer.nj.nec.com/-wetherall98ants.html>. Acesso em: 03 maio 2003.

[5] SCHWARTZ, B.; ZHOU, W.; JACKSON, A.W. Smart Packets for Active Networks. *BBN Technologies*, Jan., 1998. Disponível em: <http://citeseer.nj.nec.com/-schwartz98smart.html>. Acesso em: 03 maio 2003.

[6] WETHERALL, D.J.; TENNENHOUSE, D.L. The ACTIVE_IP option. In: THE 7TH ACM SIGOPS EUROPEAN WORKSHOP, 7, Set., 1996. Disponível em: <http://citeseer.nj.nec.com/wetherall96active.html>. Acesso em: 03 maio 2003.

[7] GUNTER, C.A.; NETTLES, S.M.; SMITH, J.M. The SwitchWare Active Network Architecture. *IEEE Networ*k, special issue on Active and Pro-grammable Networks, v.12,

n.3. , Mai/Jun, 1998. Disponível em: <http://citeseer.nj.nec.com/5813.html>. Acesso em: 03 maio 2003.

[8] FRALEIGH, C., et al. Design and Deployment of a Passive Monitoring. *Lecture Notes in Computer Science*, v.2170, p.556-567, 2001. Disponível em: <http://citeseer.nj.nec.com/-fraleigh01design.html>. Acesso em: 03 maio 2003.

[9] STALLINGS, W. **SNMP, SNMPV2, and RMON.** Addison-Wesley, 1996. 478p.

[10] PAXSON, V., et al. An Architecture for Large-Scale Internet Measurement. *IEEE Communications*, v.36, n.8, p.48-54, 1998. Disponível em: <http://citeseer.nj.nec.com/-paxson98architecture.html>. Acesso em: 03 maio 2003.

[11] **NetFlow Services and Applications**. Disponível em <http://www.cisco.com/-warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.htm>. Acesso em: 03 agosto 2002.

[12] **CoralReef website**. Disponível em: <http://www.caida.org/tools/measurement/-coralreef>. Acesso em: 03 agosto 2002.

[13] **Treno web page**. Disponível em: <http://www.psc.edu/networking/-treno_info.html>. Acesso em: 03 agosto 2002.

[14] CÁCERES, R., et al. Multicasting-based Inference of Network-internal Loss Characteristics. IEEE Transactions on Information Theory, v.45, n.7, p.2462-2480, Nov., 1999. Disponível em: <http://citeseer.nj.nec.com/485457.html>. Acesso em: 03 maio 2003.

[15] KALIDINDI, S.; ZEKAUSKAS, M.J. Surveyor: An Infrastructure for Internet Performance Measurement. In: *Proceedings of INET '99*, Jun., 1999. Disponível em: <http://www.isoc.org/inet99/proceedings/4h/4h_2.htm>. Acesso em: 03 maio 2003.

[16] McGREGOR, T.; BRAUN, H.W.; BROWN, J. The NLANR Network Analysis Infrastructure. IEEE Communications, v.38, n.5, Mai., 2000. Disponível em: <http://www.comsoc.org/ci/private/2000/may/pdf/Mcgregor.pdf>. Acesso em: 03 maio 2003.

[17] KAWAMURA, R.; STADLER, R. Active Distributed Management for IP Networks, *IEEE Communications Magazine*, Abr., 2000. Disponível em: <http://www.comsoc.org/-ci/private/2000/apr/pdf/Kawamura.pdf>. Acesso em: 03 maio 2003.

[18] BOHORIS, C.; PAVLOU, G.; CRUICKSHANK, H. Using Mobile Agents for Network Performance Management. IFIP/IEEE Network Operations and Management Symposium (NOMS'00), Hawaii, USA, IEEE, Abr. 2000. Disponível em: <http://www.ee.surrey.ac.uk/Personal/C.Bohoris/doc/boho-1-00.pdf>. Acesso em: 03 maio 2003.

[19] BIESZEZAD, A.; PAGUREK, B.; WHITE, T. Mobile Agents for Network Management, *IEEE Communication Survey*, v.1, n.1, 4th Quarter, 1998. Disponível em: <http://citeseer.nj.nec.com/bieszczad98mobile.html>. Acesso em: 03 maio 2003.

[20] BIN, X., et al. An Active Network-Based Network Management Framework. International Conference on Communication Technologies (ICCT2000), Beijing, China, Aug., 2000. Disponível em: <http://www.ifip.or.at/con2000/icct2000/icct464.pdf>. Acesso em: 03 maio 2003.

[21] TENNENHOUSE, D. L.; WETHERALL D.J. Towards an Active Network Architecture. *Computer Communication Review*, v.26, n.2, p.5-18, Apr., 1996. Disponível em: <http://citeseer.nj.nec.com/tennenhouse96towards.html>. Acesso em: 03 maio 2003.

[22] PSOUNIS, K. Active Networks: Applications, Security, Safety, and Architectures. *IEEE Communications Magazine*, p.1-16, 1st quarter, 1999. Disponível em: <http://citeseer.nj.nec.com/psounis99active.html>. Acesso em: 03 maio 2003.

[23] TENNENHOUSE, D. L. From Internet to ActiveNet. *Request for Comments*, Jan. 1996. Disponível em: <http://citeseer.nj.nec.com/tennenhouse96from.html>. Acesso em: 03 maio 2003.

[24] **Smart Packets**. Disponível em: <http://www.din.uem.br/~ra/artigos/Aplicacoes%-20Redes%20Ativas.html>. Acesso em: 07 maio 2003.

[25] BHATTACHARJEE, S.; CALVERT, K.L.; ZEGURA, E.W. On Active Networking and Congestion. *Technical Report GIT-CC-96/02*, College of Computing, Georgia Institute of Technology, Atlanta GA, 1996. Disponível em: <http://citeseer.nj.nec.com/-bhattacharjee96active.html>. Acesso em: 03 maio 2003.

[26] **CANEs**. Disponível em: http://www.cc.gatech.edu/projects/canes/>. Acesso em: 07 maio 2003.

[27] **The SwitchWare Project**. Disponível em: <http://www.cis.upenn.edu/~switchware/>. Acesso em: 07 maio 2003.

[28] **NetScript**. Disponível em: <http://www1.cs.columbia.edu/dcc/netscript/>. Acesso em: 07 maio 2003.

[29] WETHERALL, D.J. Developing Network Protocols with the ANTS Toolkit. *Laboratory for Computer Science*, MIT, Design Review, Ago., 1997. Disponível em: <http://www.cs.washington.edu/research/networking/ants/papers/programming.ps>. Acesso em: 03 maio 2003.

[30] **The Janos Project**. Disponível em: <http://www.cs.utah.edu/flux/janos/>. Acesso em: 10 março 2003.

[31] WETHERALL, D.J., **Service Introduction in an Active Network**, Ph.D. Thesis, disponível as MIT/LCS/TR-773, Fev., 1999.

[32] ACHIR, N., et al. Active Networking Systems Evaluation : A Practical Experience. MoMuC'2000, Tokyo, Out., 2000. Disponível em: <http://www-rp.lip6.fr/~fonseca/-Publication/momuc2000/momuc2000.pdf>. Acesso em: 03 maio 2003.

[33] **The Janos Project: ANTS EE**. Disponível em: <http://www.cs.utah.edu/flux/janos/-ants.-html>. Acesso em: 10 março 2003.

[34] **Java SNMP Package**. Disponível em: <http://edge.mcs.drexel.edu/GICL/people/sevy/-snmp/snmp.html.> Acesso em 10 março 2003.

[35] SUBRAMANIAN, M. **Network Management – Principles and Practice.** Addison-Wesley, 2000. 644p.

[36] MAURO, D.; SCHMIDT, E.K. **Essential SNMP.** Campus, 2001. 316p.

[37] RUBINSTEIN, M.G.; DUARTE, O.C.M.; PUJOLLE, G. Evaluating the Performance of Mobile Agents in Network Management. IEEE Global Telecommunications Conference - Globecom'99, pp. 386-390, Rio de Janeiro, RJ, Brazil, December 1999. Disponível em <http://www.gta.ufrj.br/ftp/gta/TechReports/RuDu99a.ps.gz>. Acesso em: 03 maio 2003.

[38] GOLDSZMIDT, G.; YEMINI, Y. Distributed Management by Delegation. 15a conferência internacional sobre sistemas de computação distribuídos, 1995. Disponível em: <http://www.cs.columbia.edu/~german/papers/icdcs95.ps>. Acesso em: 03 maio 2003.

[39] OBJECT MANAGEMENT GROUP. The Common Object Request Broker: Architecture and Specification (CORBA), Versão 2.0, 1995. Disponível em: <http://www.omg.org/docs/ptc/96-03-04.pdf>. Acesso em: 03 maio 2003.

[40] DEITEL, H.M.; DEITEL, P. J. **Java, How to Program**. Ed. Prentice Hall, quarta edição, 2001.

[41] STEVENS, W.R. **Unix Networking Programming**, Ed. Prentice Hall, segunda edição, 1998.

[42] **Net-SNMP website**. Disponível em: <http://www.net-snmp.org>. Acesso em: 20 outubro 2002.

[43] CECILIO, E.L. **Uma Arquitetura de Gerenciamento de Desempenho Pró-ativo Distribuído Usando Tecnologia Ativa**. 2002. 106f. Dissertação - Instituto de Matemática / Núcleo de Computação Eletrônica, Universidade Federal do Rio de Janeiro, Rio de Janeiro. Disponível em: <http://www.cecilio.ime.eb.br/mestrado/Dissertacao-Final.pdf>. Acesso em: 03 maio 2003.

[44] MARTINS, G. A.; FONSECA, J. S. **Curso de Estatística**. Ed. Atlas, sexta edição, 1998, 320p.

[45] **Gnuplot Central**. Disponível em: <http://www.gnuplot.info>. Acesso em: 23 julho de 2003.

[47] **MINITAB: The Standart for Six Sigma Success**. Disponível em: <http://www.minitab.com/ >. Acesso em: 23 julho de 2003.

A Apêndice

A.1 Descrição das Medidas Estatísticas Utilizadas nesta Dissertação:

Contagem	Número de dados observados em cada cenário (n).
Média	Cálculo da média aritmética.
Mediana	A mediana está na metade dos dados. Metade dos valores são iguais ou menores que ela. Se o total de valores (n) for ímpar, a mediana é o valor q está na metade. Se n for par, a mediana será a média dos dois valores medianos.
Desvio padrão	Informa o quanto os dados são distintos. Sua fórmula é: $\sqrt{\frac{\sum (x+\bar{x})^2}{n-1}}$
Erro Padrão	Dá o erro padrão da amostra. É calculado através do (desvio padrão) / $\sqrt{n}$.
Mínimo	O menor valor da amostra.
Máximo	O maior valor da amostra.
Amplitude	É a diferença entre o máximo e o mínimo da amostra.
Variância da amostra	A variância é uma medição que reflete o quanto a média se difere dos valores da amostra. Sua fórmula é $\frac{\sum (x+\bar{x})}{n-1}$.

B Apêndice

A.1 ColetaProtocol

```
/*
 * ColetaProtocol.Java
 *
 */

package apps.coleta;

/*
 * Definição do protocolo de coleta
 *
 */

public class ColetaProtocol
    extends ants.core.Protocol
{
    public ColetaProtocol()
   throws Exception
    {
   /*
    * Agrupa os tipos de cápsula relacionadas em um grupo de código
    *
    */
   startProtocolDefn();
   startGroupDefn();
   addCapsule("apps.coleta" + ".ColetaCapsule");
   addCapsule("apps.coleta" + ".ColetaReturnCapsule");
   addCapsule("apps.coleta" + ".ColetaCapsuleCacheKey");
   addCapsule("apps.coleta" + ".ColetaCapsuleCacheValue");
   endGroupDefn();
   endProtocolDefn();
    }
}
```

A.2 ColetaApplication

```
/*
 * ColetaApplication.Java
 *
```

```
 */

package apps.coleta;

import ants.core.Application;
import ants.core.Capsule;
import ants.core.Protocol;
import ants.core.Node;
import ants.core.NodeAddress;
import ants.core.Xdr;
import ants.core.ByteArray;
import ants.core.NeighborTable;

import ants.util.KeyArgs;
import ants.util.InvalidKeyArgsException;

import java.util.Enumeration;
import java.util.Vector;

import ants.snmp.*;
import java.net.InetAddress;

public class ColetaApplication
    extends Application
    implements Runnable
{
    private final static String[] defaults = { "-depth", "15",
                      "-maxwait", "30",
                      "-senddelay", "500"};

    /*
     * A profundidade máxima que este pedido entrará na rede antes de ser
     * cancelar (similar ao TTL)
     */
    private byte depth;

    /*
     * Os oids a serem coletados são passados em um único argumento
     * separados por vígulas
     */
    private String oids;

    /*
     * Segundos máximo de espera pela atividade antes de terminar a
aplicação
     *
     */
    private int maxwait;

    /*
     * Milisegundos de espera entre o envio do pacote da solicitação do
gerente
     * (diminui a tensão da rede)
     */
    private int rootsenddelay;

    /*
     * Mantem-se a par do número de pacotes retornados
     *
```

```
     */
    private int retcount;

    private final int LOG_LEVEL = L[6];

    private class SrcDestPair
    {
   public int src,dest;

   public SrcDestPair(int src, int dest)
   {
       this.src=src;
       this.dest=dest;
   }

   public boolean equals(Object o)
   {
       return (((SrcDestPair)o).src==src&&
          ((SrcDestPair)o).dest==dest);
   }
    }

    /*
     * Mantem-se a par dos nós que emitiram os pedidos a que não foram
     * respondidos
     */
    private Vector waitingon;

    /*
     * Mantem-se a par dos pais dos nós que tiveram retorno, mas que a
     * aplicação não estava esperando.
     */
    private Vector unmatchedrets;

    /*
     * Controla as cápsulas que entram e imprime suas informações
     *
     */
    synchronized public void receive(Capsule unknownCap)
    {
   int i,j;
   SrcDestPair sdp;
   super.receive(unknownCap);

   if (!(unknownCap instanceof ColetaReturnCapsule))
   {
       thisNode().log(LOG_LEVEL, "Cásupa de coleta não
entregue!?!"+unknownCap);
       thisNode().log(LOG_LEVEL, "ColetaCap.classLoader=
"+ColetaCapsule.class.getClassLoader()+";
outra="+unknownCap.getClass().getClassLoader());
       return;
   }

   ColetaReturnCapsule cap = (ColetaReturnCapsule)unknownCap;

   cap.showInformation(thisNode());
   retcount++;
   // Mantenha-se a par do que se espera de retorno
```

```
for (i=0;i<cap.nodessentto;i++)
{
    sdp=new SrcDestPair(cap.infosource,
         ((Integer)cap.branches.elementAt(i)).intValue());
    if (unmatchedrets.contains(sdp))
    {
   unmatchedrets.removeElement(sdp);
    } else {
   waitingon.addElement(sdp);
    }
}
sdp=new SrcDestPair(cap.parentnode,cap.infosource);
if (waitingon.contains(sdp))
{
    waitingon.removeElement(sdp);
} else {
    unmatchedrets.addElement(sdp);
}
synchronized (waitingon)
{
    waitingon.notify();
}
 }

 /*
  * Envio do gerente das cápsule de solicitação de coleta a todos
  * os seus nós vizinhos e entrada do gerente em modo de espera
  */
 public void run()
 {
int lastcount;
long lasttime;
boolean done;
int i;

long start = System.currentTimeMillis();

waitingon=new Vector();
unmatchedrets=new Vector();
thisNode().log(LOG_LEVEL, "*** Enviando cápsulas de coleta");
sendRootCapsules();

try
{
    lastcount=0;
    lasttime=thisNode().time();
    done=false;
    while (!done&&
      ((waitingon.size()!=0||
        unmatchedrets.size()!=0)||
       retcount==0))
    {
   lastcount=retcount;
   synchronized (waitingon)
   {
       waitingon.wait(maxwait*1000);
   }
   if (lastcount==retcount&&thisNode().time()-lasttime>=maxwait*1000)
       done=true;
```

```
       }
       if (waitingon.size()==0&&unmatchedrets.size()==0)
       {
      long stop = System.currentTimeMillis()-start;
      thisNode().log(LOG_LEVEL,"*** Coleta finalizada com sucesso em
"+stop+"ms.");
       } else if (done) {
      thisNode().log(LOG_LEVEL,"*** Timeout: Aplicação de coleta
finalizada");
      for (i=0;i<waitingon.size();i++)
      {
          thisNode().log(LOG_LEVEL,"*** Coleta:
"+NodeAddress.toString(((SrcDestPair)waitingon.elementAt(i)).dest)+" para
"+NodeAddress.toString(((SrcDestPair)waitingon.elementAt(i)).src)+" não
responde");
      }
       }
   } catch (InterruptedException _) {
   }
   thisNode().shutdown();
    }

    /*
     * Envio do gerente das cápsule de solicitação de coleta a todos
     * os seus nós vizinhos
     */
    private void sendRootCapsules()
    {
   NeighborTable neighbortable;
   Enumeration neighbors;
   int target;
   long keyid;
   String oid[]=new String[20];
   int numoids=0;
   // Define os oids como valor zero de oid
   String oid1="zero";
   String oid2="zero";
   String oid3="zero";
   String oid4="zero";
   String oid5="zero";
   String oid6="zero";
   String oid7="zero";
   String oid8="zero";
   String oid9="zero";
   String oid10="zero";
   String oid11="zero";
   String oid12="zero";
   String oid13="zero";
   String oid14="zero";
   String oid15="zero";
   String oid16="zero";
   String oid17="zero";
   String oid18="zero";
   String oid19="zero";
   String oid20="zero";

   // Decomposição da String de oids em oids separados
   int i=0;
   String token = oids;
```

```
   while ((token.length()>0)&&(i<20))
   {
       int posicao = token.indexOf(",");
       if (posicao!=-1) {
           oid[i] = token.substring(0,posicao);
      token = token.substring(posicao+1,token.length());
       }
       else {
           oid[i] = token;
      token = "";
       }
       i++;
       if ((i==20)&&(token.length()>0)) thisNode().log(1,"    ATENÇÃO:
Coleta apenas de 20 oids");
   }
   numoids=i;
   if (oid[0]!=null) oid1=oid[0];
   if (oid[1]!=null) oid2=oid[1];
   if (oid[2]!=null) oid3=oid[2];
   if (oid[3]!=null) oid4=oid[3];
   if (oid[4]!=null) oid5=oid[4];
   if (oid[5]!=null) oid6=oid[5];
   if (oid[6]!=null) oid7=oid[6];
   if (oid[7]!=null) oid8=oid[7];
   if (oid[8]!=null) oid9=oid[8];
   if (oid[9]!=null) oid10=oid[9];
   if (oid[10]!=null) oid11=oid[10];
   if (oid[11]!=null) oid12=oid[11];
   if (oid[12]!=null) oid13=oid[12];
   if (oid[13]!=null) oid14=oid[13];
   if (oid[14]!=null) oid15=oid[14];
   if (oid[15]!=null) oid16=oid[15];
   if (oid[16]!=null) oid17=oid[16];
   if (oid[17]!=null) oid18=oid[17];
   if (oid[18]!=null) oid19=oid[18];
   if (oid[19]!=null) oid20=oid[19];

   neighbortable = thisNode().getNeighborTable();
   neighbors=neighbortable.getNeighbors();
   keyid=thisNode().time();
   while (neighbors.hasMoreElements())
   {
       target=((Integer)(neighbors.nextElement())).intValue();
       thisNode().log(LOG_LEVEL,
            "*** Enviando o pedido de coleta para
"+NodeAddress.toString(target));
       ColetaCapsule c = new ColetaCapsule(target);
       // Composição dos campos da cápsula
       c.rootaddress=thisNode().getAddress();
       c.numoids=numoids;
       c.oid1=oid1;
       c.oid2=oid2;
       c.oid3=oid3;
       c.oid4=oid4;
       c.oid5=oid5;
       c.oid6=oid6;
       c.oid7=oid7;
       c.oid8=oid8;
       c.oid9=oid9;
```

```
    c.oid10=oid10;
    c.oid11=oid11;
    c.oid12=oid12;
    c.oid13=oid13;
    c.oid14=oid14;
    c.oid15=oid15;
    c.oid16=oid16;
    c.oid17=oid17;
    c.oid18=oid18;
    c.oid19=oid19;
    c.oid20=oid20;
    c.keyid=keyid;
    c.dpt=getPort();
    c.depth=(byte)(depth+1);
    c.now=thisNode().time();
    send(c);
    // Adiciona o destino do retorno
    waitingon.addElement(new SrcDestPair(
                thisNode().getAddress(),target));
    try
    {
   thisNode().sleep(rootsenddelay);
    } catch (InterruptedException _) {
    }
}
 }

 /*
  * Trata os argumentos na linha de comando
  *
  */
 public void setArgs(KeyArgs k)
throws InvalidKeyArgsException
 {
int i;
k.merge(defaults);

try
{
    for (i=0;i<k.length();i++)
    {
   if (k.key(i).equals("-depth"))
   {
       depth=Byte.parseByte(k.arg(i));
       k.strike(i);
   }
   if (k.key(i).equals("-maxwait"))
   {
       maxwait=Integer.parseInt(k.arg(i));
       k.strike(i);
   }
   if (k.key(i).equals("-senddelay"))
   {
       rootsenddelay=Integer.parseInt(k.arg(i));
       k.strike(i);
   }
   if (k.key(i).equals("-oids"))
   {
       oids=k.arg(i);
```

```
            k.strike(i);
         }
          }
    } catch (Exception e) {
        throw new InvalidKeyArgsException(e.getMessage());
    }
    super.setArgs(k);
     }

     /*
      * Registra o protoloco e start a thread
      *
      */
     public void start()
    throws Exception
     {
    thisNode().register(new ColetaProtocol());
    thisNode().threadStart(this);
     }

     /*
      * Construtor da aplicação
      *
      */
     public ColetaApplication()
    throws Exception
     {
     }
}
```

A.3 ColetaCapsule

```
/*
 * ColetaCapsule.Java
 *
 */

package apps.coleta;

import ants.core.Protocol;
import ants.core.Capsule;
import ants.core.Xdr;
import ants.core.Node;
import ants.core.ByteArray;
import ants.core.NodeAddress;
import ants.core.NodeCache;

import ants.core.RouteTable;
import edu.utah.janos.route.RouteEntry;

import java.util.Enumeration;
import java.util.Vector;

import ants.snmp.*;
import java.net.InetAddress;
import java.lang.*;
import java.io.*;
```

```
public class ColetaCapsule
    extends Capsule
{
    final private static byte[] MID = findMID("apps.coleta" +
".ColetaCapsule");
    final private static byte[] PID = findPID("apps.coleta" +
".ColetaCapsule");

    /*
     * Define os campos da cápsula
     *
     */
    public int rootaddress;
    public int numoids;
    public String oid1;
    public String oid2;
    public String oid3;
    public String oid4;
    public String oid5;
    public String oid6;
    public String oid7;
    public String oid8;
    public String oid9;
    public String oid10;
    public String oid11;
    public String oid12;
    public String oid13;
    public String oid14;
    public String oid15;
    public String oid16;
    public String oid17;
    public String oid18;
    public String oid19;
    public String oid20;
    public long keyid;
    public short dpt;
    public byte depth;
    public long now;
    private final static int cachetime=180;

    public Vector branches;

    protected byte[] mid()
    {
   return MID;
    }

    protected byte[] pid()
    {
   return PID;
    }

    public int length()
    {
   return
super.length()+Xdr.INT+Xdr.LONG+Xdr.SHORT+Xdr.BYTE+Xdr.LONG+Xdr.INT+Xdr.STR
ING(oid1)+Xdr.STRING(oid2)+Xdr.STRING(oid3)+Xdr.STRING(oid4)+Xdr.STRING(oid
5)+Xdr.STRING(oid6)+Xdr.STRING(oid7)+Xdr.STRING(oid8)+Xdr.STRING(oid9)+Xdr.
```

```
STRING(oid10)+Xdr.STRING(oid11)+Xdr.STRING(oid12)+Xdr.STRING(oid13)+Xdr.STR
ING(oid14)+Xdr.STRING(oid15)+Xdr.STRING(oid16)+Xdr.STRING(oid17)+Xdr.STRING
(oid18)+Xdr.STRING(oid19)+Xdr.STRING(oid20); //+Xdr.STRING(oids);
    }

    /*
     * Monta os campos da cápsula de pedido de coleta
     *
     */
    public Xdr encode()
    {
   Xdr xdr = super.encode();
   xdr.PUT(rootaddress);
   xdr.PUT(numoids);
   xdr.PUT(oid1);
   xdr.PUT(oid2);
   xdr.PUT(oid3);
   xdr.PUT(oid4);
   xdr.PUT(oid5);
   xdr.PUT(oid6);
   xdr.PUT(oid7);
   xdr.PUT(oid8);
   xdr.PUT(oid9);
   xdr.PUT(oid10);
   xdr.PUT(oid11);
   xdr.PUT(oid12);
   xdr.PUT(oid13);
   xdr.PUT(oid14);
   xdr.PUT(oid15);
   xdr.PUT(oid16);
   xdr.PUT(oid17);
   xdr.PUT(oid18);
   xdr.PUT(oid19);
   xdr.PUT(oid20);
   xdr.PUT(keyid);
   xdr.PUT(dpt);
   xdr.PUT((byte)(depth-1));
   xdr.PUT(now);
   return xdr;
    }

    public Xdr decode()
    {
   Xdr xdr = super.decode();
   rootaddress=xdr.INT();
   numoids=xdr.INT();
   oid1=xdr.STRING();
   oid2=xdr.STRING();
   oid3=xdr.STRING();
   oid4=xdr.STRING();
   oid5=xdr.STRING();
   oid6=xdr.STRING();
   oid7=xdr.STRING();
   oid8=xdr.STRING();
   oid9=xdr.STRING();
   oid10=xdr.STRING();
   oid11=xdr.STRING();
   oid12=xdr.STRING();
   oid13=xdr.STRING();
```

```
oid14=xdr.STRING();
oid15=xdr.STRING();
oid16=xdr.STRING();
oid17=xdr.STRING();
oid18=xdr.STRING();
oid19=xdr.STRING();
oid20=xdr.STRING();
keyid=xdr.LONG();
dpt=xdr.SHORT();
depth=xdr.BYTE();
now=xdr.LONG();
return xdr;
 }

 /*
  * Executa o controle de cache para verificar se
  * o nó vizinho já foi vizitado
  */
 private int branch(Node n)
 {
int numbranches;
Enumeration neighbors;
int target;
ColetaCapsuleCacheKey skey;
ColetaCapsuleCacheValue sval;
NodeCache cache;

branches=new Vector();
neighbors=n.getNeighborTable().getNeighbors();
cache=n.getCache();
skey=new ColetaCapsuleCacheKey(keyid,rootaddress,getPrevious());
sval=new ColetaCapsuleCacheValue();
sval.time=n.time();
cache.put(skey,sval,cachetime);
skey=new ColetaCapsuleCacheKey(keyid,rootaddress,rootaddress);
sval=new ColetaCapsuleCacheValue();
sval.time=n.time();
cache.put(skey,sval,cachetime);
numbranches=0;
while (neighbors.hasMoreElements())
{
    target=((Integer)neighbors.nextElement()).intValue();
    synchronized (cache)
    {
   n.log(1,"*** Checando "+NodeAddress.toString(target));
   skey=new ColetaCapsuleCacheKey(keyid,rootaddress,target);
   sval=(ColetaCapsuleCacheValue)cache.get(skey);
   if (sval!=null)
       continue;  // Já havia sido vizitado
   else
   {
       sval=new ColetaCapsuleCacheValue();
       sval.time=n.time();
       cache.put(skey,sval,cachetime);
   }
    }
    n.log(1,"*** Enviando pedido de coleta para "+
     NodeAddress.toString(target));
```

```
    /*
     * Envio de cápsulas de coleta ao vizinho de um
     * nó que recebeu a cápsula de coleta (agente)
     */
    ColetaCapsule c=new ColetaCapsule(target);
    c.rootaddress=rootaddress;
    c.numoids=numoids;
    c.oid1=oid1;
    c.oid2=oid2;
    c.oid3=oid3;
    c.oid4=oid4;
    c.oid5=oid5;
    c.oid6=oid6;
    c.oid7=oid7;
    c.oid8=oid8;
    c.oid9=oid9;
    c.oid10=oid10;
    c.oid11=oid11;
    c.oid12=oid12;
    c.oid13=oid13;
    c.oid14=oid14;
    c.oid15=oid15;
    c.oid16=oid16;
    c.oid17=oid17;
    c.oid18=oid18;
    c.oid19=oid19;
    c.oid20=oid20;
    c.keyid=keyid;
    c.dpt=dpt;
    c.depth=(byte)depth;
    c.now=n.time();
    c.prime(this);
    n.routeForNode(c,target);
    numbranches++;
    branches.addElement(new Integer(target));
}
return numbranches;
 }

 /*
  * Encode information about this node to be returned to the root
  * of the stats
  * <p>
  * To change the statistics returned by the tree, change this function
  */
 private ByteArray encodeNodeInfo(Node n)
 {
String addr = "<Unknown>";
String proto = "<Unknown>";
try
{
    addr = n.getVisibleAddressSpec();
    proto = n.getVisibleProtocolSpec();
}
catch (Throwable securityException)
{
}
ByteArray buf = new ByteArray(Xdr.INT+
               Xdr.STRING("Hostname")+Xdr.STRING(n.hostname())+
```

```
                Xdr.STRING("Endereco")+Xdr.STRING(addr)+
                Xdr.STRING("Protocolo")+Xdr.STRING(proto));
   Xdr xdr = new Xdr(buf,0);
   xdr.PUT(3);
   xdr.PUT("Hostname");
   xdr.PUT(n.hostname());
   xdr.PUT("Endereco");
   xdr.PUT(addr);
   xdr.PUT("Protocolo");
   xdr.PUT(proto);
   return buf;
    }

    public boolean evaluate(Node n)
    {
   int numbranches;

   int i=0;
   int count=0;
        String value1="zero";
        String value2="zero";
        String value3="zero";
        String value4="zero";
        String value5="zero";
        String value6="zero";
        String value7="zero";
        String value8="zero";
        String value9="zero";
        String value10="zero";
        String value11="zero";
        String value12="zero";
        String value13="zero";
        String value14="zero";
        String value15="zero";
        String value16="zero";
        String value17="zero";
        String value18="zero";
        String value19="zero";
        String value20="zero";
   String oid[]=new String[20];
   if (n.getAddress()!=getSrc())
   {
       ColetaReturnCapsule retCap = new
ColetaReturnCapsule(dpt,dpt,getSrc(),encodeNodeInfo(n));
       n.log(1,"*** ColetaCapsule chega do
"+NodeAddress.toString(getPrevious()));
       if (depth>0)
       {
      numbranches=branch(n);
       } else {
      numbranches=0;
       }

       // inicio da coleta SNMP local

       oid[0]=oid1;
       oid[1]=oid2;
       oid[2]=oid3;
       oid[3]=oid4;
```

```
        oid[4]=oid5;
        oid[5]=oid6;
        oid[6]=oid7;
        oid[7]=oid8;
        oid[8]=oid9;
        oid[9]=oid10;
        oid[10]=oid11;
        oid[11]=oid12;
        oid[12]=oid13;
        oid[13]=oid14;
        oid[14]=oid15;
        oid[15]=oid16;
        oid[16]=oid17;
        oid[17]=oid18;
        oid[18]=oid19;
        oid[19]=oid20;

        for (i=0;i<numoids;i++)
        {
       try
       {
          InetAddress hostAddress = InetAddress.getByName("localhost");
          String community = "public";
          int version = 0;  // SNMPv1
          SNMPv1CommunicationInterface comInterface = new
SNMPv1CommunicationInterface(version, hostAddress, community);
          String itemID = oid[i];
          n.log(1,"*** Obtendo o valor correspondente ao OID " + itemID);
          SNMPVarBindList newVars = comInterface.getMIBEntry(itemID);
          SNMPSequence pair = (SNMPSequence)(newVars.getSNMPObjectAt(0));
          SNMPObjectIdentifier snmpOID =
(SNMPObjectIdentifier)pair.getSNMPObjectAt(0);
          SNMPObject snmpValue = pair.getSNMPObjectAt(1);
          if (!snmpValue.toString().equals("")) oid[i] =
snmpValue.toString();
          else oid[i] = "*** sem valor ***";
       }
       catch(Exception e)
       {
          oid[i] = "*** erro ***";
       }
        }

        value1=oid[0];
        value2=oid[1];
        value3=oid[2];
        value4=oid[3];
        value5=oid[4];
        value6=oid[5];
        value7=oid[6];
        value8=oid[7];
        value9=oid[8];
        value10=oid[9];
        value11=oid[10];
        value12=oid[11];
        value13=oid[12];
        value14=oid[13];
        value15=oid[14];
        value16=oid[15];
```

```
        value17=oid[16];
        value18=oid[17];
        value19=oid[18];
        value20=oid[19];

             n.log(1,"*** Retornando a cásula de coleta para " +
NodeAddress.toString(getSrc()));
        // Retorno da coleta
        retCap.infosource=n.getAddress();
        retCap.parentnode=getPrevious();
        retCap.latency= 0;
        retCap.capTime=n.time();
        retCap.nodessentto=numbranches;
        retCap.branches=branches;
             retCap.value1 = value1;
             retCap.value2 = value2;
             retCap.value3 = value3;
             retCap.value4 = value4;
             retCap.value5 = value5;
             retCap.value6 = value6;
             retCap.value7 = value7;
             retCap.value8 = value8;
             retCap.value9 = value9;
             retCap.value10 = value10;
             retCap.value11 = value11;
             retCap.value12 = value12;
             retCap.value13 = value13;
             retCap.value14 = value14;
             retCap.value15 = value15;
             retCap.value16 = value16;
             retCap.value17 = value17;
             retCap.value18 = value18;
             retCap.value19 = value19;
             retCap.value20 = value20;
        retCap.prime(this);

        if (!n.routeForNode(retCap,getSrc()))
       return false;
        if (depth<=0)
        {
       n.log(1,"Expirou o número de nós a serem percorridos");
        }
        return true;
   } else if (n.getAddress()!=getDst()) {
        n.log(1,"*** Roteando o pedido de coleta de "+
         NodeAddress.toString(getDst())+
         " para "+
         NodeAddress.toString(rootaddress));

        return n.routeForNode(this,getDst());
   }
   return false;
    }

    public ColetaCapsule()
    {
    }

    public ColetaCapsule(int d)
```

```
    {
   setDst(d);
    }
}
```

A.4 ColetaReturnCapsule

```
/*
 * ColetaReturnCapsule.Java
 *
 */

package apps.coleta;

import ants.core.Protocol;
import ants.core.DataCapsule;
import ants.core.Xdr;
import ants.core.Node;
import ants.core.ByteArray;
import ants.core.NodeAddress;

import java.util.Vector;

public class ColetaReturnCapsule
    extends DataCapsule
{
    final private static byte[] MID = findMID("apps.coleta" +
".ColetaReturnCapsule");
    final private static byte[] PID = findPID("apps.coleta" +
".ColetaReturnCapsule");

    public int infosource;
    public int parentnode;
    public long capTime;
    public long latency;
    public int nodessentto;
    public Vector branches=new Vector();
    public String value1;
    public String value2;
    public String value3;
    public String value4;
    public String value5;
    public String value6;
    public String value7;
    public String value8;
    public String value9;
    public String value10;
    public String value11;
    public String value12;
    public String value13;
    public String value14;
    public String value15;
    public String value16;
    public String value17;
    public String value18;
    public String value19;
    public String value20;
```

```
    protected byte[] mid()
    {
   return MID;
    }

    protected byte[] pid()
    {
   return PID;
    }

    /*
     * Define o tamanho da cápsula de retorno
     *
     */
    public int length()
    {
   return
super.length()+Xdr.INT+Xdr.INT+Xdr.LONG+Xdr.LONG+Xdr.INT+Xdr.INT*nodessentt
o+Xdr.STRING(value1)+Xdr.STRING(value2)+Xdr.STRING(value3)+Xdr.STRING(value
4)+Xdr.STRING(value5)+Xdr.STRING(value6)+Xdr.STRING(value7)+Xdr.STRING(valu
e8)+Xdr.STRING(value9)+Xdr.STRING(value10)+Xdr.STRING(value11)+Xdr.STRING(v
alue12)+Xdr.STRING(value13)+Xdr.STRING(value14)+Xdr.STRING(value15)+Xdr.STR
ING(value16)+Xdr.STRING(value17)+Xdr.STRING(value18)+Xdr.STRING(value19)+Xd
r.STRING(value20);
    }

    /*
     * Monta os campos da cápsula de retorno
     *
     */
    public Xdr encode()     {
   int i;
   Xdr xdr = super.encode();
   xdr.PUT(infosource);
   xdr.PUT(parentnode);
   xdr.PUT(capTime);
   xdr.PUT(latency);
   xdr.PUT(nodessentto);
   xdr.PUT(value1);
   xdr.PUT(value2);
   xdr.PUT(value3);
   xdr.PUT(value4);
   xdr.PUT(value5);
   xdr.PUT(value6);
   xdr.PUT(value7);
   xdr.PUT(value8);
   xdr.PUT(value9);
   xdr.PUT(value10);
   xdr.PUT(value11);
   xdr.PUT(value12);
   xdr.PUT(value13);
   xdr.PUT(value14);
   xdr.PUT(value15);
   xdr.PUT(value16);
   xdr.PUT(value17);
   xdr.PUT(value18);
   xdr.PUT(value19);
   xdr.PUT(value20);
```

```
for (i=0;i<nodessentto;i++)
    xdr.PUT(((Integer)branches.elementAt(i)).intValue());
return xdr;
 }

 public Xdr decode()
 {
int i;
Xdr xdr = super.decode();
infosource=xdr.INT();
parentnode=xdr.INT();
capTime=xdr.LONG();
latency=xdr.LONG();
nodessentto=xdr.INT();
     value1=xdr.STRING();
     value2=xdr.STRING();
     value3=xdr.STRING();
     value4=xdr.STRING();
     value5=xdr.STRING();
     value6=xdr.STRING();
     value7=xdr.STRING();
     value8=xdr.STRING();
     value9=xdr.STRING();
     value10=xdr.STRING();
     value11=xdr.STRING();
     value12=xdr.STRING();
     value13=xdr.STRING();
     value14=xdr.STRING();
     value15=xdr.STRING();
     value16=xdr.STRING();
     value17=xdr.STRING();
     value18=xdr.STRING();
     value19=xdr.STRING();
     value20=xdr.STRING();
for (i=0;i<nodessentto;i++)
    branches.addElement(new Integer(xdr.INT()));
return xdr;
 }

 public boolean evaluate(Node n)
 {
// Checa a latencia
if (latency==0)
{
    latency=(n.time()-capTime)/2;
}
if (n.getAddress()==getDst())
{
    return n.deliverToApp(this,dpt);
} else {
    return n.routeForNode(this,getDst());
}
 }

 /*
  * Mostra as informações de resposta da coleta
  *
  */
 public void showInformation(Node n)
```

```
    {
   ByteArray buf=getData();
   Xdr xdr=new Xdr(buf,0);
   int i,num;
   n.log(0, "*** Recebida a cápsula de retorno de "+
         NodeAddress.toString(infosource));
   if (!value1.equals("zero")) n.log(1,"    "+value1);
   if (!value2.equals("zero")) n.log(1,"    "+value2);
   if (!value3.equals("zero")) n.log(1,"    "+value3);
   if (!value4.equals("zero")) n.log(1,"    "+value4);
   if (!value5.equals("zero")) n.log(1,"    "+value5);
   if (!value6.equals("zero")) n.log(1,"    "+value6);
   if (!value7.equals("zero")) n.log(1,"    "+value7);
   if (!value8.equals("zero")) n.log(1,"    "+value8);
   if (!value9.equals("zero")) n.log(1,"    "+value9);
   if (!value10.equals("zero")) n.log(1,"    "+value10);
   if (!value11.equals("zero")) n.log(1,"    "+value11);
   if (!value12.equals("zero")) n.log(1,"    "+value12);
   if (!value13.equals("zero")) n.log(1,"    "+value13);
   if (!value14.equals("zero")) n.log(1,"    "+value14);
   if (!value15.equals("zero")) n.log(1,"    "+value15);
   if (!value16.equals("zero")) n.log(1,"    "+value16);
   if (!value17.equals("zero")) n.log(1,"    "+value17);
   if (!value18.equals("zero")) n.log(1,"    "+value18);
   if (!value19.equals("zero")) n.log(1,"    "+value19);
   if (!value20.equals("zero")) n.log(1,"    "+value20);
    }

    public ColetaReturnCapsule()
    {
    }

    public ColetaReturnCapsule(int d)
    {
   setDst(d);
    }

    public ColetaReturnCapsule(short sa, short da, int na, ByteArray d)
    {
   super(sa,da,na,d);
    }
}
```

A.5 ColetaCapsuleCacheKey

```
/*
 * ColetaCapsuleCacheKey.Java
 *
 */

package apps.coleta;

import ants.core.Node;
import ants.core.NodeAddress;

/*
 * Responsável pelo controle de cache
```

```
 *
 */

class ColetaCapsuleCacheKey
{
    public long id;
    public int root;
    public int dest;

    public ColetaCapsuleCacheKey(long id, int root, int dest)
    {
   this.id=id;
   this.root=root;
   this.dest=dest;
    }

    public int hashCode()
    {
   return root;
    }

    public boolean equals(Object o)
    {
   if (o!=null&&
       o instanceof ColetaCapsuleCacheKey &&
       this.id==((ColetaCapsuleCacheKey)o).id&&
       this.root==((ColetaCapsuleCacheKey)o).root&&
       this.dest==((ColetaCapsuleCacheKey)o).dest)
       return true;
   return false;
    }
}
```

A.6 ColetaCapsuleCacheValue

```
/*
 * ColetaCapsuleChacheValue.Java
 *
 */

package apps.coleta;

/*
 * Responsável pelo controle de cache
 *
 */

class ColetaCapsuleCacheValue
{
    public long time;

    public ColetaCapsuleCacheValue()
    {
   time=0;
    }
}
```

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo