Universidade Federal de Santa Catarina
Programa de Pós-Graduação em Ciência da Computação

Adriano Fiorese

AVALIAÇÃO DE DESEMPENHO DO USO DE AGENTES MÓVEIS NA GERÊNCIA DE REDES UTILIZANDO TÉCNICAS DE MEDIDAS

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Carlos Becker Westphall

Florianópolis, novembro de 2001

AVALIAÇÃO DE DESEMPENHO DO USO DE AGENTES MÓVEIS NA GERÊNCIA DE REDES UTILIZANDO TÉCNICAS DE MEDIDAS

Adriano Fiorese

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação na Área de Concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____
Prof. Fernando A. O. Gauthier, Dr.
(Coordenador)

Banca Examinadora

_____
Prof. Carlos Becker Westphall, Dr. (Orientador)

_____
Prof[a]. Carla Merkle Westphall, Dr[a]

_____
Prof. Ricardo Filipe Custódio, Dr.

_____
Prof. Otto Carlos Muniz Bandeira Duarte, Dr. (UFRJ-GTA)

_____
Marcelo Gonçalves Rubinstein, Dr. (UFRJ-GTA)

Ao meu pai (*in memorian*).

Agradecimentos

Qual a melhor forma de externar o agradecimento senão o sorriso? A gratidão? A lágrima? O aplauso? Tantas formas incomparáveis e tão cheias de significado. *Emoção, Reconhecimento, Valorização.* São apenas alguns dos motivos pelos quais se faz necessário o registro nesta seção. Nesse sentido vale ainda, um pedido de perdão àqueles que forem negligenciados pela fraqueza de memória.

Aos meus pais Deoclides (*in memorian*) e Zuleide, por sempre confiarem em mim e estimularem a força de vontade necessária para conseguir galgar mais este degrau. A meu pai especialmente pelo caráter legado. Aos meus irmãos Adirciano e Adivan, pela generosidade comigo. À Elza, namorada e companheira de tantas horas, pela compreensão, paciência e carinho.

Ao Prof. Carlos Becker Westphall, pelas orientações e conselhos, confiança, incentivo e amizade. À Profª. Carla Merkle Westphall e ao Prof. Ricardo Filipe Custódio pelas contribuições.

A três grandes amigos: Rosembergue Souza, Prof. Isaias C. Boratti, André Gobbi Sanches pela amizade.

Aos amigos Sandro, André, Josmar, Fabinho, Marafa, Alexandre, Rafael, Rodrigo, Clodoaldo, "Latino", Cunha, Adriano, Eduardo, Juliana, Daniela, "Alemão", Garbin, Fiametti.

À Verinha, Valdete e Beth, funcionárias do Departamento de Informática e Estatística, pela compreensão, dedicação e carinho.

À Universidade Federal de Santa Catarina, por possibilitar espaço fértil para acalentar e concretizar sonhos; por permitir a construção da consciência sobre a importância da pesquisa para o desenvolvimento da sociedade.

À Unioeste-Foz e ITAI, pela confiança na concessão de espaço e equipamentos que possibilitaram a realização dos experimentos.

Lista de Abreviaturas, Siglas e Símbolos

AC	Ambiente Computacional
API	Application Programming Interface
ASN.1	Abstract Syntax Notation Number One
ATM	Asynchronous Transfer Mode
ATP	Agent Transfer Protocol
bps	Bits Por Segundo
BER	Basic Encoding Rules
CMIP	Common Management Information Protocol
COD	Code On Demand
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
CS	Cliente/Servidor
DPI	Distributed Protocol Interface
IA	Inteligência Artificial
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISO	International Organization for Standardization
jdk	Java Development Kit
JNI	Java Native Interface
JVM	Java Virtual Machine
KB	Kilobytes
MASIF	Mobile Agent System Interoperability Facility
MCS	Mobile Code Systems
MIB	Management Information Base
MIT	Management Information Tree
MOS	Messenger Operating System
MTU	Maximum Transmission Unit
NMS	Network Management Station
ORB	Object Request Broker
OSI	Open System Interconnection
PDU	Protocol Data Unit
PPP	Point to Point Protocol
RDPI	Reverse Distributed Protocol Interface
RAM	Random Access Memory
REV	Remote Evaluation
RFC	Request For Comments
RMON	Remote Monitoring
RPC	Remote Procedure Call
SDRAM	Synchronous Dynamic RAM
SNMP	Simple Network Management Protocol
SNMPv3	SNMP versão 3
SO	Sistema Operacional
SOR	Sistema Operacional de Rede
SQL	Structured Query Language
TCP	Transport Control Protocol
TI	Tecnologia da Informação
UE	Unidade de Execução
WAN	Wide Area Network

Lista de Figuras

Lista de Tabelas

Sumário

Resumo

Este trabalho analisa o desempenho de agentes móveis no gerenciamento de redes de computadores comparando-o com o desempenho do modelo cliente/servidor. A análise do desempenho é baseada na comparação do tempo de resposta obtido por meio de técnicas de medidas. Para a obtenção dos valores de tempo de resposta foram construídos protótipos de agentes móveis e de cliente SNMP (modelo cliente/servidor). Foi utilizada a plataforma Aglets1.1.0 para a construção dos protótipos de agentes móveis, bem como a linguagem Java. Java também foi a linguagem de programação utilizada para o desenvolvimento do protótipo de cliente SNMP. Esses protótipos foram executados sobre dois ambientes de teste distintos: rede remota, separada da estação de gerenciamento por enlace de baixa velocidade que caracterizava "gargalo de comunicação"; e rede local, sem "gargalo de comunicação". Para os experimentos com agentes móveis foram utilizados dois protótipos: um que utilizou SNMP, para recuperação de variáveis da MIB II, e outro, que usou JNI para recuperação de variáveis não presentes na MIB II. Para os experimentos com cliente SNMP foi utilizado apenas um protótipo. A análise dos resultados obtidos indica que o cliente SNMP apresenta desempenho melhor tanto em rede remota quanto em rede local, e que entre os agentes móveis o que melhor desempenho apresenta é o que utiliza JNI, para recuperação das variáveis gerenciáveis. Os resultados apontam que as baixas velocidades testadas para o enlace de "gargalo" pouco afetam o desempenho do cliente SNMP sendo que o contrário acontece com os agentes móveis.

Abstract

This work analyzes the mobile agents' acting in the management of network of computers comparing it with the client/server model. The analysis of the acting is based on the comparison of the time of answer obtained by techniques of measures. For the obtaining of the values of time of answer were built mobile agents' and a SNMP client (for the client/server model) prototypes. The platform Aglets1.1.0 was used for the construction of the mobile agents' prototypes, as well as the language Java. Java also was the programming language used for the development of SNMP client prototype. Those prototypes were executed on two differents test environments: remote network, separate from the management station for connection of low speed that characterized " communication bottleneck"; and local network, without "communication bottleneck". For the experiments with mobile agents two prototypes were used: one that used SNMP, for recovery of variables in MIB II, and another, that used JNI for recovery of variables non presents in MIB II. For the experiments with SNMP client was just used a prototype. The analysis of the obtained results indicates that the SNMP client presents so much better acting in remote network as in local network, and that the mobile agents that best performance presents is it that uses JNI, for recovery of the managed variables. The results aim that the low speeds tested for the link affect the SNMP client's and the opposite happens with the mobile agents.

1 Introdução

Com o aumento do tamanho e popularidade das redes de computadores, novos serviços e necessidades estão sendo agregados aos já bastante conhecidos e utilizados. Com isso o gerenciamento da rede torna-se tão necessário quanto a operação e a manutenção. Sendo assim, novos métodos de gerência, que baseiam-se em novas tecnologias ou adaptações das já existentes, vêm sendo propostos e testados.

Com o advento de novos modelos de desenvolvimento de *software*, principalmente com relação à distribuição das aplicações, é natural que esses novos modelos também despertem interesse na área de gerência de redes.

Dentre esses modelos, cita-se o de código móvel. Ao falar dele, naturalmente surge a comparação com o modelo cliente/servidor (CS) que é o mais utilizado atualmente no processamento de informações em ambientes distribuídos.

No modelo cliente/servidor o processamento se dá por meio de requisições e respostas. As requisições são feitas pelo cliente ao servidor. É neste último que estão localizados os recursos e as informações para o processamento. Nesse caso, o fluxo de requisições e respostas pode variar dependendo da tarefa específica que o cliente deseja realizar. Neste caso, poderá haver fluxos de informações intermediários, o que gera de certa forma ineficiência na utilização da rede.

Em contrapartida, no modelo de código móvel as requisições do modelo CS se transformam no próprio código necessário para a realização da tarefa. Nesse caso, o modelo CS pode ser estendido, sendo o cliente representado por um agente com as características de código móvel que, ao executar determinada tarefa, pode ser transferido ao servidor, executar seu código e retornar somente com os resultados do processamento otimizando assim a utilização da rede em conseqüência da eliminação do tráfego intermediário.

Segundo [RUBINSTEIN01] a principal vantagem de se usar o modelo de código móvel, referido como modelo de programação remota, consiste no fato de que somente os resultados de pedidos serão trafegados, diminuindo o tráfego na rede por não ser necessário transmitir comandos, dados do cliente para o servidor e resultados intermediários. Em [GRAY00] percebe-se a mesma abordagem.

Atualmente na gerência de redes de computadores é utilizada uma arquitetura baseada no modelo CS, na qual os gerentes de rede processam informações enviadas pelos clientes que monitoram os recursos da rede. A comunicação entre os clientes e servidores (gerentes e agentes no jargão da gerência de redes de computadores) é regulada pela utilização de protocolos específicos. Os protocolos de domínio público e mais utilizados são o SNMP (*Simple Network Management Protocol*) e CMIP (*Common Management Information Protocol*).

Assim, pensando nisso e nos problemas encontrados na gerência de redes de computadores centralizada, são necessários experimentos que comprovem a efetividade de aplicação do modelo de código móvel para a descentralização, comparado-o ao modelo cliente/servidor já sedimentado.

Deste modo, o trabalho aqui desenvolvido é baseado em comparações entre o modelo atualmente mais adotado na gerência de redes (cliente/servidor com o uso de SNMP), e o modelo de código móvel, baseado em agentes móveis para a gerência de redes de computadores

1.1 Trabalhos relacionados

Em [KOTZ99] é discutida a tendência de utilização de código móvel na Internet e o seu futuro. Apesar das barreiras técnicas e não técnicas, as tendências caracterizadas pelo crescimento não só de usuários mas também pela melhoria das tecnologias e mecanismos atuais, a utilização de código móvel será inevitável, segundo os autores. Essa inevitabilidade seria dada pelo fato de agentes móveis disponibilizarem uma

estrutura de trabalho simples e geral de forma que as aplicações distribuídas e orientadas à informação possam ser implementadas fácil e eficientemente.

No entanto, é interessante ressaltar, conforme [BALDI97] apud [BARAS01], que "validações posteriores com dados quantitativos" serão necessárias para comprovar a efetividade, principalmente quando a proposta é a utilização de agentes móveis no gerenciamento de redes.

Mais considerações a respeito da utilização de agentes móveis podem ser encontradas em [PAPAIOANNOU99]. A utilização de agentes móveis naquele trabalho não se refere à gerência de redes, mas sim a utilização de uma estrutura de trabalho de agentes móveis na exploração espacial. Essa estrutura utiliza a característica de autonomia dos agentes móveis de forma ajustável, utilizada para coordenar, através da cooperação, um conjunto de dispositivos (robôs) na resolução de problemas específicos. Esta técnica, segundo o autor, é mais apropriada que a teleoperação dos dispositivos, pelo retardo existente na transmissão dos sinais de comunicação que esta última impõe.

Como visto em [RUBINSTEIN01], Magedanz *et al*, foram os primeiros a propor a utilização de agentes móveis no gerenciamento de redes de computadores e de telecomunicações [MAGEDANZ96 *apud* RUBINSTEIN01]. Nestes trabalhos, a funcionalidade do agente e do gerente é aumentada através de um ambiente de execução de agentes de modo a habilitar a realização de "aplicações móveis de gerenciamento", implementadas por *scripts* de agentes móveis.

[BRUSIC00] apresenta uma proposta de diminuição do tráfego de mensagens de atualização de localização dos dispositivos móveis em redes de telecomunicações *wireless*, com o uso de agentes móveis. Neste mesmo trabalho é proposta, com o uso de agentes móveis, uma forma de detecção de áreas sem sinal de rádio essencial ao bom funcionamento da rede *wireless*.

[PAGUREK00] considera a utilização do [RFC1228] para criar uma plataforma de gerenciamento de redes que estende agentes tradicionais SNMP para a criação de

MIB's (*Management Information Base*) que podem ser associadas a esses agentes e consultadas/atualizadas por gerentes, ou agentes móveis habilitados a atuarem sobre elas. Este tipo de abordagem garante que aplicações de gerência legadas possam continuar sendo utilizadas.

[BARAS01] discute a utilização da linguagem Java e a mobilidade de código para o desenvolvimento de aplicações de gerenciamento de rede e a forma de integração com sistemas legados com a presença de MIB's (*Management Information Base*) ou não, baseada em *callback*, habilidade de processamento da máquina virtual java (JVM), mesmo que mínima, nos elementos gerenciados, em conjunção com JNI (*Java Native Interface*) e API's (*Application Programming Interface*) embutidas. Neste trabalho, é apresentada uma arquitetura de sistema que utiliza agentes instanciados por um gerente-coordenador que delega esses agentes para os elementos gerenciados.

Proposta de arquitetura de evolução dinâmica de software, aplicada à gerência de redes pode ser encontrada em [FENG01]. Com esta arquitetura é possível desenvolver aplicações que contenham módulos, cuja função é a disponibilização de serviços, atualizáveis dinamicamente em tempo de execução. Essa arquitetura utiliza características de mobilidade forte e também reflexão para disponibilizar atualização de módulos de software de um sistema de gerenciamento de rede baseado em SNMPv3 (*Simple Network Management Protocol versão 3*), através de *hot-swapping*. Desta forma, não há necessidade de recompilação ou outras formas de adaptação dos agentes ou gerentes participantes do sistema de gerenciamento, e a descontinuidade do serviço de gerenciamento não ocorre.

A utilização de agentes inteligentes nas arquiteturas de gerenciamento de redes é descrita em [NASCIMENTO99].

Na área de avaliação de desempenho dos agentes móveis, vários trabalhos vêm sendo realizados. [ISMAIL99] apresenta uma avaliação de desempenho dos agentes móveis e do modelo cliente servidor. Vários custos de etapas de migração de um agente móvel são apresentados em uma série de resultados de implementação. Tem-se em

[COSTA99] a análise comparativa, através de expressões analíticas, do desempenho de agentes móveis e do SNMP em diferentes topologias de rede. Já [RUBINSTEIN01] apresenta uma comparação quanto a escalabilidade de agentes móveis e SNMP, sendo realizadas medidas práticas para obtenção de valores utilizados na simulação da utilização de agentes móveis e agentes SNMP, variando diversos parâmetros, inclusive de topologia de rede, como velocidade do *link* de comunicação entre os nós gerenciados e latência de acesso à rede, tendo em vista a utilização de agentes móveis em larga escala. [KOTZ00] apresenta a avaliação de agentes móveis na filtragem de dados em redes sem fio, realizando simulações e experimentos para validar o seu modelo analítico. Ele usa um fluxo contínuo de dados que chega até um servidor por meio de uma rede com fio. Agentes móveis, que realizam a filtragem dos dados, são enviados através da rede sem fio até este servidor e nele executam retornando ao cliente com os dados relevantes. Naquele trabalho são usadas duas métricas para sua avaliação: a largura de banda requerida para a rede sem fio e o tempo de processamento. [GRAY01] apresenta uma comparação entre agentes móveis desenvolvidos em três plataformas de agentes móveis diferentes e uma aplicação baseada no modelo CS, para um sistema de filtragem e recuperação de documentos armazenados em um servidor. Naquele trabalho é comparado o tempo total de realização da tarefa de recuperar documentos dentre o total de 60, segundo uma taxa de seleção (percentagem) que pode variar em uma rede local em que várias configurações de largura de banda são possíveis. [PAVLOU00] descreve o projeto, a implementação e a avaliação de um sistema simples de monitoramento de desempenho baseado em agentes móveis. O trabalho tem como meta avaliar o uso da tecnologia de agentes móveis em comparação às abordagens de objetos estáticos em ambientes de gerenciamento de redes.

1.2 Escopo do trabalho

No campo da gerência de redes de computadores, um dos problemas clássicos é a quantidade de informações de gerência que trafega sobre o meio de comunicação que interliga os computadores e/ou as próprias redes. Em muitos casos, é necessária a construção de outra rede dedicada exclusivamente ao tráfego das informações de

gerência, pois de outra forma, a utilização da rede por parte das aplicações de uso "normal" seria impedida; recomeçando, desta forma, o ciclo de necessidades.

Esse problema se acentua ainda mais quando a gerência de redes deve atuar em *links* de baixa velocidade. Em se tratando de código móvel também. O fato da transmissão de dados intermediários durante as operações de gerência que utilizam o protocolo SNMP, leva a crer que a pequena largura de banda desse "gargalo" faça com que a tarefa total de recuperação dos dados de gerência seja prejudicada em tempo de resposta, justamente pela alta utilização do *link*. Além disso, a obtenção de um conjunto de dados válidos de gerência, de um conjunto de nós, requer alto tráfego de dados sob o *link* de baixa velocidade, o que também pode influenciar no tempo de resposta total pela falta de velocidade adequada do enlace.

Uma possibilidade de tratamento para o problema citado, e a necessidade de maiores validações do uso de agentes móveis como plataforma de gerenciamento de redes, é a base e motivação deste trabalho.

Assim, observando os diversos trabalhos relacionados e as necessidades já relatadas, será abordado neste trabalho o desempenho da utilização de agentes móveis na obtenção de valores de variáveis de gerência de redes, em comparação ao modelo *Remote Procedure Call* (RPC), utilizado por aplicações de gerenciamento de rede que utilizam o protocolo SNMP.

Dessa forma, foi pensada a elaboração de experimentos que ajudassem a avaliar o desempenho dos agentes móveis para atuação na gerência de redes, separadas por um *link* de comunicação de baixa velocidade. Essa abordagem contempla a técnica de *benchmarking*.

Foi observado que nos trabalhos de [COSTA99] e [RUBINSTEIN01] há comentários sobre a aplicabilidade de agentes móveis na gerência de redes interligadas por "gargalos" de comunicação, sendo inclusive realizadas simulações.

Nesse sentido, é proposto, através de medições práticas com a ajuda de protótipos desenvolvidos neste trabalho, avaliar o desempenho comparando valores de tempo de resposta obtidos com agentes móveis e chamadas RPC, executadas por um cliente SNMP. Esses tempos são o resultado da obtenção e retorno das informações de três variáveis da MIB II – *ifInErrors*, *sysContact e sysDescr*, conforme definido em [COSTA99], para a estação gerenciadora de rede (NMS), que se encontra do outro lado do "gargalo". Os protótipos foram desenvolvidos na linguagem Java, sendo que os agentes móveis foram desenvolvidos sob a plataforma de agentes móveis Aglets1.1.0.

O escopo deste trabalho também contempla a avaliação dos tempos de resposta obtidos por agente móvel que utiliza a tecnologia JNI. Ele obtém informações relacionadas à quantidade de memória RAM – *Random Access Memory* - (disponível e total), memória virtual (disponível e total) e tamanho do espaço disponível em disco da máquina visitada.

1.3 Organização do trabalho

A utilização de código móvel vem sendo cada vez mais discutida no âmbito da gerência de redes. Desta forma, este trabalho busca aproveitar essa discussão e realizar a análise de desempenho da utilização de agentes móveis usando, para validação, técnica de medida consistindo no levantamento da métrica tempo de resposta, com a utilização de agentes móveis e do protocolo SNMP.

Desta forma, este trabalho encontra-se organizado da seguinte forma:

Inicialmente, é discutido o aspecto de mobilidade de código no Capítulo 2 – Fundamentos da Mobilidade de Código e os Agentes Móveis. Este capítulo apresenta modelos arquiteturais de *Mobile Code Systems* (MCS) em comparação com sistemas distribuídos tradicionais. Além disso, discute o aspecto de mobilidade de código (mobilidade forte ou fraca), apresentando a definição para agentes móveis. Além disso, são apresentadas algumas razões para a utilização da abordagem de agentes móveis;

comparativamente ao modelo de sistemas distribuídos e/ou cliente/servidor em geral. Nesse sentido, é visto que algumas das razões que compreendem a utilização de agentes móveis são:

- Conservação da largura de banda;
- Redução no tempo total de realização da tarefa;
- Redução na latência;
- Operação *off-line* e computação móvel;
- Balanceamento de carga;
- Distribuição dinâmica.

Além disso, ainda no Capítulo 2 é aproveitada a oportunidade de apresentar os exemplos mais comuns de aplicações que utilizam a abordagem de agentes móveis.

Já no Capítulo 3, é abordado o assunto de gerência de redes. Neste capítulo, o enfoque principal baseia-se na discussão da problemática da centralização das operações de gerenciamento da rede e na introdução de conceitos relativos a utilização de agentes móveis para a realização das atividades. Assim, é focalizada a atuação de código móvel na gerência de redes. Para tanto, são abordadas as desvantagens da centralização das operações de gerenciamento, bem como aspectos de descentralização dessas operações. Também são tratados outros aspectos relacionados com a descentralização, como o gerenciamento por delegação, o gerenciamento através de código sob demanda e um aspecto importante no sentido de integrar tecnologia nova com a migração de código por meio de agentes móveis e aplicações de gerenciamento legadas, que utilizam o modelo agente/gerente SNMP, que foi chamada de Flexibilidade com o uso da Extensão. Nesta seção, basicamente são apresentados dois protocolos que “conversam” com o protocolo SNMP implementado pelos agentes legados e que podem ser utilizados para o desenvolvimento de agentes (móveis ou não) que estendam as MIB’s suportadas por esses ambientes de gerência.

Já no Capítulo 4, são ilustradas várias plataformas de agentes móveis, classificadas por linguagem de programação que utilizam para que sejam desenvolvidos os agentes. Neste capítulo é traçado sucintamente um paralelo entre elas e destacadas

algumas diferenças, principalmente com relação à característica de mobilidade de código.

No Capítulo 5, são relatados os experimentos realizados, explicitando a maneira como foram executados e os resultados obtidos. Basicamente são apresentados os gráficos de tempo de resposta obtidos pelos vários experimentos e destacados os detalhes de implementação para que possam ser repetidos. Também, será feita uma comparação entre os resultados obtidos no ambiente de teste de rede remota e também de rede local.

Já no Capítulo 6 é apresentada a conclusão. Para tanto, é racionalizada a análise dos dados obtidos, tendo em vista as várias restrições existentes e as imperfeições naturais de experimentos empíricos e de tratamento prático, bem como as expectativas com relação à trabalhos futuros. O Capítulo 7 relata as referências bibliográficas.

Finalmente, há os anexos contendo o código fonte utilizado nos experimentos. Assim, é apresentado o código dos agentes móveis com suporte a SNMP e JNI, bem como do aplicativo SNMP e da biblioteca que disponibiliza o suporte a JNI para o agente móvel.

2 Fundamentos da Mobilidade de Código e os Agentes Móveis

Várias propostas de Sistemas de Código Móvel (MCS - *Mobile Code Systems*) estão disponíveis, tanto comerciais quanto acadêmicas. A pluralidade deste ambiente propicia ambigüidade de conceitos, abstrações, termos e semânticas quando se tratando de mobilidade de código. Essa ambigüidade dificulta a compreensão, a avaliação e discussão de técnicas, metodologias, ferramentas e aplicações de MCS's.

A terminologia utilizada muitas vezes deixa a desejar, devido a confusão existente muitas vezes entre as próprias áreas de conhecimento dentro das Ciências da Computação. Essa "confusão" é caracterizada pelo uso excessivo do termo "agentes móveis". Dentro da área de sistemas distribuídos, o termo agente móvel é usado para denotar um componente de *software* que apresenta mobilidade, ou seja seu código pode ser transferido para diferentes sistemas computacionais desde que estes possuam alguma comunicação, permanente ou não. Na área da Inteligência Artificial, no entanto, o termo "agentes móveis" é utilizado para definir um componente de *software* orientado a objetivos, capaz de reagir e executar ações em um ambiente dinâmico [WOOLDRIDGE95]. O agente da Inteligência Artificial é guiado pelo conhecimento da relação entre eventos, ações e objetivos. Este conhecimento pode ser incrementado através da comunicação entre agentes ou por mecanismos inferenciais [GENESERETH94]. Apesar da mobilidade não ser uma característica essencial desses componentes, existe uma tendência de se misturar a noção de agentes inteligentes com a definição encontrada na área de sistemas distribuídos. Ao mesclar estes dois conceitos, tem-se a crença de que todo agente móvel é inteligente e vice-versa.

Assim sendo, para que se entenda o funcionamento de um sistema de código móvel é necessário que haja consciência da sua arquitetura.

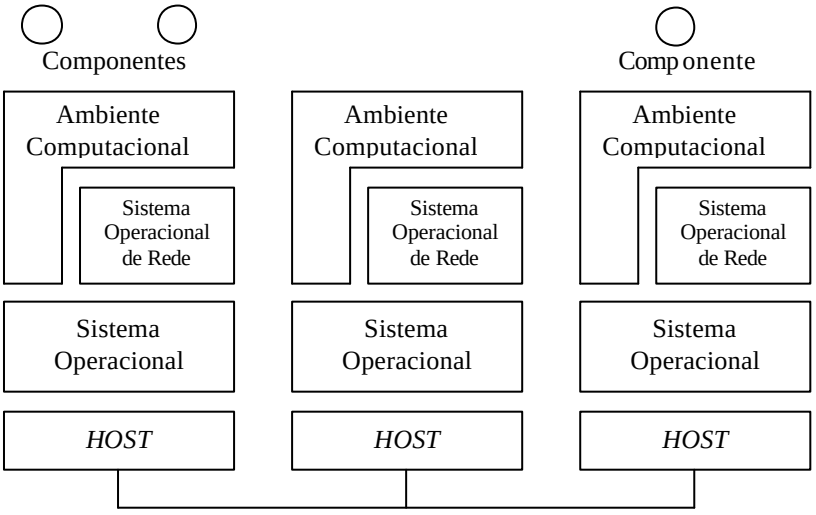


Figura 2.1 - Sistema de código móvel

Neste modelo arquitetural [VIGNA98] nota-se a camada inferior chamada de *Host* representando o *hardware* da máquina. A camada logo acima, Sistema Operacional (SO), é responsável por prover as funcionalidades básicas de um sistema operacional como sistema de arquivos, gerenciamento de memória, gerenciamento de processos, etc. Nesta camada não se encontra suporte para comunicação ou interconexão de máquinas. Estas funções são oferecidas pela camada do Sistema Operacional de Rede (SOR), onde serviços de endereçamento possibilitam a comunicação entre duas ou mais máquinas conectadas fisicamente. Assim como uma pilha de protocolos, a camada do SOR utiliza serviços da camada imediatamente inferior a ela, o SO, e assim por diante. Na última camada tem-se o segredo desta arquitetura. Cada máquina participante do MCS tem uma camada de *software* chamada Ambiente Computacional (AC), responsável por oferecer às aplicações, serviços de alocação e transferência dos componentes para outros nós da rede. Essa camada é consumidora dos serviços do SO para preparar o código e os dados para uma transferência qualquer.

Essa arquitetura contrasta com a arquitetura de um Sistema Distribuído que utiliza uma camada responsável por oferecer transparência ao usuário conhecida como *Object Request Broker* – ORB (conhecidos como sistemas distribuídos orientados a objetos). Nos Sistemas de Código Móvel essa camada não existe, pois a característica desse tipo de sistema é a necessidade de se especificar o local onde cada componente será executado.

Até a camada SOR, a arquitetura de ambos é idêntica. A última camada é que irá distinguir, como já dito, um sistema distribuído orientado a objetos de um sistema de código móvel. Na última camada da Figura 2.2 [VIGNA98] é oferecida uma arquitetura onde componentes localizados em máquinas diferentes são vistos e utilizados pelo usuário como se estivessem em sua própria máquina. Através dessa transparência, não é necessário o conhecimento da topologia da rede onde o sistema é executado, nem determinar qual o nó da rede que irá oferecer e executar um serviço que seja invocado. Os serviços e facilidades do *Common Object Request Broker Architecture* (CORBA) são exemplos deste conceito, uma vez que os sistemas desenvolvidos com base nesta arquitetura dispõem do ORB para realizar tarefas como a localização e busca dos componentes.

A arquitetura do modelo de código móvel se assemelha muito à arquitetura dos sistemas distribuídos, diferindo porém, na última camada, responsável por prover o suporte a serviços de distribuição ao sistema. A substituição por diversos ambientes computacionais (AC), de acordo com o número de nós da rede que irão participar do sistema, é notada na Figura 2.1. Dessa forma, acaba-se associando a cada AC a identidade da máquina onde ele está sendo executado. O propósito de cada AC é oferecer às aplicações serviços de alocação e transferência dos componentes para outros nós da rede.

É necessário que se comente sobre os componentes dispostos sobre o modelo de arquitetura, tanto dos sistemas distribuídos orientados a objeto, quanto dos sistemas de código móvel. Sendo assim, para que se entenda um componente, este é subdividido em unidade de execução (UE) e recursos. As unidades de execução representam qualquer fluxo seqüencial de computação como por exemplo processos *single thread* ou *threads* individuais de processos *multi-thread*. E sobre os recursos, estes são as entidades compartilhadas entre duas ou mais unidades de execução. Exemplos típicos de recursos são: sistemas de arquivos, variáveis do sistema operacional, etc. A Figura 2.3 nos dá uma amostra deste modelo.

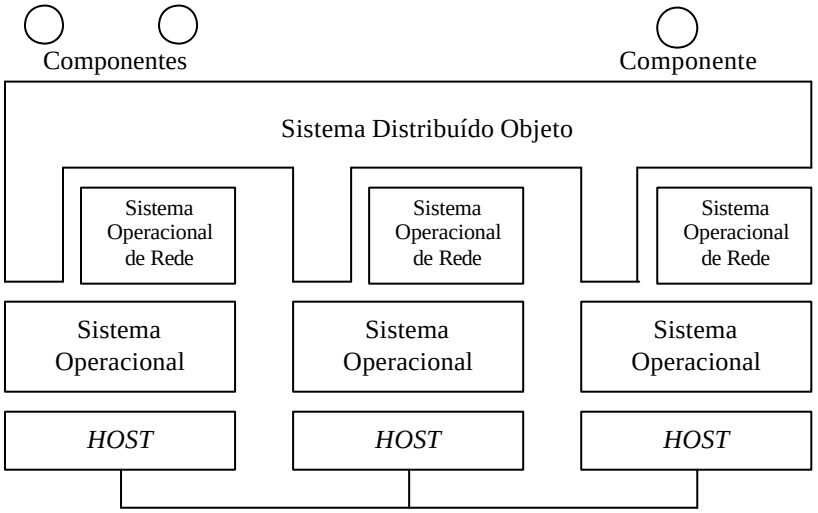


Figura 2.2 - Sistema distribuído objeto

O estado de um componente é formado por um espaço de dados e um estado de execução. O espaço de dados é o conjunto das referências dos componentes (UE + recursos) que podem ser acessados e utilizados. Esse paradigma permite que estas referências apontem para componentes remotos, ou seja, localizados em máquinas de endereços distintos na rede. O estado de execução contém dados privados que não podem ser compartilhados tais como informações de controle (apontador de instrução, *call stack*) referentes ao estado da unidade de execução. Conforme [VIGNA98]:

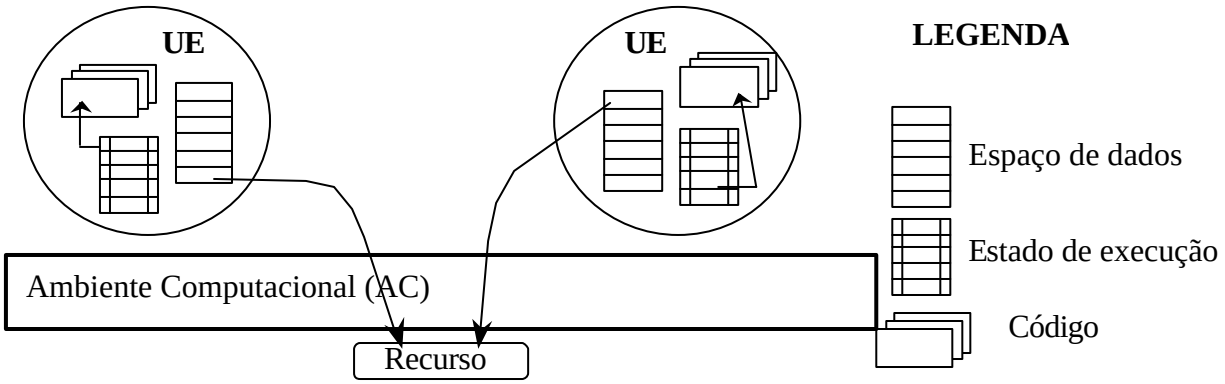


Figura 2.3 – Componentes da arquitetura de sistemas de código móvel

2.1 Aspectos da mobilidade de código

Os sistemas baseados em código móvel apresentam algumas características que os distinguem do modelo tradicional de aplicações distribuídas. Dentre essas características foi selecionada a de mobilidade, por melhor se adequar aos objetivos deste trabalho. Outras características distintas entre os sistemas distribuídos tradicionais e os de código móvel envolvem aspectos de segurança e também de comunicação.

Assim, nos sistemas de código móvel, o segmento de código, estado de execução e espaço de dados da unidade de execução podem ser transferidos para um ambiente computacional diferente e disponível.

Nesse sentido, os sistemas de código móvel possuem duas características quanto a essa transferência de execução denominada mobilidade. Há os sistemas com mobilidade forte e os sistemas com mobilidade fraca.

> *"(1) Mobilidade forte, onde o sistema captura os dados, estados de execução e código do agente, permitindo continuar a execução do ponto exato da interrupção; (2) mobilidade fraca, onde o sistema captura somente os dados e código, e chama um ponto de entrada conhecido no código para reinicializar o agente na nova máquina."*[Traduzido pelo autor de GRAY00 et al, 2000, pg. 25]

A mobilidade forte permite que o estado completo de um agente móvel seja capturado, sendo dessa forma possível a ele, com segurança, reiniciar sua operação no destino do ponto exato onde estava antes de mover-se.[SURI00 apud FENG01].

Na mobilidade fraca o código ao ser transferido para o ambiente computacional destino é associado dinamicamente a uma unidade de execução ativa naquele ambiente (*code fragment*) ou é aproveitado para gerar uma nova unidade de execução (*stand-*

alone code). A transferência do código pode ser iniciada pela unidade de execução proprietária do código (*code shipping*), ou ainda, requisitada por outra unidade de execução que precise incorporá-la ao seu próprio código (*code fetching*).

Segundo [GRAY00], o modelo de mobilidade forte é mais adequado ao programador final, porém mais trabalhoso para o desenvolvedor do sistema, uma vez que rotinas para capturar o estado de controle devem ser adicionadas aos interpretadores existentes.

A Figura 2.4 apresenta um resumo da classificação dos aspectos de mobilidade, segundo [VIGNA98].

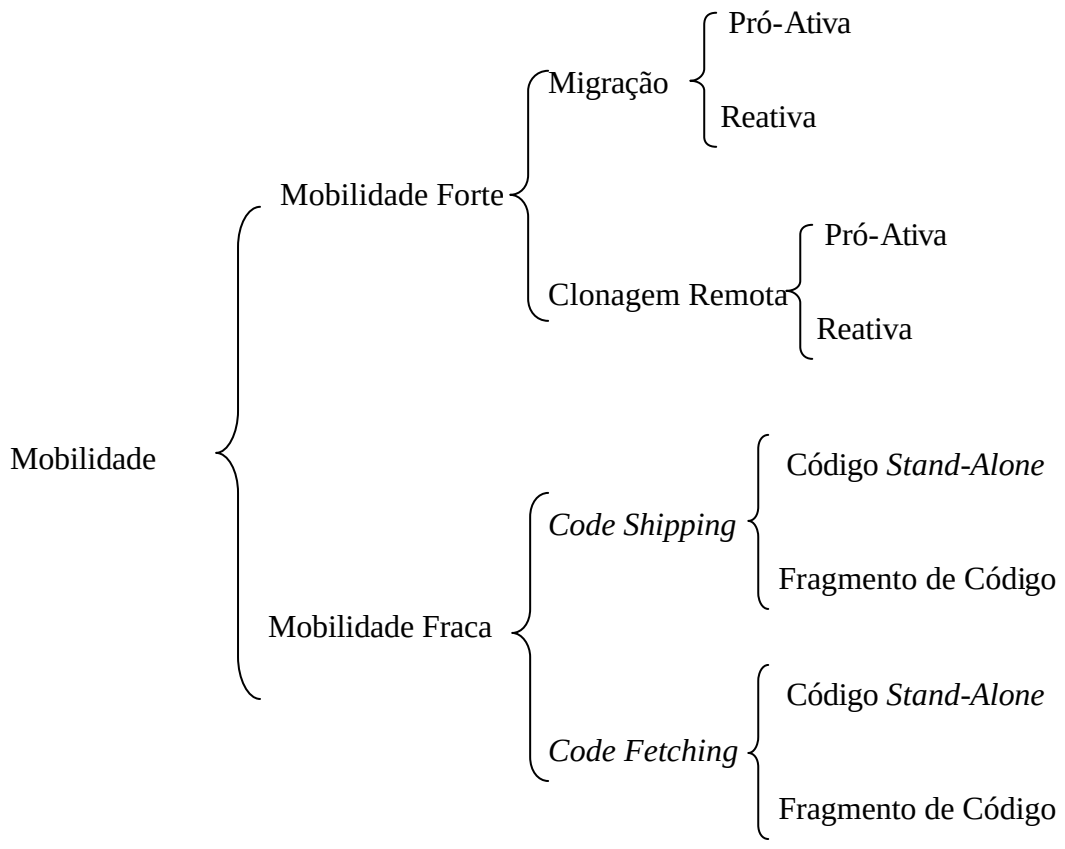


Figura 2.4 – Uma classificação dos mecanismos de mobilidade

2.2 Agentes Móveis

Assim, considerando os conceitos já discutidos, será utilizado no contexto deste trabalho o termo “agentes móveis” como uma especialização de “unidade de execução”. Essa especialização refere-se ao conceito atribuído a agentes móveis na área de Sistemas Distribuídos, onde agente móvel representa um componente de *software* com a característica de mobilidade.

2.3 Razões para o uso de agentes móveis

Agentes móveis são uma nova tecnologia e, segundo [GRAY00], são programas que podem migrar de máquina em máquina em uma rede heterogênea. Assim, é importante considerar em profundidade a utilidade e as situações em que eles podem ser usados efetivamente. A seguir, serão apresentadas seis razões para o uso de agentes móveis.

2.3.1 Conservação da largura de banda

No modelo tradicional cliente/servidor, o servidor disponibiliza um conjunto fixo de operações que um cliente invoca através da rede. Se o servidor não disponibiliza uma simples operação que case com a tarefa exata que o cliente quer realizar, o cliente deve invocar uma série de operações do servidor ou o desenvolvedor do servidor deve adicionar uma nova operação no servidor. A primeira opção traz dados intermediários pela rede em cada operação, podendo desperdiçar uma significativa quantidade de largura de banda da rede, especialmente se os dados intermediários não são usados na tarefa do cliente. A segunda opção torna-se uma sobrecarga de programação intratável, devido ao crescimento do número de clientes. Ainda, essa opção não é tratada pela engenharia de *software* moderna, uma vez que o servidor torna-se uma coleção de rotinas complexas e especializadas.

Um agente móvel, no entanto, não desperdiça largura de banda mesmo se o servidor disponibiliza apenas operações de baixo nível, simplesmente por que o agente move-se até o servidor. O agente realiza a seqüência de operações localmente, e retorna apenas o resultado final para o cliente. Agentes que fazem mais trabalho evitam mais mensagens intermediárias e conservam mais largura de banda.

Um fator que favorece agentes móveis na conservação da largura de banda é o tamanho deles. Em geral, são de tamanho pequeno. Mesmo porque o objetivo do agente móvel é a realização de determinada tarefa. É dessa forma que o pequeno tamanho pode ser explicado. Primeiramente, quanto menos passos para a realização de determinada tarefa, menor será o agente. O uso de linguagens de programação interpretadas com poderosos conjuntos de comandos para os mais variados tipos de processamento de informações, sejam elas *strings* de caracteres ou não, ajuda muito. Segundo, os agentes móveis realizam somente aqueles passos. A aplicação principal permanece na máquina cliente ou servidora. Em terceiro lugar, comentários e espaços em branco desnecessários são removidos do código antes dele ser enviado para outra máquina. Isto faz o agente ficar ainda menor.

No entanto, há de ser observado um fato, quando discutindo-se uso da largura de banda *versus* tamanho do agente. O cabeçalho do agente, que contém informações sobre a origem do agente e proprietário. Esse valor é fixo para a maioria dos agentes móveis[1] e é transmitido a cada movimento do agente. Assim, o cabeçalho vai influenciar no consumo de largura de banda.

Levando em consideração o tamanho do agente móvel e outras características, a discussão do consumo da largura de banda só tem sentido quando é realizada a comparação entre dois componentes tecnológicos. Por exemplo, o modelo cliente/servidor baseado em RPC, e agentes móveis. Nesse sentido, pode-se estabelecer uma relação de equivalência entre um determinado agente móvel e um determinado cliente RPC, e grupos do mesmo agente e grupos do mesmo cliente. Assim, dado que

[1] Depende da plataforma de desenvolvimento. Aproximadamente 1Kb (GRAY00)

um agente consuma menos largura de banda que um cliente, um conjunto desses agentes consumirá menos largura de banda que um conjunto daqueles clientes RPC.

> *"Para todos os casos onde um único agente usa menos a rede que um único cliente RPC, um grupo daqueles clientes usará menos a rede que um grupo equivalente de clientes. Os agentes produzirão poucas colisões e poucas retransmissões, e cada agente individual estará ainda gerando menos tráfego."* [Traduzido pelo autor de GRAY et al. 2000, pg. 8].

Naturalmente, em algumas aplicações, o código do agente será maior que qualquer resultado intermediário[2] e o agente consumirá mais largura de banda que a solução cliente/servidor equivalente. Assim, a meta de muitos pesquisadores em agentes móveis é disponibilizar introspecção e capacidades de comunicação suficientes para que o agente possa estimar o uso potencial da largura de banda por ele mesmo e decidir se é melhor mover-se para a localização dos dados ou permanecer estacionado e interagir com os dados através da rede (como se ele fosse um cliente tradicional).

2.3.2 Redução no tempo total de realização da tarefa

Como visto anteriormente, a arquitetura de agentes móveis muitas vezes utiliza menos largura de banda que um cliente RPC. Assim, com exceção do usuário que está pagando pelo uso da rede por *byte*, o usuário final está interessado na conservação da largura de banda se ela induz a uma redução no tempo total de realização da tarefa. A principal razão pela qual um agente pode completar tarefas mais rápido que clientes RPC é o fato que agentes móveis evitam o tempo que seria consumido pela transmissão de resultados intermediários, e em certas ocasiões, pela transmissão das próprias informações desejadas.

[2] Principal característica de desperdício da largura de banda pelo modelo tradicional baseado em RPC

2.3.3 Redução na latência

A meta primária das aplicações de recuperação de informação, utilizando agentes móveis, é reduzir o tempo total de realização da tarefa, isto é, o tempo entre o instante no qual o usuário submete a tarefa e quando ele recebe os resultados. No modelo tradicional cliente servidor, o tempo que cada chamada RPC leva para “cruzar o cabo” não é importante, uma vez que a latência é pequena relativa ao tempo de processamento das informações e o tempo de transmissão dos resultados.

Há aplicações, no entanto, em que os tempos para as submissões, para obtenção dos resultados e para o processamento são pequenos. Nestes casos, os tempos de latência podem dominar o tempo total de execução da tarefa. O melhor exemplo de uma aplicação destas é aquela que deve reagir rapidamente a algum evento enviando algum *status* ou informação de controle. Nestes casos, o componente reativo seria implementado como um agente móvel. Ele pode mover-se para perto do produtor dos eventos, do consumidor do *status* ou informação de controle, ou ambos. Ainda, o agente pode mover-se se o conjunto de produtores e consumidores ou as condições da rede mudam constantemente, colocando-se na melhor localização da rede. Assim, as informações de controle ou *status* alcançarão os consumidores mais rapidamente do que se o componente reativo fosse instalado em uma localização fixa na rede. Nas situações onde um administrador tem acesso a todas as máquinas da rede, e pode pré-instalar o componente reativo em cada localização dela, mesmo assim a proposta de agentes móveis ainda é atrativa. Essa proposta elimina a necessidade de pré-instalação e disponibiliza um modo simples de transferir o controle reativo de uma localização para outra. O agente simplesmente se move.

O mais simples dos exemplos é qualquer aplicação distribuída que necessite interagir com usuário através de uma interface gráfica. Assim, enviando o componente de *software* que cuida da interface gráfica para a localização do cliente, a aplicação poderá responder mais rapidamente às ações do usuário. É o caso dos *applets* Java.

2.3.4 Operação *off-line* e computação móvel

Um computador móvel muitas vezes desconecta-se da rede, e quando retorna, pode encontrar-se em uma conexão radicalmente diferente da anterior em termos de largura de banda, latência ou confiabilidade. A simples habilidade dos agentes de migração para o outro lado de um *link* de rede é uma vantagem grande, uma vez que o agente pode evitar o uso intensivo da pouca largura de banda ou alta latência do *link*, e continuar a tarefa dele, mesmo que o *link* tenha caído. Em redes sem fio, onde os computadores estão continuamente movendo-se, os agentes móveis são mais vantajosos ainda. As conexões de rede oscilam entre ativas e não ativas, e quando um computador tem uma conexão ativa, não pode contar com ela por muito tempo. Assim, existindo tarefas que necessitem ser executadas em ambientes de execução dessa natureza, a melhor alternativa é a transformação da tarefa a ser realizada em um agente móvel e movê-lo para a localização adequada do que tentar realizar esta tarefa passo a passo, seqüencialmente, durante breves períodos de conexão.

Em outras palavras, aplicações tradicionais cliente/servidor requerem uma conexão contínua e ainda são forçadas a parar a execução de determinada tarefa que envolva comunicação enquanto a conexão não está ativa. Um agente móvel, no entanto, pode mover-se do cliente para um servidor ou máquina *proxy*, ou vice-versa, e então trocar informações ou realizar a tarefa específica. Se o *link* cair neste momento, o diálogo continua e o agente retornará com os resultados quando o *link* for restabelecido. A habilidade de executar “desconectado” da máquina cliente ou servidora é um dos grandes fatores de motivação para o uso de agentes móveis em aplicações distribuídas e também para a computação móvel. Essa habilidade faz com que nem a máquina que requisita algo, usando agentes móveis, nem quem requisita, necessitem estar disponíveis durante o tempo de resolução desta requisição.

2.3.5 Balanceamento de carga

Problemas de balanceamento de carga aparecem em um sistema de computação distribuído quando existe alocação de trabalho desigual nos diferentes elementos de processamento do sistema. O desempenho aumenta se o trabalho pode ser distribuído eqüitativamente entre os elementos. Muitas técnicas de balanceamento de carga existem, cada qual com o objetivo de aumentar o desempenho por meio do particionamento das aplicações distribuídas em componentes distribuídos entre múltiplos processadores.

Balanceamento de carga pode ser feito estaticamente ou dinamicamente. O balanceamento de carga é dito estático se o trabalho de alocação é baseado em uma análise *a priori* do conjunto de dados de entrada e da computação executada. Uma vez alocado o trabalho, tipicamente antes da execução iniciar, ele não é mais realocado durante o ciclo de vida da computação. O balanceamento de carga é dinâmico se a carga de trabalho é realocada durante a execução.

Agentes móveis suportam facilmente o balanceamento de carga dinâmico através de três características chave: eles podem mover-se de uma plataforma para outra, eles podem mover-se entre plataformas heterogêneas (geralmente com o uso de uma máquina virtual) e eles carregam consigo o código da aplicação. Desta forma, os agentes móveis suportam a forma mais geral e efetiva de balanceamento dinâmico de carga: a execução de processos implementados como agentes móveis que podem migrar através de controle interno ou externo para qualquer nó habilitado a rodar agentes móveis dentro do sistema distribuído.

No modelo agentes móveis, os componentes de uma computação distribuída, que são candidatos para balanceamento de carga, são implementados como agentes móveis. Máquinas que participam do sistema de balanceamento de carga executam um servidor de agentes móveis, da mesma forma que outros sistemas de balanceamento de carga requerem a pré-instalação de alguma infra-estrutura apropriada. Os agentes móveis podem iniciar a execução de qualquer lugar na rede e, em resposta a sinais de controle

baseados no monitoramento da carga distribuída, migrar para novas máquinas. Desde que a migração de agentes móveis permita a determinação da máquina para a qual irão migrar, em tempo de execução, a máquina pode ser determinada por um módulo de balanceamento de carga externo. Muitos algoritmos de controle de balanceamento de carga dinâmico foram propostos, e qualquer um destes algoritmos pode ser usado como a lógica de controle dentro do módulo de balanceamento de carga, permitindo que o código do agente seja independente do esquema atual de balanceamento de carga.

Muitos sistemas de balanceamento de carga não são tão flexíveis quanto agentes móveis, pois falta a esses sistemas uma das três características chave já apresentadas. Eles não permitem que um processo executante possa se mover, ou suportam apenas plataformas homogêneas ou requerem que o código da aplicação seja pré-instalado em cada máquina. Por outro lado, eles permitem o balanceamento de carga com código compilado, coisa que nenhum sistema de agentes móveis suporta.

Como os sistemas de agentes móveis atuais usam linguagens interpretadas, ao invés de código compilado, a decisão de usar agentes móveis para balanceamento de carga deve levar em consideração não somente o custo da migração, mas também as penalidades de velocidade da interpretação. Uma vez sendo os agentes móveis interpretados, uma aplicação que realiza computação intensiva verá uma queda de execução significativa somente por ela ter sido implementada com agentes móveis ao invés de um programa tradicional compilado. Por esta razão, sistemas de agentes móveis não são ainda apropriados para aplicações em que o balanceamento de carga é a principal razão de existência, ou seja aplicações em que o tempo de CPU (*Central Processing Unit*) é o único recurso de interesse. Se agentes móveis são usados por outras razões, entretanto, a carga pode ser balanceada facilmente pela migração dos agentes de máquina em máquina.

2.3.6 Distribuição dinâmica de software

Como mencionado neste capítulo, se um servidor tradicional não suporta diretamente uma determinada operação, o cliente deverá realizar múltiplas chamadas RPC para o servidor para realizar aquela tarefa, transmitindo resultados intermediários sobre a rede em cada chamada. Aplicações que usam agentes móveis podem evitar esta transmissão desnecessária. Esta "nova" operação do servidor pode ser implementada como um agente móvel e mandada do cliente para a máquina servidora, onde é executada localmente sobre os recursos do servidor, retornando somente o resultado final para o cliente. O agente pode realizar a operação um única vez, terminando sua execução tão rápido quanto possível, ou permanecer realizando esta operação para as requisições futuras do cliente.

Deste modo, o uso de agentes móveis representa exemplos específicos de evolução dinâmica de software, onde uma aplicação dinamicamente instala um componente de *software* em alguma máquina remota, e então invoca aquele componente de *software* como se ele fosse parte dos serviços pré-instalados daquela máquina. No caso dos agentes móveis, o agente essencialmente se instala e se invoca, uma vez que ele simplesmente migra para uma máquina remota e continua lá a execução.

Muitas outras abordagens de distribuição dinâmica de software existem. Abordagens comuns para empresa como ferramentas de gerenciamento de *software*, e sistemas de gerenciamento UNIX relativamente clássicos são significativos para um administrador de sistema que necessita instalar um pacote de *software* completo, e não são suficientemente leves para um programa cliente que simplesmente queira estender a funcionalidade do servidor com uma simples operação. Outras abordagens são mais aplicáveis a *softwares* cliente. Por exemplo, muitos sistemas de procedimentos armazenados (*stored procedures*) particularmente aqueles presentes em servidores de banco de dados SQL (*Structured Query Language*), permitem ao cliente carregar (*upload*) código para posterior utilização. O código é armazenado no servidor e pode ser invocado tantas vezes quanto necessário ou desejado. De outra forma, a abordagem *Remote Evaluation* (REV) manda o código do procedimento como parte de cada

invocação. Classes dinâmicas que são uma variação de *stored procedures* e REV permitem que uma nova subclasse de uma superclasse pré-conhecida seja dinamicamente inserida em um programa sendo executado. Neste caso, o cliente envia uma instrução de inserção para o programa remoto. A instrução inclui a localização de onde o programa pode encontrar uma biblioteca compartilhada contendo a implementação da subclasse.

O obstáculo para todas essas abordagens está na dificuldade ou até na impossibilidade de um componente dinamicamente distribuído criar novos subcomponentes, ou para um determinado componente continuar se distribuindo. Agentes móveis, no entanto, manipulam estas dificuldades facilmente. Por exemplo, uma aplicação cliente de recuperação de informações manda um código de filtragem para um *proxy site,* que por sua vez manda o código da pesquisa para a localização da base de dados. Em tarefas de recuperação de dados mais complicadas, uma aplicação pode necessitar pesquisar muitas bases de dados seqüencialmente, isto é, a pesquisa sobre uma base de dados depende dos resultados da pesquisa feita sobre uma base de dados anterior. Neste caso, o agente pesquisador pode migrar seqüencialmente através das localizações das bases de dados e o agente *proxy* pode migrar para um novo *proxy site* que está mais perto das base de dados envolvidas na subtarefa atual. Em outras aplicações, o agente *proxy* pode migrar seguindo outros critérios, como por exemplo, latência da rede, largura de banda, carga da máquina atual ou probabilidade que um *link* de rede caia em um futuro próximo. Em todos os casos, um agente pode migrar para uma máquina que não tenha conhecimento prévio da tarefa do agente ou sua aplicação. Isto é particularmente importante para computação móvel, onde o melhor *proxy site* ou réplica da base de dados pode ser completamente diferente, dependendo da localização do computador móvel.

Da mesma forma que no balanceamento de carga, agentes móveis podem suportar as formas mais gerais de distribuição dinâmica, onde uma aplicação pode distribuir seus componentes *"on-the-fly"* (em tempo de execução) para localizações arbitrárias na rede, e estes componentes podem mover-se de um lugar para outro, conforme as condições mudam.

Além disso, como visto em [FENG01], a aplicação pode não somente distribuir seus componentes em tempo de execução como também realizar a substituição de seus componentes *"hot-swapping"* por ocasião de atualização tecnológica, eliminação de *bugs*, além de outras questões envolvidas com o gerenciamento de software. No entanto, é necessário que a aplicação seja modular e que os módulos de software da aplicação estejam preparados para serem substituídos dinamicamente e em tempo de execução.

De fato, embora a verdadeira força dos agentes móveis seja a combinação de suas forças individuais, pode-se argumentar que a distribuição dinâmica flexível é o que faz dos agentes móveis uma escolha efetiva para aplicações distribuídas [GRAY00]. Agentes móveis conservam largura de banda, reduzem latências e tempo total de realização de tarefas, manipulam operações desconectados e balançam a carga distribuindo-se; as vezes continuamente, para localizações mais atraentes na rede.

2.4 Motivando aplicações

Agentes móveis são programas que podem migrar, de máquina para máquina em uma rede heterogênea [GRAY00]. Portanto, os sistemas de agentes móveis têm a propriedade de mobilidade do código mencionada na seção 2.1. Apesar desta característica e de outras mais, além de existirem vários sistemas de agentes móveis, esses sistemas não são muito utilizados em aplicações em produção. Principalmente se for considerado o número de aplicações que utiliza o modelo cliente/servidor.

Há três razões para iss: primeiramente, a mobilidade de agentes é mais utilizada em ambientes de computação móvel, altamente dinâmicos. Estes ambientes estão se tornando populares com o aumento do número e poder de processamento dos dispositivos de computação portáteis. Segundo, a maioria dos sistemas de agentes móveis são protótipos de pesquisa. Por último e mais importante, não há no momento aplicações que requeiram necessariamente a utilização de agentes móveis. Qualquer aplicação específica pode ser realizada tão eficientemente quanto possível com técnicas

tradicionais. Porém, esta abordagem envolve muito mais trabalho administrativo e de implementação do que a solução com agentes móveis.

Quando um conjunto de aplicações é considerado como um grupo, entretanto, a vantagem de agentes móveis torna-se clara. Eles disponibilizam uma estrutura de trabalho simples e geral, na qual um grande número de aplicações distribuídas pode ser implementada facilmente, eficientemente e robustamente [KOTZ99]. Sem agentes móveis, muitas aplicações requerem combinações das mais variadas técnicas tradicionais de implementação, e mais importante, diferentes aplicações requerem diferentes combinações de técnicas.

Uma das aplicações mais comuns para agentes móveis é o processamento de informações distribuídas, particularmente em cenários de computação móvel onde usuários têm dispositivos de computação portáteis com conexões intermitentes e de pouca largura de banda à rede principal. Nesse tipo de aplicação, um agente móvel pode deixar o dispositivo onde encontra-se, mover-se até a localização onde estão as informações desejadas, e então realizar localmente uma tarefa personalizada de recuperação das informações. Somente no final da tarefa, os dados seriam transmitidos ou levados para o dispositivo de onde o agente saiu. Dados iniciais, ou resultados intermediários, não necessitam ser transmitidos. Assim, em alguma situação na qual o *link* com a rede principal do recurso onde encontra-se o agente móvel em determinado momento falhar, o referido agente móvel pode continuar a tarefa de recuperação de modo que quando o *link* tornar-se ativo novamente, ele pode mover-se para outra localização fonte de informações ou retornar à localização original.

Uma derivação para o uso de agentes móveis são as aplicações que realizam recuperação e processamento de informações distribuídas para gerência de serviços de Tecnologia da Informação (TI). Tipicamente, pode-se utilizar agentes móveis para a gerência de configuração, mais especificamente sobre o inventário das organizações. Além, naturalmente de diversas outras aplicações práticas para gerência de desempenho e capacidade, onde os agentes móveis atuariam mais especificamente na coleta e análise

das informações de largura de banda dos *links*, capacidade de roteamento dos roteadores.

Existem ainda no campo da Internet, aplicações de busca de informações totalmente configuráveis pelo usuário. Estas aplicações fariam parte do domínio de aplicações de recuperação de informação.

Encontram-se, ainda, na área militar, aplicações contra terrorismo [GRAY00], onde a atuação básica seria a troca de informações e pesquisas relevantes, entre os combatentes no campo de batalha.

Usando a definição de agentes móveis dada por [GRAY00] é possível que aplicações que atuem na área de serviços avançados de telecomunicações possam usufruir dos benefícios já apontados. Aplicações nessa área, através de seus componentes que atuam como agentes móveis, podem trabalhar no suporte, gerenciamento e bilhetagem de serviços como videoconferência, vídeo por demanda e serviços de conferência de voz, por exemplo. Além disso, similarmente aos serviços de computação móvel, agentes móveis podem dar suporte a usuários móveis de serviços de telecomunicações.

Nos processos de trabalho desenvolvidos pelas corporações, cujas atividades dependem do fluxo de informações, aplicações de *workflow* são fundamentais. Assim, os agentes móveis podem suportar a mobilidade característica dessas atividades, encapsulando a definição e o estado da atividade a ser realizada. Nesse sentido, os agentes móveis suportam tanto a cooperação, através de aplicações de *workflow*, como também podem ser utilizadas plataformas de agentes móveis para desenvolvimento destas aplicações.

Encontra-se em [VIGNA98] descrição de utilização do paradigma Código sob Demanda (COD) através da mobilidade de código em redes ativas. Principalmente na idéia de *programable switch*. Neste caso, o código móvel proporciona extensão aos dispositivos de rede através da ligação dinâmica do código. Um exemplo desta ligação

pode ser um roteador multiprotocolo que pode dinamicamente fazer o *download* do código, ou solicitar um agente móvel, necessário para manipular pacotes cujo protocolo é desconhecido.

No comércio eletrônico também pode-se utilizar agentes móveis. Geralmente este ambiente é composto de várias entidades independentes e possivelmente competidoras. Uma transação envolve negociação com entidades remotas e pode requerer acesso à informações que estão continuamente mudando, como por exemplo cotações de estoque. Neste contexto, os agentes móveis podem ser utilizados para personalizar o comportamento das partes envolvidas no processo de comercialização, negociando protocolos e componentes de aplicação que necessitam mover-se até as informações relevantes em uma transação.

Como já comentado, qualquer aplicação específica pode ser realizada tão eficientemente quanto possível com técnicas tradicionais. Porém, aplicações não comuns requerem combinações de diferentes técnicas tradicionais, e as técnicas que funcionam para uma aplicação podem não funcionar para outra. Agentes móveis no entanto, suportam a distribuição dinâmica e flexível do código para localizações arbitrárias na rede, e assim disponibilizam uma estrutura de trabalho geral em que um grande número de aplicações pode ser implementada facilmente, eficientemente e robustamente. E o mais importante com isso: as aplicações podem exibir características extremamente flexíveis em virtude de condições de mudança na rede, simplesmente realocando um ou mais de seus componentes.

3 Gerência de Redes e Código Móvel

O uso de ferramentas para monitoramento e atuação sobre o conjunto de redes instaladas vem aumentando nos últimos tempos. Isto suscita a discussão, não somente do uso dessas ferramentas, mas também das tecnologias utilizadas por elas.

Basicamente, a gerência de redes conta hoje com duas linhas de gerenciamento baseadas em dois protocolos diferentes. O SNMP (*Simple Network Management Protocol*) proposto pela IETF (*Internet Engineering Task Force*), e o CMIP (*Commom Management Information Protocol*), sendo este último baseado no modelo de gerenciamento OSI (*Open System Interconnection*). Ambos adotam uma arquitetura similar, embora sejam diferentes quanto ao modo e o número de operações. Eles implementam uma gerência centralizada baseada no modelo cliente/servidor. A utilização do SNMP é mais abrangente quando trata-se de gerência de elementos de rede envolvendo equipamentos de conectividade e estações, enquanto a abrangência do CMIP está mais ligada a equipamentos de telecomunicações, onde o padrão OSI é o padrão de fato.

A inteligência do sistema de gerenciamento concentra-se em uma ou mais estações gerenciadoras da rede (*Network Management Station* – NMS), onde estão os gerentes[3] que são responsáveis por interagir com os agentes[4]. Esses agentes de gerência são entidades computacionais com o propósito de prover uma interface padronizada de acesso as informações dos recursos computacionais onde eles estão localizados. Essas informações são organizadas hierarquicamente na forma de uma base de dados chamada de *Management Information Base* (MIB). A comunicação entre os gerentes e os agentes

[3] O termo gerente em gerência de rede trata do *software* que manipula as informações de gerência obtidas do agente.

[4] O termo agente neste contexto não tem relação com o termo agentes móveis ou agentes inteligentes. Significa o *software* que fica "rodando" em uma determinada localização monitorando o estado de variáveis pré-definidas.

é feita através de um protocolo de gerência, que especifica o formato do pacote para cada operação. Daí os diferentes nomes de modelos de gerência.

3.1 Código móvel na gerência de redes

Uma solução que está sendo considerada para a otimização da gerência de redes é a mobilidade de código, uma vez que ela oferece um nível de flexibilidade necessária para lidar com os problemas mencionados nos capítulos anteriores. Essa idéia considera que as funções de gerenciamento devem se deslocar para os dados, ao invés de mover os dados para as funções [GOLDSZMIDT95].

O tratamento dado à gerência de redes neste tópico baseia-se na problemática da centralização das operações de gerenciamento e na introdução de conceitos relativos a utilização de código móvel para estas atividades.

3.1.1 As desvantagens da centralização

Como sabe-se, o processamento das informações relacionadas à gerência de redes é realizada na NMS. Isso gera limitações tanto ao acesso das informações da MIB, pela falta de operações disponíveis dependendo do protocolo, e também com relação a escalabilidade da gerência de redes [YEMINI93].

Assim, a partir do momento que a rede cresce, a NMS tem que interagir com uma quantidade cada vez maior de recursos e dessa forma processar uma quantidade maior de informações, gerando com isso a necessidade de aquisição de *hardware* mais potente para esta tarefa, encarecendo o custo de manutenção do serviço de gerência. Seguindo essa mesma linha de raciocínio, podem surgir problemas com relação ao tráfego e congestionamento dos meios de transmissão.

Considerando-se que se todas as informações de gerência fluem para a NMS é provável que os segmentos de rede aos quais ela está ligada tenham tráfego alto e

portanto sujeitos a colisões. Uma maneira de evitar que o tráfego de dados que não são relativos a gerência sofram atrasos devido ao tráfego de dados de gerência é a construção de uma rede em separado para a gerência da rede de produção. Mais uma vez o problema dos custos, e generalizando-se, seria possível dizer que isso pode gerar recursividade, uma vez que para gerenciar esta última rede seria necessária outra e assim por diante.

Porém, o maior problema ocorre quando a rede apresenta congestionamento. Nesse período, a atividade de gerência deve atuar sobre este problema. No entanto, nesse momento pode-se perceber que:

a) a NMS aumenta o número de interações com os recursos gerenciados para descobrir o motivo do congestionamento, aumentando assim o congestionamento;
b) o acesso a recursos no segmento congestionado é demorado e difícil ou até mesmo impossível;
c) com o congestionamento, os recursos irão apresentar diversas notificações à NMS, aumentando o congestionamento.

3.1.2 Aspectos de descentralização

As desvantagens e problemas da gerência centralizada já foram reconhecidos pela comunidade de gerência de redes, e tanto a IETF quanto a IS0 (*International Standardization Organization*) introduziram aspectos de descentralização nas especificações mais recentes de suas arquiteturas de gerenciamento.

A notificação de eventos foi o primeiro passo, uma vez que possibilitou aos agentes o envio para o(s) gerente(s) sem este(s) ter(em) requisitado, das chamadas *traps* que são mensagens de notificação de determinados eventos. Seguindo a idéia da IETF de manter os agentes simples, as *traps* tratam apenas eventos que requeiram uma "computação leve". Exemplos típicos de *traps* são: modificação no *status* de um *link* (*up* para *down* ou vice-versa), um recurso que tenha sido reinicializado. A proposta da ISO (*International Organization for Standardization*) especifica agentes mais

complexos, capazes de realizar processamentos mais refinados nos dados coletados, gerando assim um esquema de notificação mais elaborado, baseado em um número maior de parâmetros. As notificações no jargão OSI são chamadas de eventos ou alarmes. No entanto, os agentes não têm autonomia para realizar quaisquer operações sobre os recursos, em resposta às notificações que estão emitindo. Dessa forma, o conhecimento da resolução do problema ainda está na NMS onde reside o gerente.

Outra tentativa de descentralização partiu da ISO ao implementar a operação ACTION, onde a NMS pode invocar ações em alguns objetos gerenciados, reduzindo tanto a carga de processamento da NMS, quanto o tráfego associado à atividade de gerência.

Outro conceito relevante à descentralização do processo de gerência de redes é o conceito de *proxy agents* levantado pelo IETF. Enquanto um agente cuida de apenas um recurso gerenciável, um *proxy agent* atua sobre um conjunto de recursos, agindo como cliente em relação aos recursos, e como servidor, ao reportar informações à NMS.

O IETF também propôs outra abordagem ainda mais descentralizada, chamada de *Remote Monitoring* (RMON). O RMON é considerado uma abordagem orientada ao tráfego, uma vez que os monitores ou *probes* RMON monitoram o tráfego de pacotes e analisam seus cabeçalhos, oferecendo com isso informações sobre os *links*, conexões entre as estações, padrão de tráfego e *status* dos nós da rede. A diferença entre as abordagens que trabalham com o monitoramento dos recursos gerenciados consiste em que a conclusão é tirada dos pacotes que trafegam na rede e não da inspeção dos recursos; mesmo porque, parte do processamento das informações de gerência é realizada localmente nos *probes* RMON, conseguindo com isso detecção de falhas, problemas e identificação de situações críticas, como no caso de congestionamento da rede. A comunicação dos agentes RMON com a NMS é feita utilizando-se o protocolo SNMP. Nesta abordagem também tem-se o conceito de notificação, porém de uma forma mais elaborada, semelhante a que ocorre no modelo OSI (CMIP). Para tanto, o agente RMON é configurado para analisar amostras periódicas de parâmetros

relevantes, sempre observando que os limites aceitáveis não sejam ultrapassados. Caso sejam ultrapassados, a notificação é enviada.

Diante das vantagens alcançadas com as tentativas de descentralização descritas, o paradigma de código móvel parece adequado para as aplicações de gerência de redes. Além de benefícios como a compressão semântica e redução de tráfego, obtidas com as propostas da OSI e da IETF, ele oferece autonomia, tolerância a falhas, possibilidade de personalização e provisão de serviços de forma dinâmica.

3.1.3 Gerenciamento por delegação

As pesquisas envolvendo mobilidade de código em aplicações de gerência de redes são geralmente classificadas na área de gerenciamento por delegação *(Management by Delegation*). Esta área surgiu muito antes da popularização dos MCS na Internet. Uma arquitetura de gerenciamento, onde os recursos buscam e atualizam seus *scripts* de gerenciamento de forma dinâmica, é a proposta original [YEMINI91]. Esta arquitetura inclui ainda um protocolo de gerência, agentes e um ambiente de execução para cada recurso. Nesse caso, a estação gerenciadora pode enviar programas contendo uma série de operações aos recursos gerenciados, delegando a eles a execução das tarefas, ao invés de simplesmente trocar mensagens de gerência. Isso libera a estação gerenciadora para executar outras tarefas, além de introduzir um certo grau de paralelismo no sistema de gerenciamento. A atualização do código é fácil e simples, uma vez que o envio de programas aos recursos é dinâmico. Isso deixa o sistema de gerência mais flexível, pois a estação gerenciadora pode personalizar e melhorar dinamicamente os serviços oferecidos pelos agentes em cada recurso.

No gerenciamento por delegação, a migração de código é sempre realizada pela estação gerenciadora, não existindo suporte para a mobilidade autônoma dos *scripts.*

3.1.4 Código sob demanda: flexibilidade

Segundo a IETF (SNMP), a estrutura da MIB é implementada a partir do código do agente e não pode ser alterada durante a execução (*runtime*). Esta falta de flexibilidade traz algumas restrições ao processo de notificação de eventos. A definição de eventos significativos está atrelada à estrutura da MIB, não permitindo que novos eventos personalizados possam ser definidos. A criação de agentes estendidos, capazes de analisar arbitrariamente os eventos e de reagirem a estes sem a intervenção da estação gerenciadora seria de extrema utilidade para a gerência de redes.

Utilizando-se de alguns artifícios do paradigma cliente-servidor, pode-se estender um agente de gerência. As funcionalidades do CORBA podem ser aproveitadas pelos agentes de gerência para reforçar propriedades de segurança e tolerância a falhas. Através de uma especificação mais refinada, a ISO permite que a MIT (*Management Information Tree*) seja alterada dinamicamente.

Essa extensão dos agentes representa um aumento de complexidade que nem sempre compensa. Na maioria das vezes, a adição de complexidade é feita de forma estática, incluindo novas funcionalidades em um grupo de agentes, ainda que a maior parte deles jamais necessite usufruir destas funções. Na verdade, a própria natureza dinâmica da gerência de redes requer agentes customizados dinamicamente.

Além do mais, o aumento de complexidade nos agentes sempre foi evitado pela comunidade de gerência de redes. O argumento é de que os agentes devem ser o mais simples possível para não exigir muitos recursos dos equipamentos gerenciados. Apesar do poder computacional dos equipamentos hoje não ser mais um problema determinante, agentes que realizam operações complexas sem necessidade tornam o processo de gerência mais lento, podendo ainda gerar mais tráfego na rede.

A arquitetura de Código sob Demanda pode aumentar a flexibilidade e combinar as vantagens da IETF e da ISO nas aplicações de gerência de redes, mantendo os agentes pequenos e simples. Quando houver necessidade de modificar a estrutura das

informações de cada recurso, como por exemplo alterar o critério de detecção de um evento para emitir um *trap*, o código do agente pode ser automaticamente atualizado com as novas instruções a executar. Desta forma, os agentes são "reconstruídos" sem a necessidade de recompilação e reinstalação. Se a atualização de código dos agentes for realmente autônoma, não é necessária uma intervenção direta da pessoa responsável pela rede, a não ser quando uma nova funcionalidade precisar ser codificada no servidor de código.

3.1.5 Flexibilidade com o uso da extensão

Uma proposta para extensão de agentes, [RFC1228], considera o uso de subagentes (móveis ou não), que entrariam no sistema de gerência, atuando sobre variáveis cuja MIB associada ao agente não abrange. Esse subagente comunica-se com o agente através de um protocolo chamado DPI (*Distributed Protocol Interface*). Daí o comumente chamado protocolo SNMP-DPI. Naturalmente, este subagente tem associado uma MIB que será reconhecida pela NMS através do registro desta MIB pelo agente que está sendo estendido. Por sua vez, o agente que está sendo estendido sabe da existência desta MIB através da comunicação estabelecida com o subagente através do protocolo DPI, utilizando-se da primitiva REGISTER. Logo, é necessária modificação nos agentes legados tornando-os disponíveis à extensão com o uso do DPI.

O agente deve se responsabilizar por receber as requisições do gerente sobre as variáveis que estão na MIB, cujo subagente controla [PAGUREK00]. Assim, um agente SNMP habilitado a utilizar o protocolo DPI é um agente padrão, modificado para reconhecer requisições inicializadas por subagentes ou por gerentes. Na Figura 3.1 é possível observar a arquitetura funcional de um agente com suporte a extensão pelo protocolo DPI e também pelo RDPI.

O protocolo RDPI (*Reverse Distributed Protocol Interface*) [PAGUREK00] foi projetado para reverter a direção de requisições e respostas, definindo um método de acesso à MIB do agente SNMP, sem a necessidade de criação de mensagens SNMP

completas no formato de PDU (*Protocol Data Unit*) SNMP ou codificação dos dados no formato ASN.1 BER (*Abstract Syntax Number One - Basic Encoding Rules*). Assim, o propósito deste protocolo é dar, à um subagente registrado (seja ele móvel ou não), acesso à MIB de um agente SNMP com suporte a RDPI. Naturalmente este subagente deve utilizar RDPI.

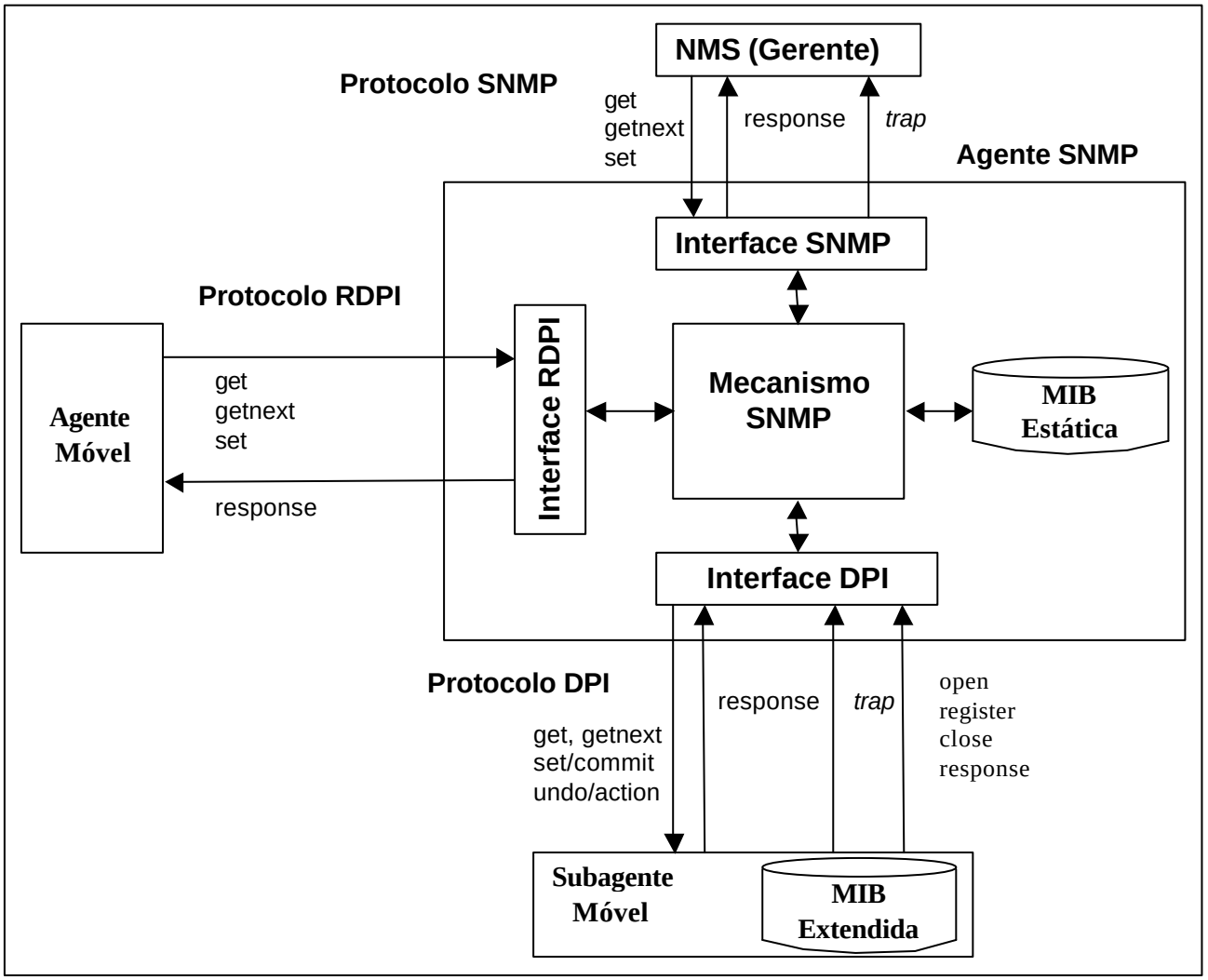


Figura 3.1 – Arquitetura funcional de um agente SNMP-DPI-RDPI

De qualquer maneira, para o gerente, esses subagentes são invisíveis. Apenas a MIB estendida por eles pode ser visualizada.

Desta maneira, percebe-se que a implementação deste protocolo é voltada para a utilização de agentes SNMP e agentes móveis em conjunto, ou seja na mesma solução

de gerência de redes. Assim pode-se continuar utilizando sistemas legados de gerência de redes, baseados no modelo agente-gerente (com uma pequena alteração no agente para suportar DPI e RDPI), aproveitando e estendendo-os com MIB's que forem necessárias.

Portanto, a utilização de formas de extensão aos agentes também contribui para uma maior flexibilidade dos sistemas de gerência.

3.1.6 Remote evaluation

O gerenciamento tradicional utiliza a transferência dos dados dos agentes para o gerente (estação gerenciadora), sendo estes dados processados lá. Isso auxilia na manutenção dos agentes compactos e simples, deixando o código pesado para a NMS executar.

A largura de banda consumida pelo tráfego de dados, por exemplo, a transferência de uma grande tabela da MIB, torna-se um problema no modelo tradicional. Por outro lado, a implementação de funções de compressão semântica de dados - para suprimir o envio de toda a tabela quando na verdade busca-se apenas um valor, por exemplo no agente, deixando-o maior e mais pesado, pode não compensar devido a pouca utilização desse tipo de função.

Assim, o paradigma REV (*Remote Evaluation*) pode ser utilizado na solução deste problema. Para o caso citado, é possível implementar uma função com as primitivas de gerência necessárias para a busca do valor, e, uma vez pronta , a função é enviada para o recurso desejado para que seja executada localmente. Somente o resultado da função é retornado depois da execução, economizando assim largura de banda proporcional ao tamanho dos dados que seriam enviados caso não fosse utilizado este tipo de solução, e das funções que deveriam estar implementadas no agente. Além da compressão semântica, parte do processamento é distribuído entre os agentes, liberando a NMS.

4 Sistemas de Agentes Móveis

Neste capítulo, será apresentada uma classificação de alguns dos sistemas (plataformas) de agentes móveis mais significativos, com as principais características e aplicações de cada um deles.

Os sistemas de agentes móveis podem ser categorizados de diversas formas. Uma dessas formas, e a que será adotada, é a classificação pela linguagem de programação que eles suportam.

Nesse sentido, alguns sistemas de agentes móveis permitem que agentes sejam escritos em várias linguagens; outros somente na linguagem Java, que é a mais popular das linguagens para agentes móveis, e outros ainda permitem que agentes móveis sejam escritos em uma outra linguagem de programação única que não seja Java.

4.1 Sistemas multilinguagens

Uns poucos sistemas tentam suportar agentes móveis fora do contexto de uma linguagem de programação específica. Em seguida, serão apresentados alguns desses sistemas que permitem a utilização de mais de uma linguagem para o desenvolvimento dos agentes móveis.

4.1.1 ARA

O sistema ARA [PEINE97] suporta agentes escritos em TCL, Java e C/C++. Os agentes escritos em C/C++ são compilados em um *bytecode* interpretado mais eficiente chamado MACE. Para todas as três linguagens, TCL, Java e MACE, é disponibilizada a instrução ***go*** que automaticamente captura o estado completo do agente, transfere o estado para a máquina destino e reinicia a execução do agente no exato ponto da

instrução *go*. Diferente de outros sistemas de multi-linguagens, o sistema completo ARA é *multi-thread*; o servidor de agentes e os interpretadores TCL, Java e MACE são executados em um único processo UNIX. Apesar desta abordagem complicar a implementação, ela tem vantagens significativas de desempenho, uma vez que o *overhead* de comunicação e de inicialização do interpretador é pequeno. Assim, quando um novo agente chega, simplesmente é iniciada a execução em uma nova *thread*, e quando um agente quer se comunicar com outro, simplesmente é transferida a estrutura da mensagem para o agente destino, evitando o uso de comunicação interprocesso. Quase todos os sistemas que suportam apenas Java são também *multi-thread* e têm as mesmas vantagens de desempenho.

4.1.2 D'Agents

D'Agents também já foi conhecido como Agent TCL [GRAY95]. Suporta agentes escritos em TCL, Java e Scheme, bem como agentes estacionários escritos em C e C++. Como o sistema ARA, D'Agents também disponibiliza uma instrução ***go*** para cada linguagem e automaticamente captura e restaura o estado completo de um agente em migração. Ao contrário do sistema ARA, somente o servidor D'Agents é *multi-thread.* Assim, cada agente é executado em um processo separado, o que simplifica a implementação consideravelmente, porém adiciona o *overhead* da comunicação interprocesso. O servidor D'Agents usa criptografia de chave pública para autenticar a identidade do proprietário do agente que chega; utiliza também agentes estacionários gerenciadores de recursos, que associam ao agente direitos de acesso baseados na autenticação e preferências do proprietário; e também módulos de aplicação específicos à linguagem que testam os direitos de acesso, prevenindo que uma violação ocorra (por exemplo, acesso a um *filesystem*) ou terminando o agente quando uma violação ocorrer (por exemplo, tempo total de CPU excedido).

Cada gerenciador de recurso é associado com um recurso específico, como por exemplo, um *filesystem*. Os gerenciadores de recursos podem ser tão complexos quanto desejado, mas os gerenciadores *default* simplesmente associam uma lista de direitos de

acesso à cada proprietário de agente. Diferente do sistema ARA, no D'Agents muitos gerenciadores de recursos não são consultados quando o agente chega, mas somente quando o agente tenta acessar o recurso correspondente ou quando é explicitamente requisitado um direito de acesso específico.

4.1.3 Tacoma

Tacoma [JOHANSEN98] suporta agentes escritos em C, C++, ML, Perl, Python e várias outras linguagens. Ao contrário dos sistemas ARA e D'Agents, o sistema Tacoma não disponibiliza facilidades de captura automática de estado dos agentes; ao contrário, quando um agente quer migrar para uma nova máquina, ele cria um *folder* dentro do qual é empacotado o código e quaisquer informações de estado desejadas. O *folder* é mandado para a nova máquina, que inicializa o ambiente de execução necessário e então chama um ponto de entrada conhecido dentro do código do agente para reiniciar a execução do agente. Apesar desta abordagem onerar o programador de agentes móveis pela captura do estado de execução, ela permite a integração de novas linguagens de programação ao sistema Tacoma, fazendo com que os interpretadores e as máquinas virtuais existentes possam ser usados sem modificações.

As versões públicas do Tacoma contam com o suporte do sistema operacional para disponibilizar aspectos de segurança, porém deixam alternativas para que sejam adicionados mecanismos de autenticação com a utilização de criptografia. Com a adição destes mecanismos, é possível rejeitar agentes não confiáveis.

4.2 Sistemas baseados em Java

Java é uma linguagem popular para o desenvolvimento de aplicações distribuídas. A implementação desta linguagem como uma máquina virtual permite que os programas em Java sejam executados em uma grande quantidade de sistemas operacionais. Além disso, ela tem suporte para serialização, segurança, invocação de

métodos remotos e carregamento dinâmico de classes. Todos esses fatores fazem da linguagem Java uma escolha natural para agentes móveis. Vários sistemas têm sido desenvolvidos tendo em mente especificamente a linguagem Java. Nesse sentido, na seqüência serão apresentados alguns sistemas que a utilizam como única linguagem de desenvolvimento.

4.2.1 Aglets

Aglets [AGLETS] foi um dos primeiros sistemas baseados em Java. Como todos os sistemas comerciais, incluindo Concordia, Jumping Beans, e Voyager, o sistema Aglets não captura o estado de execução (*thread*) durante a migração, pois a captura do estado de execução requer modificações na máquina virtual Java padrão. Assim, ao invés de disponibilizar a primitiva ***go*** do D'Agents e ARA, o sistema Aglets e outros sistemas comerciais usam variantes do modelo do sistema Tacoma, onde a execução do agente é reinicializada em um ponto de entrada conhecido depois de cada migração. Especificamente Aglets utiliza um modelo orientado a eventos. Quando um agente quer migrar, ele chama o método *dispatch*. O sistema Aglets chama então o método *onDispatching*, que realiza atividades específicas da aplicação, "mata" as *threads* do agente, serializa o código do agente e o estado dos objetos e manda o código e o estado dos objetos para a nova máquina. Na nova máquina, o sistema chama o método *onArrival*, que realiza inicialização específica da aplicação, e então chama o método ***run*** para reiniciar a execução do agente.

Aglets inclui um mecanismo simples de persistência, que permite que um agente escreva seu código e estado dos objetos para um dispositivo de armazenamento secundário e temporariamente desative-se. Também inclui serviço de *proxies* que atuam como representantes para o sistema Aglets, e entre outras coisas, disponibiliza transparência de localização. Além disso, é disponibilizado também o serviço de *lookup* para encontrar agentes em movimento e um conjunto de características de passagem de mensagens para comunicação interagentes. O modelo de segurança do Aglets é similar a ao D'Agents e ao ARA, e outros modelos de segurança de outros sistemas de agentes

móveis baseados em Java. Cada aglet tem um proprietário e um usuário. Quando o agente entra em um contexto (isto é, em um lugar virtual) em uma máquina particular, o contexto associa um conjunto de permissões para o agente, baseado na autenticação do proprietário e do utilizador. Essas permissões são reforçadas com os mecanismos de segurança padrão do Java, como o gerenciador de segurança personalizado.

4.2.2 Concordia

Concordia [WONG97] é um sistema de agentes móveis baseado em Java que tem o foco fortemente voltado para a segurança e a confiabilidade. Como outros sistemas baseados em Java, ele move de uma máquina a outra o código e o estado dos objetos do agente, mas não o estado de execução. Como muitos outros sistemas, os agentes são empacotados com um itinerário de "lugares" para visitar, que pode ser ajustado pelo agente durante sua peregrinação. Agentes, eventos e mensagens podem ser enfileirados se o "lugar" remoto não está disponível em determinado momento. Agentes são cuidadosamente salvos em um meio de armazenamento persistente, antes de deixar uma máquina e depois de chegar em uma nova, para evitar a perda de agentes no caso de uma pane na(s) máquina(s). Os agentes são protegidos através de criptografia enquanto estão sendo transmitidos ou no disco e as máquinas são protegidas de agentes maliciosos através de autenticação criptográfica do proprietário do agente e de listas de controle de acesso que guardam cada recurso.

4.2.3 Jumping Beans

Nesse sistema, os agentes móveis visitam "agências" *Jumping Beans* [JB98] em computadores associados a algum "domínio" *Jumping Beans*. Cada domínio tem um servidor central, que autentica as agências que se associam ao domínio. Assim, os agentes movem-se de agência em agência dentro do domínio, e agentes podem enviar mensagens para outros agentes no domínio. Ambos os mecanismos são implementados por passagem pelo servidor de domínio. Assim, este servidor torna-se um ponto central para armazenamento de informações de histórico, para gerenciamento e autenticação

dos agentes. Ele também torna-se um ponto central de falhas ou um gargalo de desempenho. Existe um planejamento para o desenvolvimento de servidores escaláveis que executarão em máquinas paralelas. Outra abordagem para a escalabilidade é a criação de vários pequenos domínios, cada qual com seu próprio servidor.

A segurança e confiabilidade são aspectos chave do sistema *Jumping Beans.* A criptografia de chave pública é utilizada para autenticar agências no servidor e listas de controle de acesso para controlar o acesso aos recursos por parte do agente, baseadas nas permissões dadas ao proprietário do agente.

Este sistema, como no Aglets, requer que tanto o agente quanto os objetos que são atributos do agente, sejam objetos serializáveis, o que significa que o mecanismo de mobilidade de código utilizado é o fraco; comum entre os sistemas de agente móvel baseados em Java.

4.2.4 Voyager

Voyager [OBJ97] é um ambiente de agentes móveis baseado em Java, integrado com CORBA. Ele disponibiliza um modo conveniente para interagir, de certa forma transparentemente, com objetos remotos (via objetos *proxies* chamados referências virtuais) e com objetos móveis. Quando um objeto se move, ele deixa um objeto repassador que redireciona para a nova localização quaisquer mensagens que cheguem na antiga localização. Objetos “agentes”, ao contrário de outros objetos, podem mover-se autonomamente aplicando o método *moveTo()* neles mesmos. Voyager move o código e os dados do agente, mas não o estado de execução, para a nova localização e invoca lá[5],o método desejado. Voyager também permite que objetos e agentes sejam persistentes através de chamada explícita ao método *saveNow()*. Voyager também suporta comunicação em grupo (*multicast*), e disponibiliza um serviço de diretório compartilhado. Voyager disponibiliza os mesmos mecanismos de segurança básicos como outros sistemas baseados em Java.

4.2.5 Mole

Foi a primeira plataforma a utilizar a linguagem Java [MOLE01]. Foi desenvolvida pelo grupo de Sistemas Distribuídos da Universidade de Stuttgart.

Nela, dois tipos de agentes são disponibilizados: agentes de sistema e agentes de usuário. Agentes de sistema não podem migrar. Estes agentes são normalmente interfaces para recursos fora da plataforma. Estes agentes têm mais direitos que agentes de usuário. Os agentes de usuário só podem ter acesso aos recursos disponibilizados pela plataforma.

Mole também implementa a mobilidade fraca, sendo a migração implementada pela serialização de objetos do Java. Assim, depois que um *thread* do agente chama o método *migrateTo()*, todos os *threads* pertencentes ao agente são suspensos. A partir deste momento, não são mais aceitas novas mensagens ou chamadas remotas. Após a entrega ao agente de todas as mensagens pendentes, o agente é removido da lista de agentes ativos. É feita então a serialização do agente. No destino, ocorre a instanciação do código serializado e um novo *thread* é iniciado. Logo que esse *thread* assume o controle do agente, uma mensagem de sucesso é enviada à fonte que finaliza todos os *threads* pertencentes ao agente, removendo-o do sistema. Se ocorrer um erro em qualquer fase da migração, esta é interrompida e os *threads* do agente na fonte voltam a ser executados.

[5] Aglets chama o mesmo método a cada movimentação, enquanto *Jumping Beans*, Concordia e Voyager permitem que o agente especifique um método diferente.

4.3 Outros sistemas

Ainda, há um conjunto de sistemas cujo foco está na utilização de outra linguagem de programação única que não é a linguagem Java. Assim, esses sistemas desenvolveram cada um uma nova linguagem.

4.3.1 Messengers

O projeto Messenger [MUHUGUSA98] usa código móvel para construir sistemas distribuídos flexíveis, não especificamente sistemas de agentes móveis. Neste sistema, os computadores executam um Sistema Operacional Messenger (MOS), que disponibiliza alguns poucos serviços. Deste modo, o MOS pode enviar e receber *messengers*, que são pequenos pacotes de dados e código escritos na linguagem de programação própria do Messengers ($M_ø$). O MOS pode interpretar programas $M_ø$, que podem acessar um dos dois serviços: dicionário global, que permite troca de dados entre *messengers*, e o serviço dicionário, que é uma lista pesquisável de *messengers* que oferecem seus serviços para outros *messengers.*

4.3.2 Obliq

Obliq [BROWN97] é uma linguagem orientada a objeto, não tipada e interpretada, que suporta computação distribuída. Um objeto Obliq é uma coleção de campos que contém métodos, aliases e valores. Um objeto pode ser criado em uma localização remota, clonado, ou migrado com uma combinação de clonagem e reencaminhamento. A implementação de agentes móveis com este sistema é direta. Um agente consiste em um procedimento definido pelo usuário, que toma como argumento objetos Obliq, que o procedimento utilizará para realizar a tarefa. A migração ocorre quando o procedimento e seus argumentos são enviados para o destino. No destino, o procedimento é invocado para reinicializar a execução do agente. Obliq não atua sobre características de segurança. Obliq disponibiliza controle de acesso, quer dizer, verificação de acesso

associada com todos os comandos considerados “perigosos”. Porém, Obliq não tem mecanismos de autenticação ou criptografia. Tipicamente, a verificação de acesso simplesmente pergunta ao usuário se o agente pode executar determinada ação ou não.

4.3.3 Telescript

Telescript [WHITE97] foi o primeiro sistema de agentes móveis comercial. No Telescript, cada nó da rede executa um servidor que mantém um ou mais “lugares” virtuais. Um agente que chega, especifica que lugar ele deseja entrar. O lugar autentica a identidade do proprietário do agente, examinando as credenciais criptografadas, e então, associa ao agente um conjunto de direitos ou permissão (*permits*) de acesso. Um *permit*, por exemplo, pode especificar um tempo máximo de vida para o agente, enquanto outro pode especificar uma quantidade máxima de uso de espaço em disco. Um agente que tenta violar seus *permits* é encerrado imediatamente. Assim, para a manutenção do lugar e para garantir as restrições de segurança, o servidor continuamente guarda, em dispositivo de memória não volátil, o estado interno dos agentes que estão sendo executados, para que seja possível recuperá-los em caso de falha daquele nó da rede.

Um agente Telescript é escrito em uma linguagem imperativa, orientada a objeto, que é similar a Java e C++ e é compilada para *bytecodes* para uma máquina virtual que faz parte de cada servidor. Como no D’Agents, um agente Telescript migra para o destino com a instrução ***go***, que captura dele o código, dados e estado de execução. No destino, o agente continua a execução na instrução imediatamente após a instrução ***go***. Nesse sistema, os agentes podem comunicar-se de dois modos: 1) os dois encontram-se no mesmo lugar. Neste caso, ambos recebem referências de seus objetos e podem invocar os métodos um do outro; e 2) um agente pode conectar-se a um objeto em um lugar diferente – os dois agentes passam objetos através da conexão. O sistema Telescript é considerado um dos mais seguros, tolerantes a falhas e eficientes sistemas de agentes móveis, tendo sido superado no mercado devido a rápida difusão da linguagem Java.

4.4 Similaridades e diferenças

Todos os sistemas de agentes móveis têm a mesma arquitetura geral: um servidor em cada máquina que aceita agentes que chegam e, para cada agente, inicializam um ambiente de execução apropriado, carregando as informações de estado dos agentes e inicializando a sua execução. Alguns sistemas, como os sistemas que utilizam somente a linguagem Java, têm servidores *multi-thread* e executam cada agente em uma *thread* separada; outros sistemas têm servidores multiprocesso e executam cada agente em um processo separado; e os sistemas que restam, utilizam alguma combinação dos dois extremos. D'Agents, por exemplo, tem um servidor *multi-thread* para aumentar a eficiência, mas processos interpretadores separados para simplificar a implementação. Jumping Beans é um caso particular onde, devido a sua arquitetura centralizada, os agentes devem passar por um servidor central ao migrar de uma máquina para outra. Apesar deste servidor poder tornar-se um gargalo de desempenho, ele simplifica as questões de segurança, manutenção de histórico, administração e outras características, possivelmente aumentando a aceitação inicial no mercado.

Os sistemas de agentes móveis atuais são orientados à proteção individual da máquina que os executam, contra agentes móveis maliciosos. Para tanto, o mecanismo mais utilizado é a assinatura digital do agente pelo proprietário. Quando do recebimento do agente, o servidor de agentes verifica a assinatura, aceitando ou rejeitando o agente baseado nesta assinatura. Em caso de aceitação, são associados ao agente restrições de acesso, baseadas também na assinatura digital e no histórico de migração daquele agente, executando-o em um ambiente de execução que garanta estas restrições.

Existem algumas diferenças entre os sistemas de agentes móveis mais comuns descritos aqui. Entre elas, a granularidade de seus mecanismos de comunicação, podendo interagir com CORBA ou não, e se ainda os agentes estão dentro dos padrões de agentes móveis MASIF (*Mobile Agent System Interoperability Facility*) [MILOJICIC99]. Fora estas diferenças, entretanto, todos os sistemas de agentes móveis

discutidos acima, com exceção de *Messengers*, foram projetados para aplicações de *workflow*, gerenciamento de rede e instalação automatizada de software. Todos os sistemas são ideais para aplicações de recuperação de informações e a decisão de qual deles utilizar deve ser baseada na linguagem de implementação, no nível de segurança e desempenho desejados.

No caso específico deste trabalho, foi escolhido o sistema Aglets em sua versão 1.1.0. Além das características mencionadas acima, foram utilizadas as características de custo e disponibilidade do código fonte. Assim, tendo código fonte aberto, sendo gratuita e utilizando a linguagem Java, bem como não havendo necessidade de persistência e grandes esquemas de segurança para os propósitos desse trabalho, optou-se por esta plataforma.

5 Os Experimentos

Como já mencionado, o uso do protocolo SNMP é um dos modelos de gerência de rede existentes. Esse modelo conhecido como agente-gerente é largamente utilizado. Com funções de recuperação de informações e atuação sobre os objetos gerenciáveis, ele utiliza a comunicação assíncrona entre os componentes do modelo. Todas essas peculiaridades acabam fomentando a necessidade de validação de seu uso quando novas tecnologias surgem.

Atualmente, não existem equipamentos de conectividade que disponibilizem a execução de uma plataforma de agentes móveis de forma nativa, embora haja proposta que se assemelhe [BARAS01]. Por esta razão, a realização do experimento com agentes móveis utilizou estações de trabalho que permitiram a execução da plataforma de agentes móveis Aglets1.1.0 [AGLETS] através do sistema operacional Windows 98® e da máquina virtual java na versão 1.1.8.

Desta forma, neste trabalho, a avaliação do desempenho entre agentes móveis e requisições SNMP executadas por aplicativo de acordo com o modelo cliente/servidor, por meio da métrica tempo de resposta, empregou técnicas de medidas na obtenção de valores de 3 variáveis da MIB II, tendo em vista [COSTA99], que são *ifInErrors*, *sysDescr* e *sysContact*.

Assim, com o intuito de avaliar o desempenho da utilização do modelo de agentes móveis na gerência de redes através das medidas da métrica tempo de resposta em uma aplicação particular, foi desenvolvido, neste trabalho, um agente móvel na linguagem de programação Java para a plataforma Aglets1.1.0, que obtém as informações desejadas da MIB II através da mobilidade fraca e execução de requisições SNMP GET. Esse agente tem o tamanho inicial de 4,08 KB. O tamanho das ferramentas envolvidas com a obtenção da métrica tempo de resposta, foi obtido através das informações de tamanho disponibilizadas pelo sistema operacional utilizado.

Da mesma forma, foi desenvolvido outro agente móvel para a mesma plataforma, buscando informações não disponibilizadas na MIB II, através da JNI. Essas informações são a quantidade total e disponível de memória volátil (RAM), quantidade total e disponível de memória virtual e espaço disponível em disco. Este agente tem o tamanho inicial de 4,41 KB.

A comparação entre a métrica obtida com as diferentes abordagens de agentes móveis poderá, relativamente, mostrar qual das tecnologias de recuperação de informação utilizadas com agentes móveis, melhor se adapta aos propósitos da recuperação de informações para o gerenciamento de redes.

Já a comparação entre os tempos de resposta do aplicativo cliente/servidor baseado em RPC e, os agentes móveis, busca subsidiar qual é o melhor modelo de recuperação de informações de gerenciamento de redes de computadores, dentre os experimentados.

Para tanto, basicamente será colocado lado a lado o paradigma de agentes móveis na gerência de redes e o já conhecido modelo SNMP, enfatizando o tempo total para a realização de determinada tarefa dentro de um conjunto de equipamentos.

Como já mencionado na literatura, os maiores problemas de gerenciamento encontram-se quando da existência de gargalo (*link* lento) de comunicação entre a estação gerenciadora (NMS) e a (sub)rede gerenciada. Desta forma, para fazer as comparações foi executada a avaliação do tempo de resposta para os experimentos com agentes móveis e com SNMP em dois ambientes diferentes. O primeiro foi um ambiente de rede local. Neste caso (Figura 5.1) a estação gerenciadora estava no mesmo segmento que a rede local, tanto quanto os equipamentos de conectividade e as estações de trabalho com as plataformas de agentes móveis.

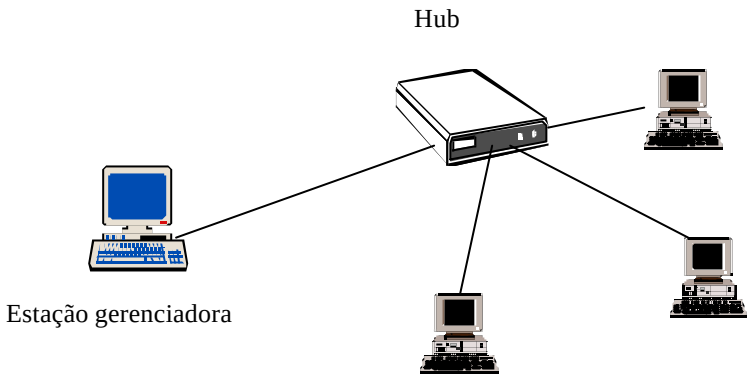


Figura 5.1 – Ambiente de teste em rede local (Ambiente 1)

No segundo caso, conforme Figura 5.2, têm-se a estação gerenciadora separada da rede a ser gerenciada por um *link* de comunicação que é lento. Para essa configuração atribui-se a denominação de rede remota. Assim os experimentos realizados com essa configuração foram chamados de experimentos em rede remota.

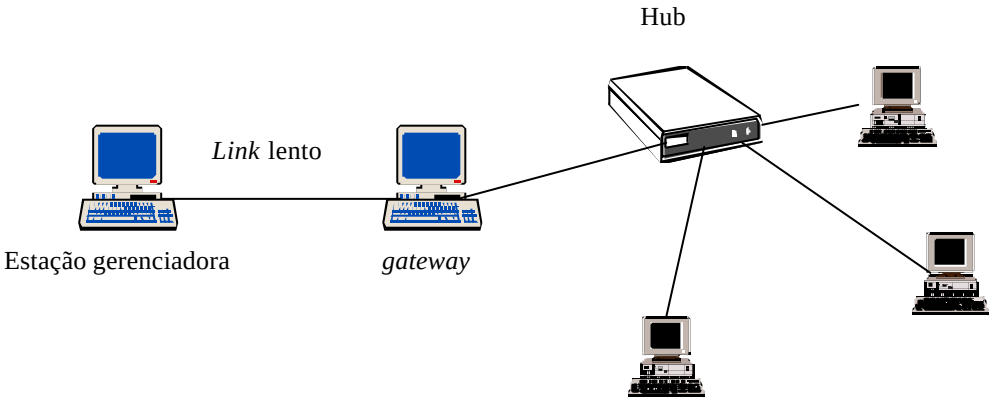


Figura 5.2 – Ambiente de teste em rede remota (Ambiente 2)

5.1 O ambiente de desenvolvimento

Foi utilizada a estrutura do laboratório de computação do Instituto de Tecnologia e Automação Industrial (ITAI) da Universidade do Oeste do Paraná (Unioeste) do município de Foz do Iguaçu – Paraná. A estrutura utilizada para a realização dos experimentos usou 5 microcomputadores Pentium II 400 Mhz, com 64 MB de memória RAM SDRAM, e com capacidade de armazenamento em disco idêntica entre eles.

Essa estrutura foi organizada de forma a disponibilizar a realização de experimentos com ambiente de rede local e rede remota, conforme as Figuras 5.1 e 5.2.

No caso dos experimentos envolvendo rede remota, essa estrutura foi organizada de forma a prover a atuação da NMS em uma ponta do gargalo de comunicação, tendo na outra, a (sub)rede a ser gerenciada. Para tanto foi utilizado em dois microcomputadores o sistema operacional Windows NT® na versão Enterprise 4.0. Ambos com o serviço de acesso remoto configurado e ativo. Um deles atuou como NMS e o outro como *gateway* da (sub)rede gerenciada. Essas duas máquinas foram conectadas através de cabo serial com a utilização do serviço de acesso remoto. A conexão lógica entre as duas máquinas foi realizada via *dial up*, com a utilização de TCP/IP (*Transport Control Protocol/Internet Protocol*) sobre *frames* PPP (*Point to Point Protocol*), sendo que o servidor *dial* nesse caso atuou como *gateway*, disponibilizando acesso à rede remota para a NMS que foi o cliente *dial*.

A utilização da porta serial como dispositivo de comunicação (*null modem*) foi utilizada pela disponibilidade e facilidade de configuração da velocidade, atuando como gargalo de comunicação de maior ou menor intensidade, possibilitando a realização dos experimentos com três diferentes velocidades: 38400, 57200 e 115200bps. Apesar de estas velocidades não estarem sendo mais utilizadas como velocidade de *link* total em uma WAN (*Wide Area Network*), elas ainda fazem parte de componentes do sistema de interligação de redes WAN através de *links* separados que juntos formam um enlace único.

A Figura 5.3 retrata a estrutura do “gargalo” utilizada nos experimentos com rede remota.

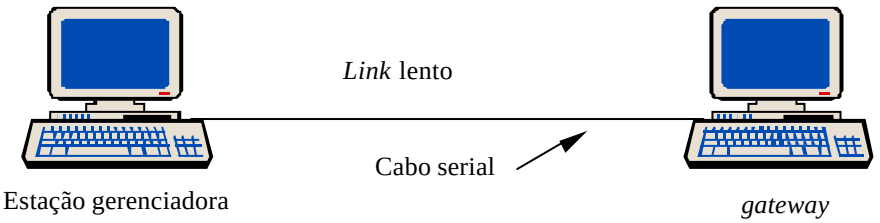


Figura 5.3 – Gargalo de comunicação

A plataforma Aglets1.1.0 com o servidor Tahiti, juntamente com o jdk 1.1.8, foi utilizada tanto na NMS quanto nas estações gerenciadas. Assim, os agentes desenvolvidos eram inicializados na NMS, migravam para a primeira estação, seguindo então um itinerário contendo tantas estações quanto solicitadas na inicialização. Desta forma, a alternativa encontrada para solucionar o problema do número de estações necessárias para o experimento (pois fisicamente estavam disponíveis apenas três), foi repetir o itinerário tantas vezes quanto necessário, simulando o número de estações desejadas, o que chamou-se de escalabilidade dos experimentos. No caso, o número máximo de estações utilizado para os experimentos foi de 252. As Figuras 5.4 e 5.5 ilustram a migração dos agentes móveis, tanto para o experimento em rede local quanto para a rede remota.

As demais estações executaram o sistema operacional Windows 98®, em uma estrutura de rede local Ethernet a 10Mbps. Quando do experimento com rede local, a NMS passou a integrar a mesma estrutura e não houve mais a necessidade do *gateway*.

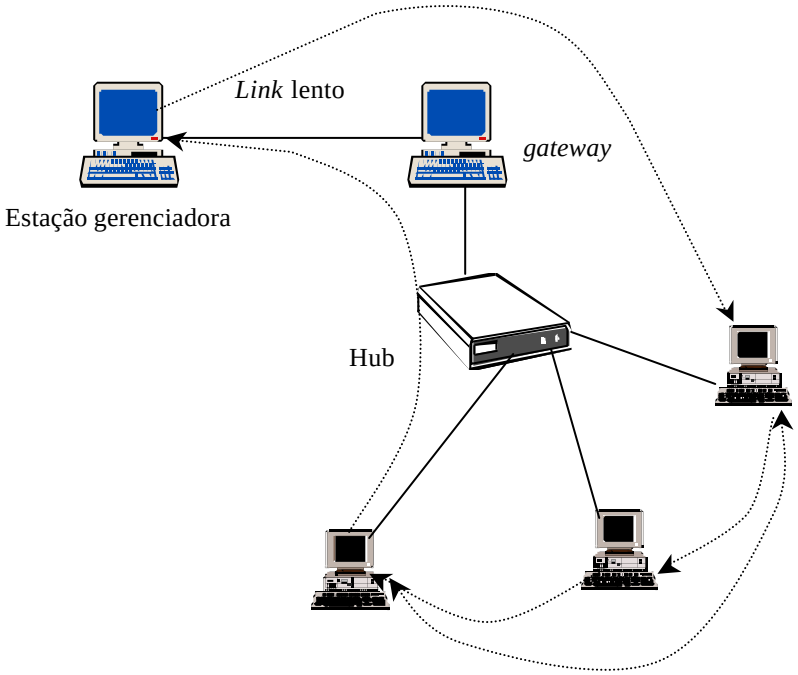


Figura 5.4 – Migração dos agentes móveis em rede remota

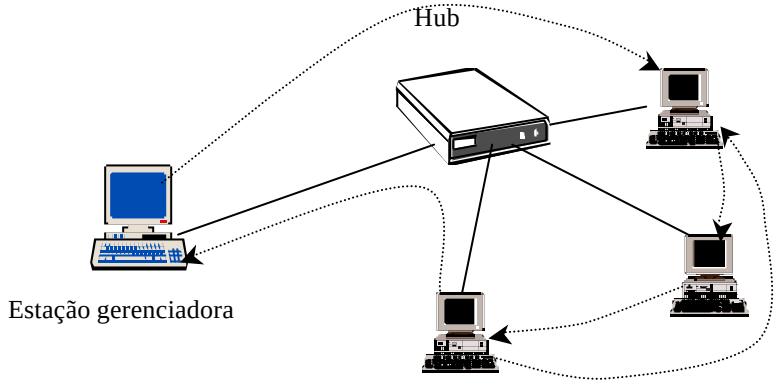


Figura 5.5 – Migração dos agentes móveis em rede local

Em ambos os ambientes de teste, para que ocorra a recuperação das informações de gerência, para o experimento com agente móvel com suporte a SNMP, há a interação entre o agente móvel e o agente SNMP estático. Processo similar ocorre para o caso do agente com suporte a JNI. As figuras 5.6 e 5.7 ilustram essa interação para recuperação das informações.

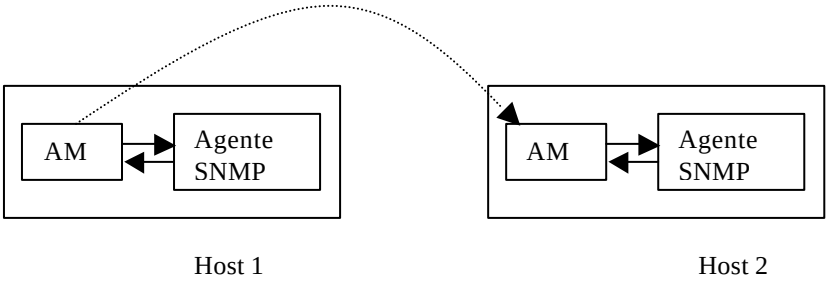


Figura 5.6 – Recuperação das informações pelo agente móvel com suporte a SNMP

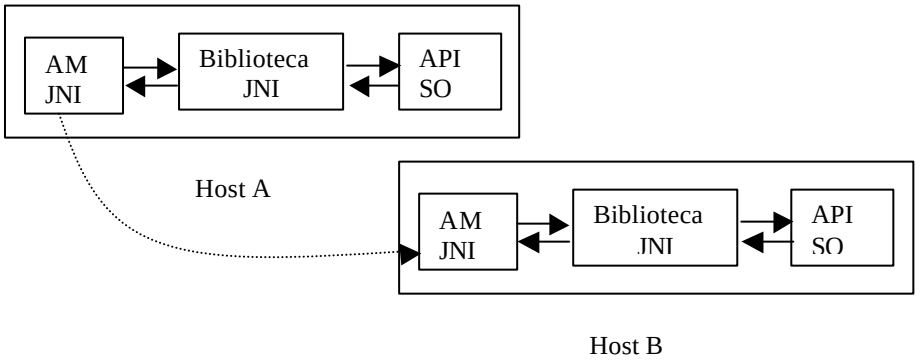


Figura 5.7 – Recuperação das informações pelo agente móvel com suporte a JNI

Para o caso dos experimentos com o aplicativo SNMP, em ambos os ambientes de teste, a ferramenta utilizada para obtenção dos valores de tempo de resposta foi desenvolvida na linguagem Java. Essa ferramenta utiliza classes de objetos disponibilizadas pela ferramenta de gerenciamento AdventNetSNMPv3, e do *kit* de desenvolvimento java em sua versão 1.2. Ela realiza a operação GET do SNMP sobre um conjunto fixo de variáveis, em cada estação do conjunto de estações testada. Essa ferramenta tem o tamanho de 3,29 KB. As Figuras 5.8 e 5.9 exemplificam as operações nos dois ambientes utilizados.

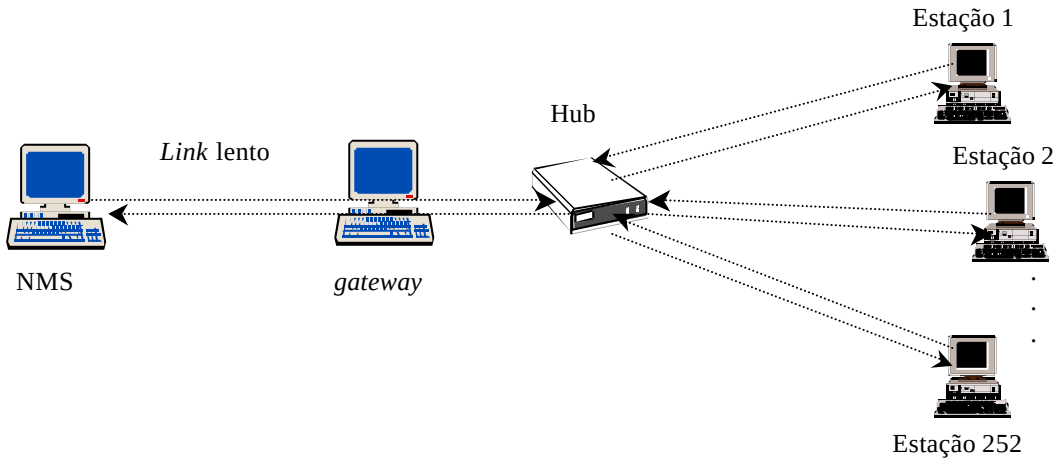


Figura 5.8 – Operação GET no ambiente de rede remota do aplicativo SNMP

Nesses experimentos, o procedimento de marcação do tempo adotado foi o de antes de enviar a requisição SNMP *GET* para determinada variável gerenciada, guardar em uma variável o número de milisegundos passados até o presente momento. E, após receber a resposta válida SNMP, diminuir o número de milisegundos atual do anteriormente guardado e, somar a um contador total de tempo, que fora inicializado com zero, antes do procedimento iniciar. Esse processo é o mesmo para cada uma das varáveis gerenciadas da MIB, utilizadas nos experimentos com o aplicativo SNMP, tanto para o ambiente de rede local quanto para o ambiente de rede remota. Ao final, o tempo total é mostrado na tela.

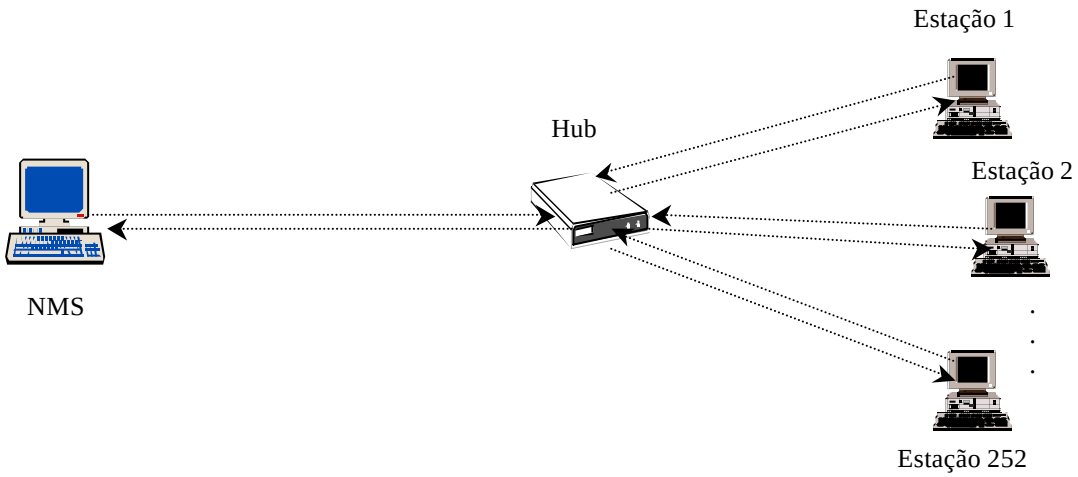


Figura 5.9 – Operação GET no ambiente de rede local do aplicativo SNMP

Para o caso dos experimentos com agentes móveis, o procedimento de marcação do tempo foi diferente. Para todos os efeitos, a realização da tarefa de recuperação das informações para a máquina que inicializou o agente é única, mesmo envolvendo a migração de código. Desta forma, para cômputo do tempo total foi utilizado o tempo obtido na origem; quando o agente sai para iniciar sua “viagem” e quando o agente chega da “viagem”, na estação de gerenciamento (NMS), trazendo as informações. À diferença entre o tempo obtido no final da “viagem” com o tempo obtido no início deu-se o nome de tempo de resposta, ou seja, a métrica utilizada para as comparações.

Além disso, para que fosse possível entender com maior clareza os resultados apresentados, procurou-se avaliar alguns dos componentes do tempo total de resposta,

para o caso dos agentes móveis em rede remota, quais foram: o tempo de recuperação das informações na localização da coleta e os tempos de passagem pelo gargalo; tanto indo da NMS para a rede a ser gerenciada, quanto da rede gerenciada para a NMS.

Todos os experimentos foram realizados separadamente. No momento da realização deles foi utilizado um *hub* exclusivo para as máquinas envolvidas. Desta forma, havia um ambiente controlado, onde a carga das estações estava limitada aos *softwares* de gerenciamento dos recursos de hardware, disponibilizados pelo sistema operacional, a plataforma java e de agentes móveis e o aplicativo SNMP desenvolvido, dependendo do experimento realizado. De tal sorte que, pela dificuldade de medição desta carga, ela foi ignorada para os efeitos dos experimentos, considerando que a diferença nos resultados finais seria relativa.

5.2 Os resultados

Foram realizados vários experimentos no ambiente de teste em rede remota. Esses experimentos realizaram-se aumentando e diminuindo a capacidade do gargalo e executando os agentes móveis e o aplicativo SNMP. No entanto, em ambos os ambientes de teste, os resultados considerados correspondem a média de cinco medidas do mesmo experimento para cada número de estações gerenciadas simuladas.

Para que o agente móvel (com capacidade SNMP) obtenha os valores das variáveis SNMP (*inErrors*, *sysContact* e *sysDescr*) é necessário que, ao chegar no destino, execute chamadas ao agente SNMP que está executando lá. Desta forma, para o cômputo do tempo total de resposta, em cada passo da "viagem" do agente, os componentes são: tempo de migração e tempo de execução da tarefa que é composto do tempo de recuperação da informação, ou seja, o tempo necessário para o envio da requisição SNMP, recepção da resposta e seu armazenamento em uma variável interna ao agente móvel.

Processo similar ocorre com o agente que utiliza JNI, que é utilizado para recuperar as informações de memória volátil, memória virtual e espaço disponível em disco. Ao chegar no nó destino, o agente invoca métodos contidos em API nativa (biblioteca para o sistema operacional utilizado) já carregada, desenvolvida neste trabalho.

Desta forma, foi medido o tempo necessário para a execução das duas componentes da realização da tarefa (recuperação e armazenamento da informação), de maneira que fosse possível comparar o resultado do tempo total de resposta dos experimentos com agentes móveis, levando em consideração o tempo gasto com a recuperação - através de RPC, no caso do SNMP, e através de invocação dinâmica em API nativa, no caso do agente com JNI – e armazenamento das informações desejadas.

No caso da aplicação SNMP, a divisão em duas componentes de tempo (recuperação e armazenamento da informação desejada) não ocorre, uma vez que a recuperação das informações é feita através de requisição e resposta, e é ao mesmo tempo a verdadeira razão do experimento. No entanto, é de se esperar que ocorra retardo de tempo desde o envio de uma requisição *GET* e o recebimento da resposta *RESPONSE*. Sendo assim, também para o experimento com o aplicativo SNMP foi medido o tempo total entre o envio e o recebimento das requisições, tanto no ambiente de teste em rede remota quanto em rede local.

Para que fosse tratada a situação do tempo de migração dos agentes móveis, de forma que fosse possível identificar o tempo de migração entre a última máquina e a NMS (tempo de passagem pelo gargalo de comunicação no retorno dos agentes móveis), e entre a primeira estação e a NMS (tempo de passagem pelo gargalo de comunicação na saída dos agentes móveis), procurou-se realizar a sincronização dos relógios das máquinas envolvidas.

A sincronização foi realizada de forma manual, sujeita portanto a falhas, uma vez que não se tratava do objetivo a realização de tomadas parciais de tempo, apenas o tempo de resposta final. Além disso, a instalação de servidor de sincronização de tempo

poderia influenciar os resultados pelo tempo de processamento dedicado ao gerenciamento do tempo e pela carga gerada pela troca de mensagens de sincronização.

Por outro lado, e para tratar a questão da migração dos agentes móveis não só no gargalo, a partir do momento que se tem o tempo de execução das operações para recuperação das informações e também o tempo de resposta total, é possível, com a subtração, obter o tempo de migração e/ou execução do restante do código dos agentes ou aplicativo SNMP. Desta forma, para uma melhor apresentação dos resultados obtidos, esta será feita por ambiente de execução. Assim, segue a apresentação dos resultados obtidos com os experimentos.

5.2.1 Os resultados em rede remota

A apresentação dos resultados em rede remota será feita de acordo com as 3 diferentes velocidades utilizadas para o gargalo de comunicação: 38400, 57200 e 115200bps.

Inicialmente serão apresentados os resultados obtidos com agentes móveis, tanto utilizando SNMP como JNI. Em seguida, serão apresentados os resultados obtidos com o aplicativo SNMP.

5.2.1.1 Gargalo em 38400bps

Na comparação dos tempos absolutos entre agentes móveis com chamada RPC ao agente SNMP, e agentes móveis com chamada JNI, deve-se considerar a chamada JNI semelhante à chamada RPC, pois o dado é buscado através de invocação à API específica, implementada como biblioteca de vínculo dinâmico, e não disponibilizada com as plataformas de desenvolvimento envolvidas com os experimentos. Nesse caso, os resultados são comparáveis.

Como exemplo da maneira de como foram organizados os dados obtidos, é apresentada a Tabela 5.1, exemplificando os principais dados obtidos.

Número de estações	Tempo de resposta em milisegundos	Tempo de recuperação das informações em milisegundos	Tempo de passagem pelo gargalo na ida em milisegundos	Tempo de passagem pelo gargalo na volta em milisegundos
1	18847	0	104	19125
3	19088	50	130	18869
12	20729	220	169	18860
30	25497	470	302	18869
51	29863	750	357	18910
102	43142	2170	303	18895
252	104190	2890	270	19143

Tabela 5.1 – Resultados do agente móvel com SNMP em rede remota para gargalo em 38400bps

Assim, na Figura 5.10 é possível observar os resultados de tempo de resposta com gargalo em 38400bps

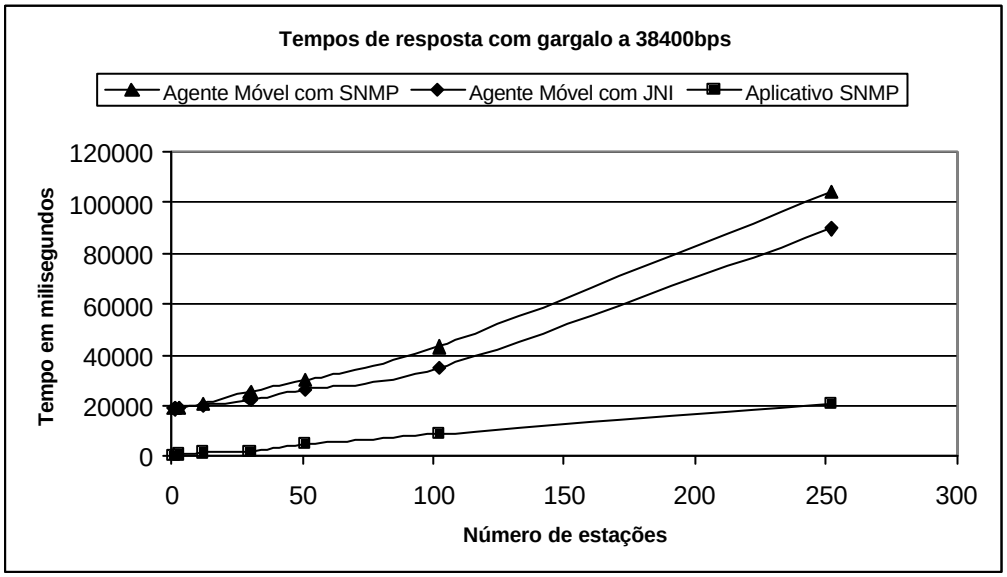


Figura 5.10 – Tempos de resposta em rede remota com gargalo a 38400bps

5.2.1.2 Gargalo em 57200bps

Nesse caso, é possível perceber uma pequena diminuição dos valores de tempo de resposta para os três casos, conforme visualizado na Figura 5.11.

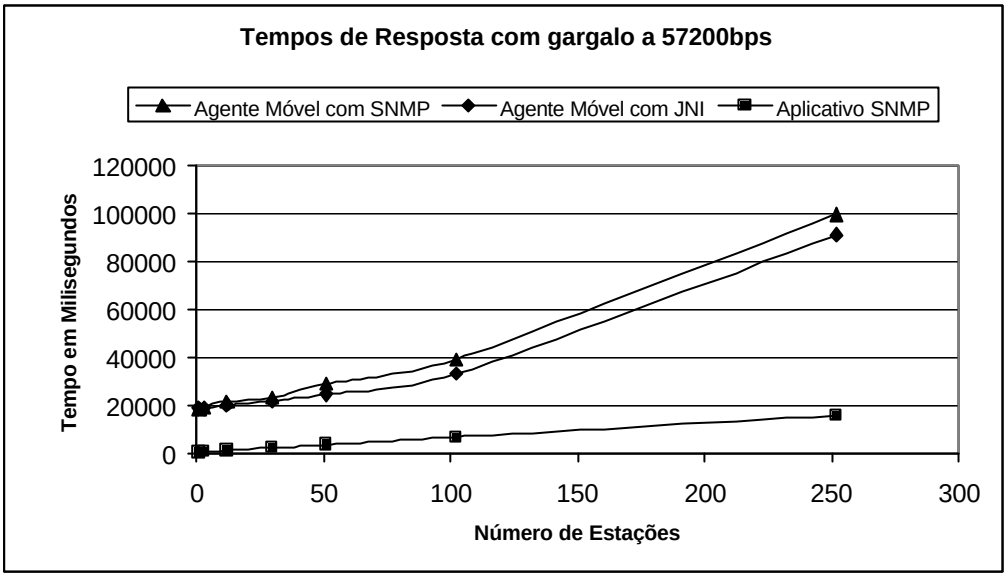


Figura 5.11 – Tempos de resposta em rede remota com gargalo a 57200bps

Porém, mesmo com o aumento de velocidade no enlace de gargalo, e tendo um tamanho inicial menor, a solução de agentes móveis com RPC SNMP continua tendo um tempo de resposta maior que a solução com JNI.

5.2.1.3 Gargalo em 115200bps

Neste caso, é possível perceber uma tendência em que, quanto maior o número de estações a serem gerenciadas, mais os tempos dos agentes móveis tendem a se tornar iguais. Isso é diferente da tendência demonstrada com as velocidades de gargalo anteriores. A Figura 5.12 ilustra o resultado obtido com o gargalo a 115200bps.

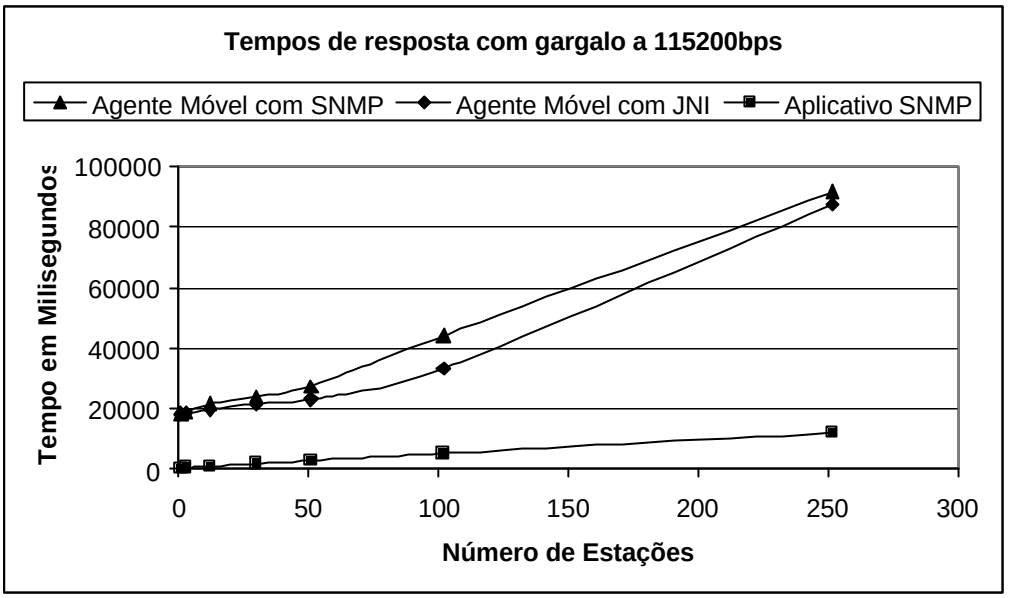
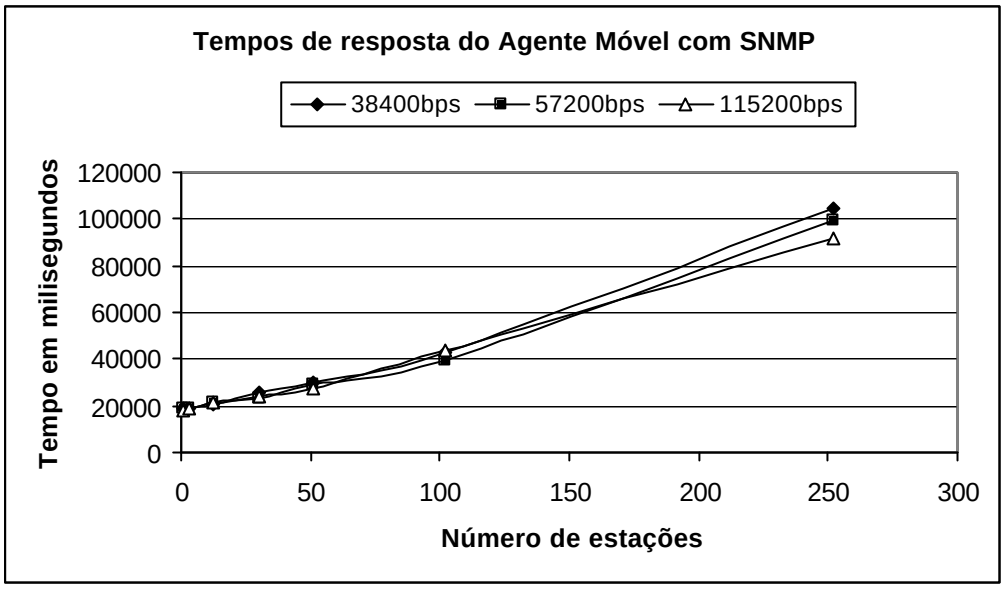


Figura 5.12 – Tempos de resposta em rede remota com gargalo a 115200bps

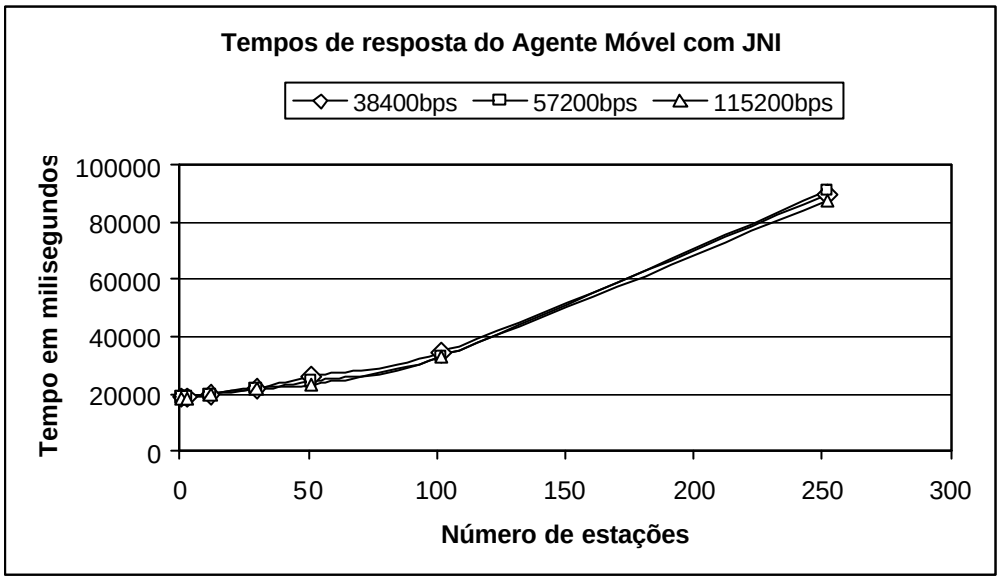
5.2.2 Análise dos resultados em rede remota

Para que seja possível comparar os resultados obtidos em rede remota, é necessário que sejam levados em consideração vários aspectos relativos aos efeitos de diversas variáveis sobre os experimentos. Entre eles, o que mais se adequa aos objetivos deste trabalho está a velocidade do enlace de gargalo. Desta forma, serão apresentados os resultados dos experimentos individualmente, comparando os resultados de tempo de resposta para cada variação de velocidade.

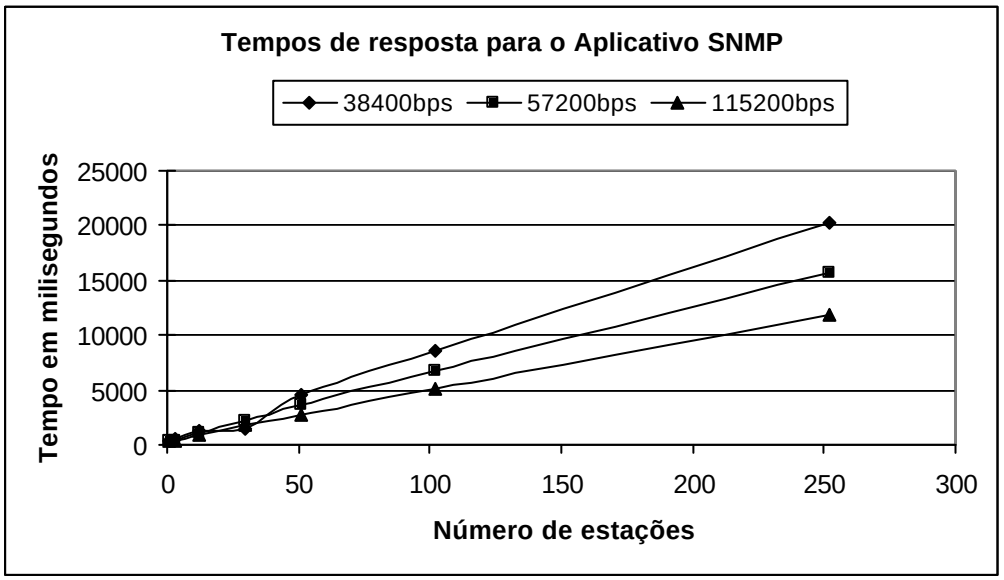
Neste caso, é possível perceber a influência do enlace de gargalo com maior clareza nos experimentos com o aplicativo SNMP, conforme a Figura 5.13 (c). Tanto naquele experimento quanto nos outros - Figura 5.13 (a) e (b) - quanto menor a velocidade do enlace de gargalo maior o tempo de resposta. Para os agentes móveis, essa relação também existe, porém é bem mais sutil. Guardadas as proporções, percebe-se que: tanto para os experimentos com agentes móveis quanto com o cliente SNMP, os valores permanecem bastante similares, independentemente da velocidade de gargalo, até uma quantidade de estações a serem gerenciadas que gira em torno de 30.



(a)



(b)



(c)

Figura 5.13 – Tempos de resposta x Velocidades do gargalo, em rede remota

De qualquer forma, a variação nos tempos de resposta para o agente móvel com SNMP na média gira em torno de 15700ms, para uma variação de 76800bps (115200 – 38400), ou seja, uma variação na proporção de 1ms para cada 5,2bps. Já, para uma variação menor da velocidade do gargalo, como no caso de 115200 para 57200bps, a variação torna-se menor, chegando a 1ms para cada 12,6bps. Essa relação de variação é de 1ms para 5,5bps em 76800bps e de 1ms para 7,65bps em 58000bps para o aplicativo SNMP. Porém, para o agente móvel que utiliza invocação de método nativo (JNI), em ambas as variações de velocidade (76800 ou 58000bps), têm-se que a proporção gira em torno de 1ms para cada 10bps. Em uma primeira análise, é possível que se diga que para este último experimento, as diferenças de velocidade no gargalo de comunicação quase não fazem diferença, uma vez que quanto menores as diferenças entre as proporções, mais parecidas serão as curvas de velocidade por tempo, o que é possível observar na da Figura 5.13 (b).

De uma forma geral percebe-se que os tempos de resposta para o conjunto dos experimentos com agentes móveis são bastante semelhantes, da mesma forma que para o conjunto dos experimentos com o aplicativo SNMP, conforme é possível observar na Figura 5.14.

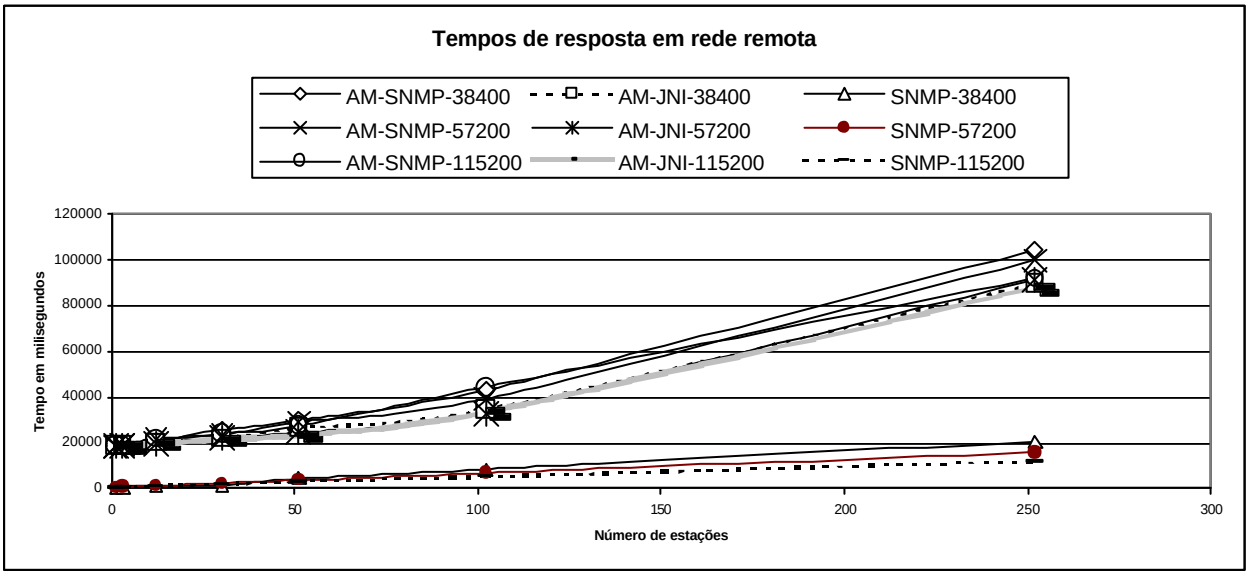
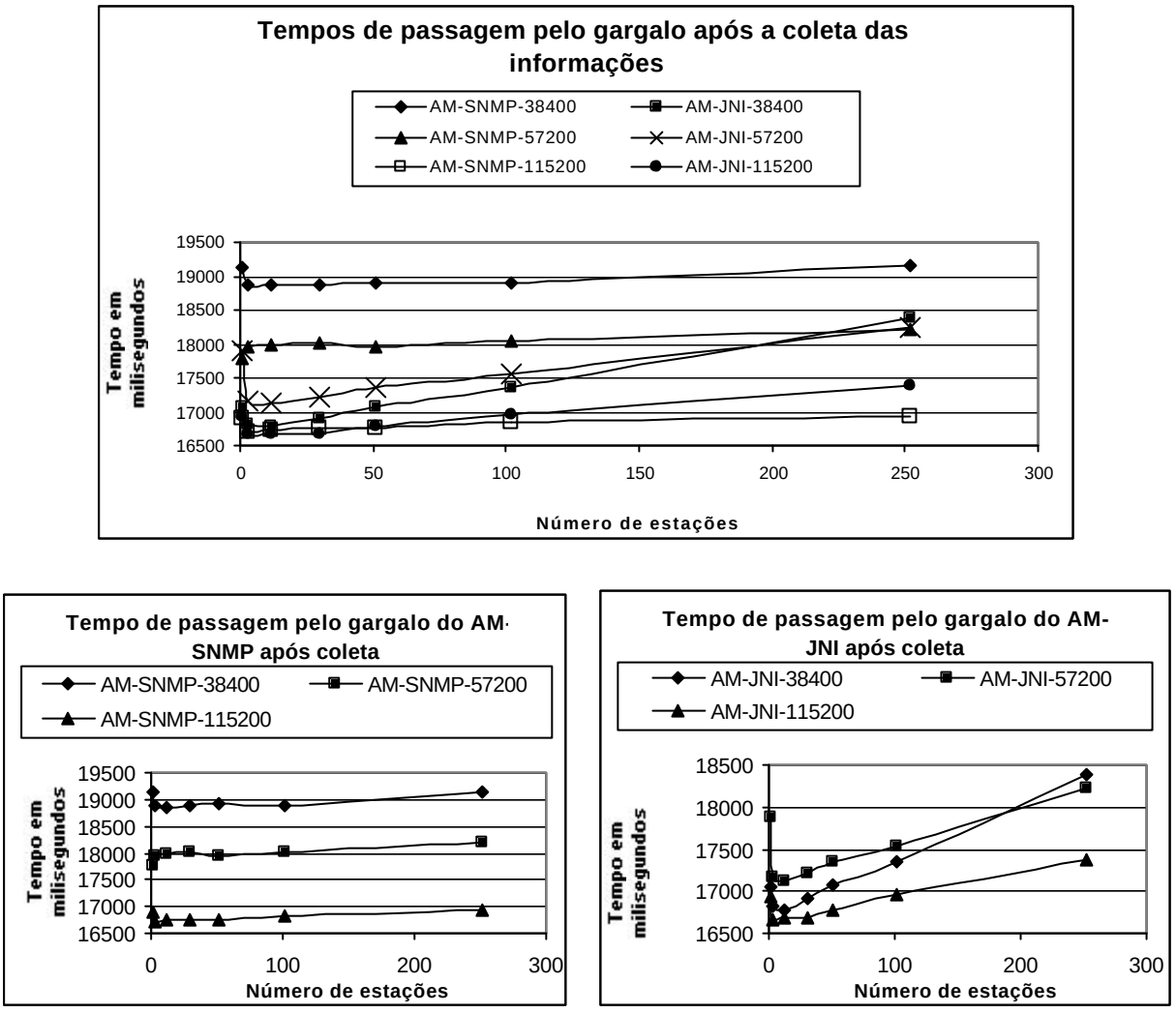


Figura 5.14 – Comparação dos tempos de resposta entre os experimentos em rede remota

Porém, a diferença entre o conjunto dos tempos dos agentes móveis e do conjunto dos tempos do aplicativo SNMP é suficientemente grande para que seja investigada. Para tanto, é possível analisar as diversas componentes da métrica avaliada: tempo de resposta. Dentre esses componentes, foi mensurado o tempo de passagem dos agentes pelo gargalo, após a coleta das informações e o tempo necessário para a recuperação da informação propriamente dita, na estação de trabalho.

Dessa forma, quando observa-se a componente tempo de passagem pelo gargalo, após a coleta das informações, conforme Figura 5.15 (a) e (b), percebe-se uma maior estabilidade ao longo da escalabilidade do experimento com agentes móveis com suporte a SNMP. Isso significa que houve pouco aumento no tempo de passagem no gargalo em relação ao aumento do tamanho do agente móvel, conforme aumentava o número de estações gerenciadas. Para o agente móvel com JNI, esse aumento no tempo de passagem pelo gargalo, após a coleta das informações, foi maior. Na verdade, a taxa de crescimento desse agente também é maior uma vez que ele coleta um número maior de informações por estação gerenciada, o que explicaria retas com maior grau de elevação no tempo de passagem pelo gargalo, após a coleta.



(a) (b)

Figura 5.15 – Tempos de passagem pelo gargalo após coleta das informações

Porém, apesar da maior estabilidade no tempo de passagem pelo gargalo, após a coleta das informações, para cada velocidade do gargalo que foi testada, (o que incita a idéia de que o tamanho final dos agentes móveis com SNMP, em função do aumento do número de estações gerenciadas, pouco influenciou no tempo de resposta) os valores de tempo apresentados para aquele experimento são maiores; a não ser para o experimento cuja velocidade era de 115200bps.

Somado a esse fato, há a observação do componente tempo de recuperação das informações, no tempo de resposta total.

Observando-se a Figura 5.16, é possível verificar que o tempo de recuperação das informações, no caso do agente móvel com SNMP, também contribuiu para que o tempo de resposta fosse maior e menos uniforme que o do agente com JNI. Nela, pode-se observar que, apesar de não depender da velocidade do gargalo, o tempo de recuperação das informações é maior para o agente móvel com SNMP. Já para o agente móvel com JNI, os tempos são extremamente baixos e uniformes.

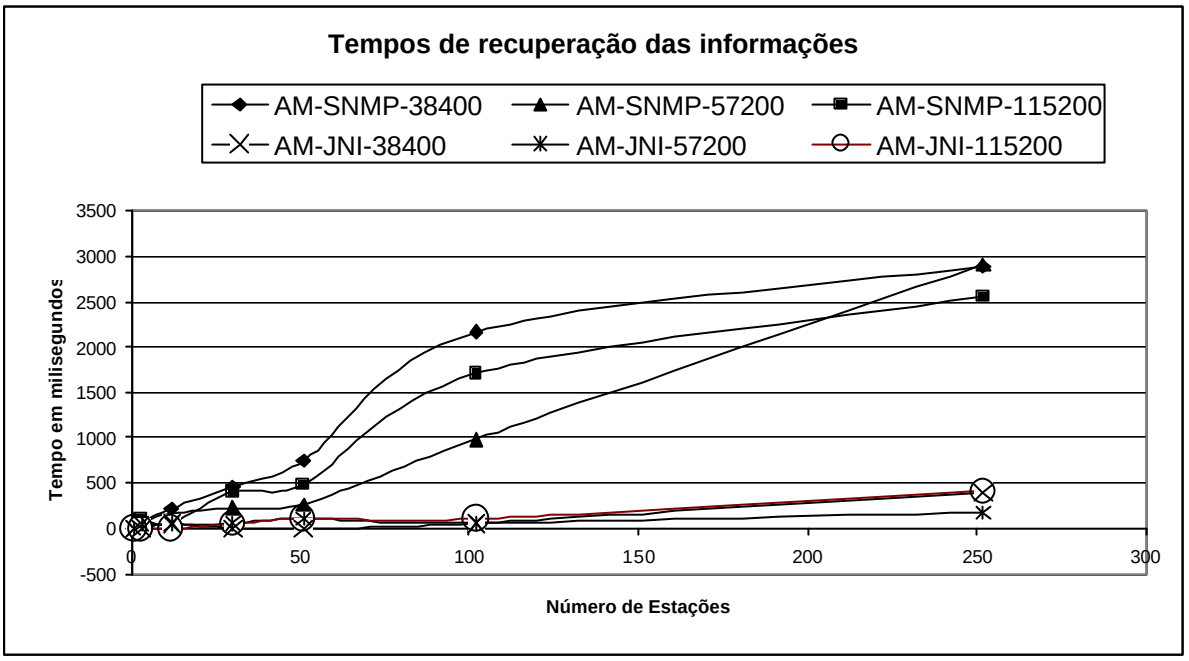


Figura 5.16 – Tempos de recuperação das informações pelos agentes móveis

Isso significa que o mecanismo de recuperação das informações por meio de chamada a API nativa é muito efetivo. Na verdade, a chamada feita pelo agente móvel com SNMP sofre do *overhead* de toda chamada RPC, que é baseada no modelo cliente servidor. A única diferença nesse caso é que o cliente (agente móvel) está na mesma máquina que o servidor (agente SNMP). Assim, o efeito da variável, ou informação a ser recuperada, tem um subcomponente que é o modelo ou técnica utilizada para a recuperação. Percebe-se que, apesar do maior número de chamadas à API, já que o agente móvel com JNI recupera uma quantidade maior de informações, o tempo de recuperação dessas informações é menor, é a componente que mais influencia no tempo de resposta como um todo.

Pode-se ainda observar na Figura 5.17 o efeito do tamanho dos agentes móveis na passagem pelo gargalo, antes da coleta das informações.

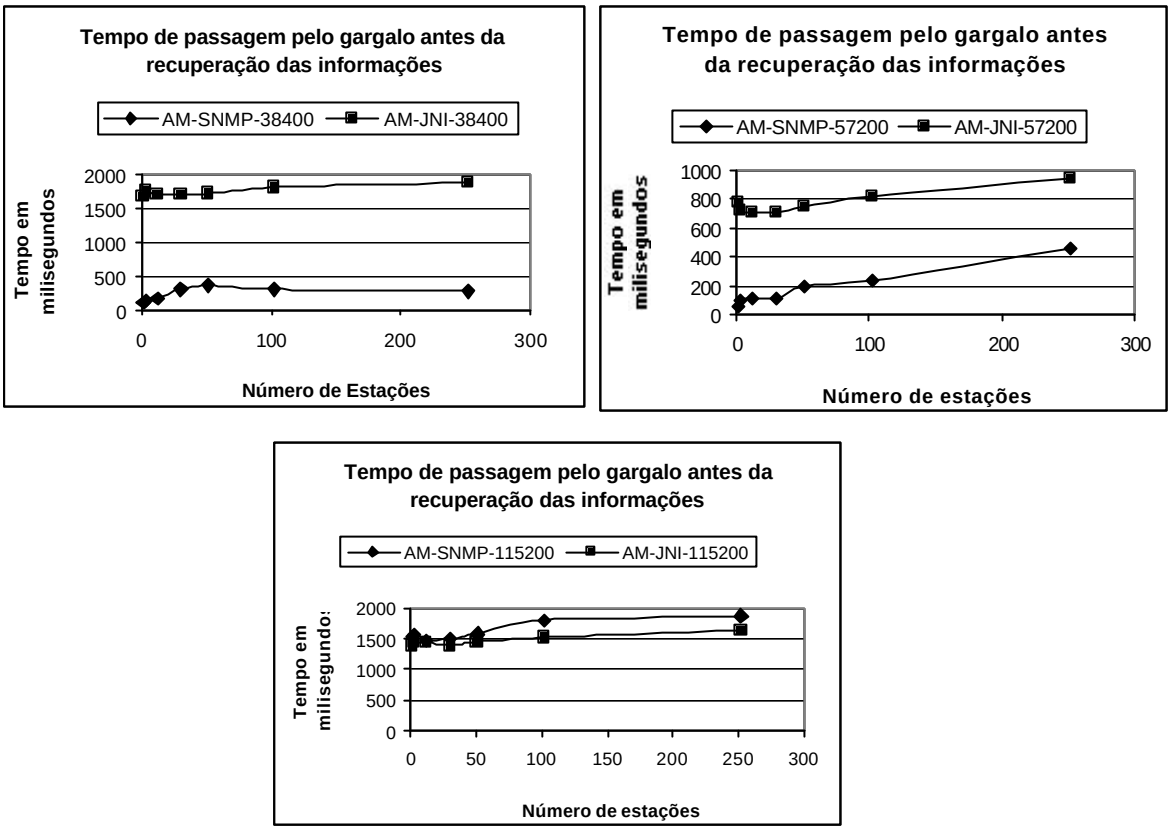


Figura 5.17 – Tempo de passagem pelo gargalo, dos agentes móveis, antes da coleta das informações

É possível observar, como esperado, que o agente móvel com suporte a JNI, por ser maior inicialmente, foi mais lento na passagem pelo gargalo, quando saindo da NMS para a (sub)rede a ser gerenciada. Entretanto, com o gargalo em 115200bps, os tempos apresentados são muito próximos, e a relação se inverte, sendo que o agente móvel com SNMP torna-se mais rápido. Assim, ignorando-se a dificuldade de sincronização dos relógios das máquinas envolvidas, têm-se na média geral, entre os experimentos, que o agente móvel SNMP foi em torno de 48,98% mais rápido[6] que o agente JNI, na

[6] Percentagem obtida pela diferença entre as somas do tempo de passagem pelo gargalo antes da coleta das informações, dos 3 experimentos agente móvel JNI e agente móvel SNMP, em relação ao tempo total de passagem pelo gargalo antes da coleta das informações pelo agente JNI.

passagem pelo gargalo, antes da coleta das informações, para uma diferença de 0,33 KB inicial.

Ainda, quando se observa em conjunto as Figuras 5.17 e 5.15 percebe-se a influência do crescimento dos agentes móveis durante a viagem de coleta das informações. Para se ter uma idéia, considerando as variações de velocidade do gargalo, a passagem do agente móvel com suporte SNMP pelo gargalo após a coleta das informações, chega a ser em torno de 1485% mais lenta[7] que na passagem pelo gargalo com o tamanho inicial.

Quanto ao aplicativo SNMP, o tempo de passagem pelo gargalo, antes e após a coleta das informações, está registrado juntamente com o tempo de recuperação das informações, uma vez que este tempo no experimento com o aplicativo SNMP é marcado desde o envio da requisição SNMP até o retorno da resposta.

No entanto, todas as características avaliadas em conjunto mostram a influência e explicam que, no tempo de resposta, os resultados apresentados pelo agente móvel com suporte a JNI não foram só valores menores que os do agente móvel com suporte a SNMP, mas também gráficos de tempo semelhantes, conforme variava a velocidade do gargalo.

Porém, para a configuração utilizada, o tempo de resposta apresentado pelas medidas do experimento com o aplicativo SNMP foram insuperáveis. Mesmo realizando a passagem de ida e volta pelo gargalo de comunicação para cada uma das estações gerenciadas, o tempo de resposta apresentado foi menor que o dos agentes móveis. É possível, no entanto que, se o gargalo apresentasse uma latência maior e as estações fossem mais rápidas, os AM's apresentassem um desempenho melhor.

[7] Proporção obtida pela soma do tempo de passagem pelo gargalo antes da coleta das informações,com a soma do tempo de passagem pelo gargalo após a coleta das informações dos três experimentos com agente móvel com SNMP.

5.2.3 Os resultados em rede local

Nestes experimentos não existem todos os componentes do tempo de resposta encontrados com os experimentos em rede remota, uma vez que não há de um *link* de comunicação que represente "gargalo". Assim, foi obtida apenas a métrica tempo de resposta e também o tempo de recuperação das informações nas estações de trabalho. Desta forma, é possível observar os tempos de resposta dos três experimentos na Figura 5.18.

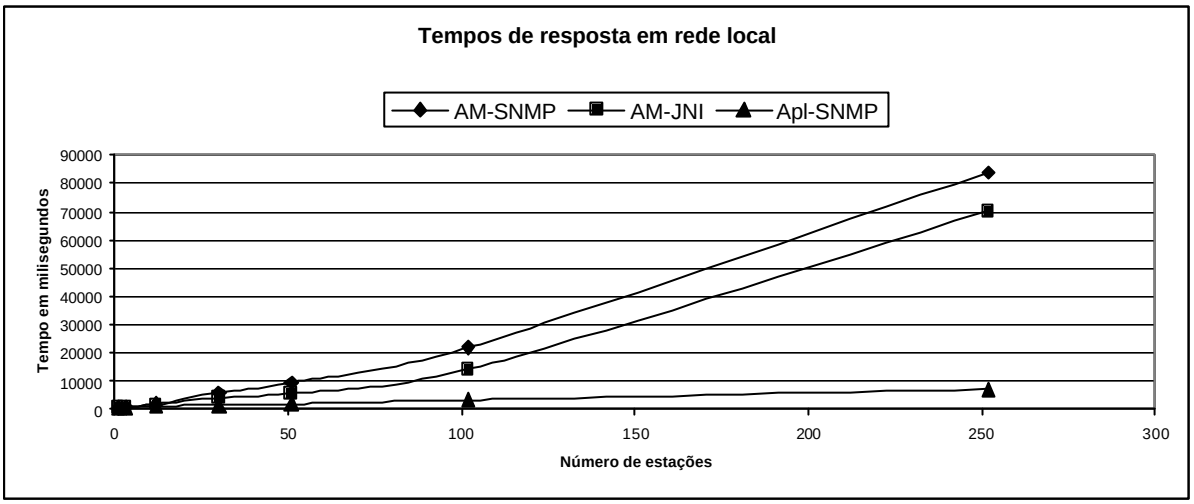


Figura 5.18 – Tempos de resposta em rede local

Para os tempos de recuperação das informações observou-se o seguinte comportamento, conforme a Figura 5.19.

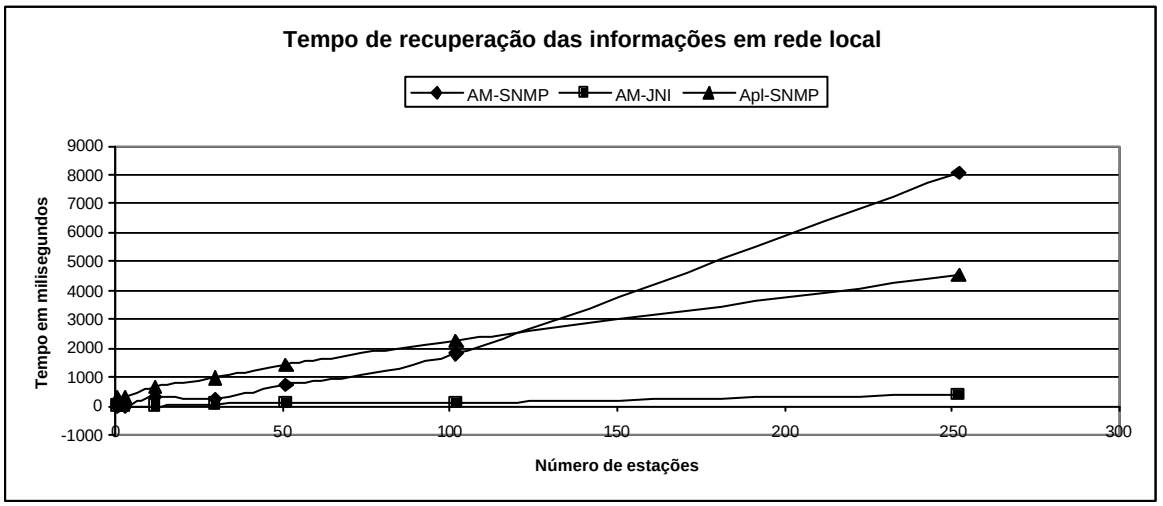


Figura 5.19 – Tempo de recuperação das informações em rede local

Nesse caso do tempo de recuperação das informações em rede local, têm-se que em média, cerca de 68% do tempo de resposta do aplicativo SNMP é constituído pelo tempo de recuperação propriamente dito da informação. Esse índice chega a ficar em torno de 89%, quando o número de estações gerenciadas é pequeno. Para o agente móvel com SNMP, o tempo de recuperação da informação representa bem menos no tempo de resposta, sendo que na média ocorre uma percentagem de cerca de 9%; chegando a ser nula quando o número de estações for menor. Para o agente móvel com JNI, observou-se que a influência é menor: cerca de 0,7% na média.

É possível observar que, em comparação, o tempo de recuperação da informação em rede remota, frente ao tempo de resposta total, representa cerca de 3% para o experimento com agente móvel com SNMP. Já, para o agente móvel com JNI, representa cerca de 0,3% e para o aplicativo SNMP cerca de 70%. Esses valores consideram a soma dos valores dos experimentos com as três velocidades testadas.

Desta forma, percebeu-se que também para o experimento em rede local o aplicativo SNMP obteve melhor desempenho. Mais especificamente, esse experimento foi em média 84,21% mais rápido que o experimento com agente móvel com suporte a JNI e 87,7% mais rápido que o experimento com agente móvel com suporte a SNMP.

5.2.4 Comparação dos resultados em rede remota e rede local

Com os resultados apresentados nas seções anteriores, é natural que eles sejam comparados posteriormente. Para tanto, e para que seja percebido de uma maneira mais efetiva os efeitos do ambiente de desenvolvimento dos experimentos e a relação existente com ambientes de rede reais, principalmente no tocante ao gargalo de comunicação, serão apresentadas as comparações em relação a métrica utilizada, por tipo de experimento realizado.

Na comparação entre os experimentos em rede local e rede remota, é possível observar, principalmente, o efeito do gargalo de comunicação, pois em rede local, esse efeito não é associado diretamente. É possível observar que os tempos de resposta para todos os experimentos em rede local são menores, embora as curvas de tempo sejam semelhantes, o que indica que o efeito do aumento dinâmico no tamanho dos agentes móveis é observável, tanto em rede local quanto em rede remota.

Esperava-se que no cenário de rede remota os agentes móveis tivessem um comportamento resultante melhor que o do cliente SNMP. Variáveis como carga da rede e carga de processamento no nó visitado (no caso dos agentes móveis), ou requisitado (no caso do cliente SNMP), não foram levadas em consideração. No entanto, dentro do ambiente de desenvolvimento dos experimentos, buscou-se minimizar estas variáveis, fazendo com que não houvesse outras aplicações executando nos nós, que não fossem o agente SNMP, o servidor de agentes móveis (quando do experimento com) e os serviços de sistema operacional de rede (SOR), necessários a manter o funcionamento das máquinas.

Em média, os valores de tempo de resposta para o agente móvel com SNMP em rede local são em torno de 49,33% menores quando comparados aos valores apresentados pelo experimento com agente móvel com SNMP em rede remota que obteve melhores resultados, aquele cujo gargalo de comunicação era 115200bps. A Figura 5.20 ilustra os valores.

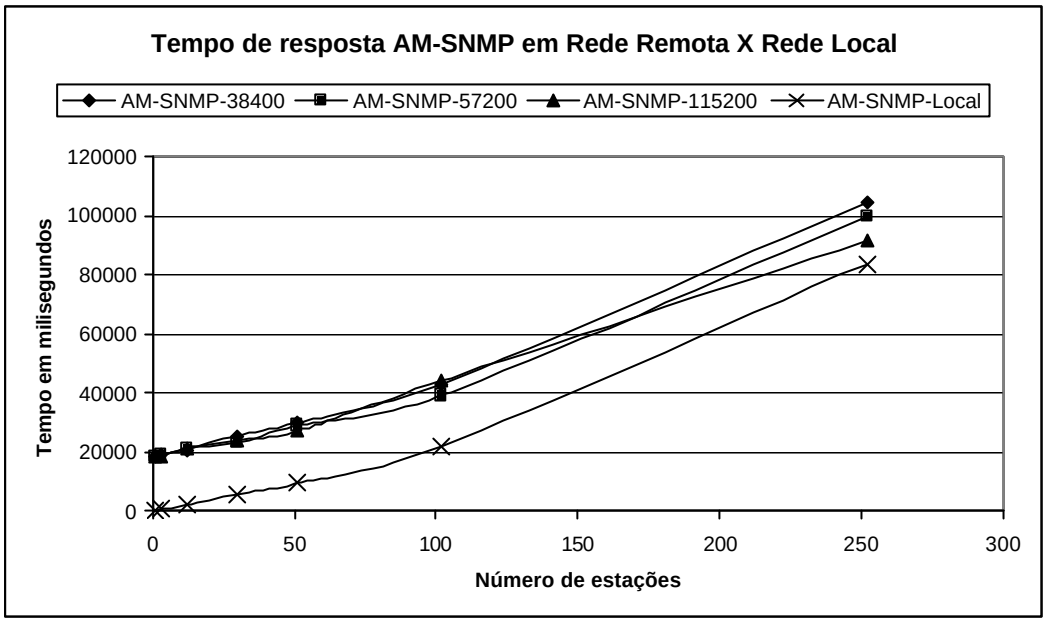


Figura 5.20 – Tempo de resposta do agente móvel SNMP em rede remota X rede local

Já a comparação da métrica tempo de resposta entre os experimentos com agente móvel com JNI mostrou que na média, em rede local, foi aproximadamente 56,34% mais rápido que o experimento em rede remota, que apresentou melhor desempenho. O experimento com agente móvel JNI, que apresentou melhor desempenho em rede remota, foi, como esperado, o experimento cuja velocidade do gargalo era de 115200bps. A Figura 5.21 ilustra estes dados.

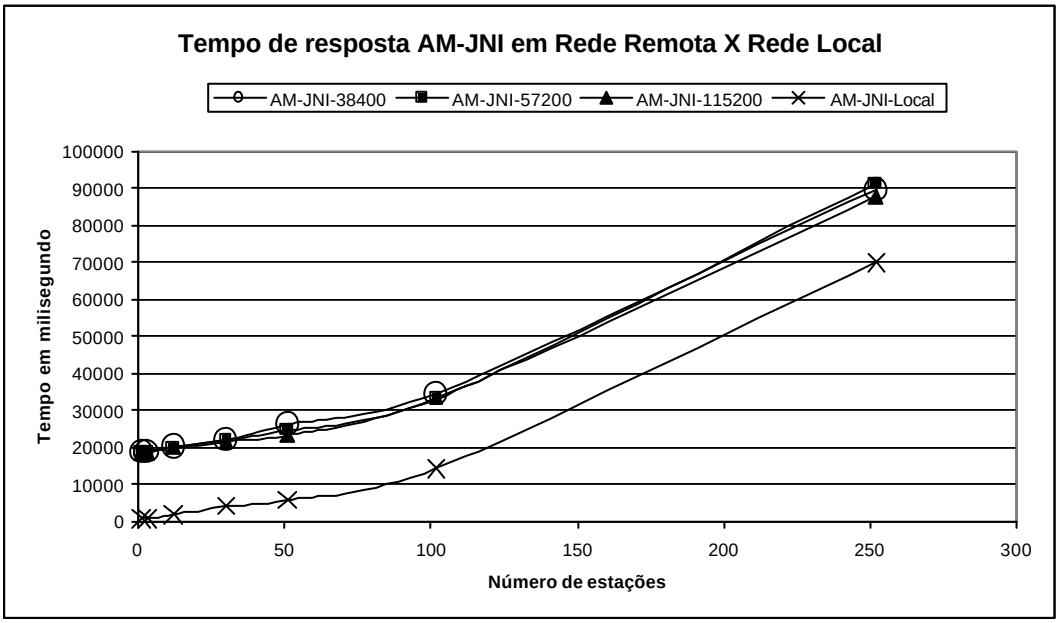


Figura 5.21 – Tempo de resposta do agente móvel JNI em rede remota x rede local

No caso do aplicativo SNMP, a comparação não é diferente. Também aqui o desempenho é melhor em relação a mesma aplicação em rede remota. Mesmo quando comparado ao melhor desempenho em rede remota (quando a velocidade de gargalo é de 115200bps), o aplicativo SNMP em rede local é cerca de 33,53% mais rápido na recuperação das informações requeridas. A Figura 5.22 ilustra esse comportamento

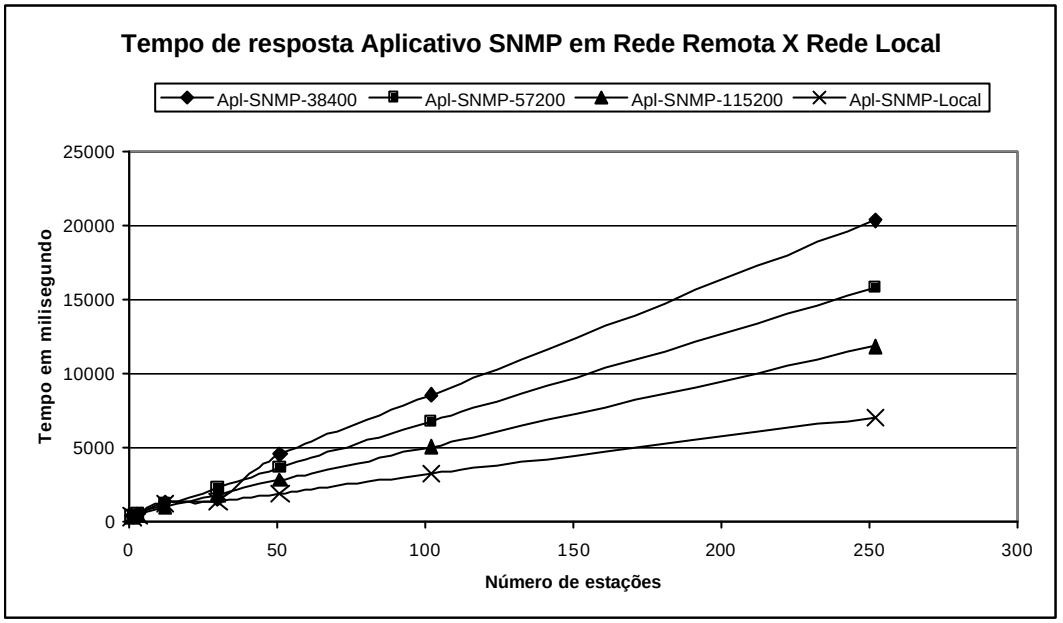


Figura 5.22 – Tempo de resposta do aplicativo SNMP em rede remota x rede local

6 Conclusões

O modelo de agentes móveis já possui uma certa maturidade, mas ainda é bastante pesquisado e cada vez mais utilizado. Dentre as aplicações que mais o utilizam estão aquelas que fazem uso da redução no tempo de resposta.

As atividades da gerência de redes de computadores envolvem a recuperação de informações. Nesse sentido, esta área também busca reduzir o tempo de realização destas tarefas. Assim, é plausível considerar a utilização deste paradigma para tanto. Porém, são necessários estudos que comprovem a efetividade dessa solução.

Esse trabalho apresentou uma análise de desempenho entre o modelo de agentes móveis e o modelo de gerenciamento baseado em SNMP. Essa análise foi baseada na comparação da métrica tempo de resposta, entre agentes móveis e um aplicativo cliente com suporte a SNMP. A métrica tempo de resposta foi obtida através de técnicas de medidas, realizando a recuperação de informações relevantes para a gerência de redes de computadores.

Foram implementados agentes móveis para a plataforma Aglets1.1.0, bem como um aplicativo com suporte a SNMP, para ser executado sobre a JVM. Os agentes suportavam SNMP ou JNI.

Dentre os fatores que foram percebidos como relevantes aos resultados obtidos está o tempo de passagem pelo gargalo de comunicação tanto dos agentes móveis utilizados como também do aplicativo cliente SNMP. A influência exercida por este fator no experimento com agentes móveis leva à constatação de que por mais rápido que seja o processamento nos clientes para a recuperação de informações, bem como tão rápida seja a rede local por onde circulam os agentes móveis, o *link* de comunicação é fator preponderante no desempenho total.

Foi avaliada, pelos experimentos realizados, a tecnologia de recuperação das informações (SNMP e JNI) para agentes móveis, além da variação de velocidade de gargalo, que era o principal comportamento a ser estudado na questão do desempenho entre agentes móveis e aplicativo SNMP.

No caso dos experimentos realizados, a influência do tamanho dos agentes móveis, ocasionada pela quantidade de informações requisitadas, reflete-se diretamente no número de mensagens trocadas entre as máquinas participantes da transmissão dos agentes móveis através da conexão *Agent Transfer Protocol* (ATP) utilizada pelo Aglets [AGLETS], já que neste protocolo de aplicação é realizado o reconhecimento de mensagens. Assim, quanto maior o agente móvel, maior o número de mensagens de reconhecimento trocadas e maior o tempo necessário para a transmissão do agente. E isso ocorre em cada nó visitado. Além disso, problema semelhante pode ocorrer com o protocolo TCP no esquema de controle de congestionamento através do mecanismo de janela deslizante, uma vez que, dependendo do tamanho da janela do transmissor e do receptor, ocorrerá fragmentação de pacotes. O mesmo acontece na camada de enlace quanto ao parâmetro *Maximum Transmission Unit* (MTU), quando o(s) agente(s) percorrem redes que utilizam diferentes tecnologias de enlace de dados.

Os resultados indicam que os agentes móveis consomem um tempo maior de processamento, principalmente pela serialização/desserialização dos objetos, criação de *threads* e troca de mensagens, inclusive de reconhecimento.

É importante enfatizar que a recuperação das informações pelos agentes móveis, em rede remota, com o uso de chamadas JNI baixou o tempo de resposta em 10,20%, quando comparado ao agente móvel com o uso de requisições SNMP. Portanto, ao se pensar em uma solução de gerenciamento de redes que utilize agentes móveis, é importante observar a tecnologia de recuperação da informação.

Também em rede remota, houve redução de 88,09% no tempo de resposta, na média geral, do cliente SNMP em comparação ao agente móvel com suporte a SNMP.

Em rede local, as comparações não foram diferentes. Esses valores foram de aproximadamente 22,03% de redução no tempo de resposta, relativo ao agente móvel com JNI, em comparação ao agente móvel com SNMP. Também houve redução no tempo de resposta, quando executado o cliente SNMP. Nesse caso, quando comparado ao tempo de resposta do agente móvel com suporte a SNMP, a redução foi de aproximadamente 87,70%.

Com a utilização da plataforma de agentes móveis, percebeu-se que otimizações visando melhora no protocolo de transferência de agentes faria com que o desempenho dos agentes móveis melhorasse. Observou-se com os dados obtidos pelos agentes móveis, que o acesso de forma direta às informações de gerência melhora significativamente o tempo de resposta.

Assim, na decisão de se adotar um sistema que utilize agentes móveis deve-se levar em consideração a tecnologia de coleta de dados a ser utilizada. Se as informações estão disponíveis de uma forma fácil de recuperá-las, melhor será para o quesito desempenho, pois menor será o tempo gasto especificamente com as operações de captura e armazenamento delas.

Dessa forma, diante dos dados levantados, no cenário estudado onde poucas informações são buscadas, a alternativa que melhor se adapta é a utilização de agentes SNMP estáticos, recebendo requisições e enviando respostas com as informações desejadas.

6.1 Trabalhos Futuros

Como trabalhos futuros, é possível citar a continuação da avaliação de desempenho da utilização de agentes móveis em aplicações de gerência de redes que se utilizem de gargalo de comunicação, até mesmo com latência maior, utilizando aplicações de gerenciamento legadas, onde se possa estender a MIB padrão, por exemplo, com as informações de memória física e de disco, levantadas nos

experimentos que utilizaram a interface nativa Java - JNI. Para tanto, poderia ser utilizado um sistema onde, em uma rede separada de outra pelo gargalo de comunicação, estivesse sendo executado agente SNMP com capacidades de suporte aos protocolos DPI e RDPI. De outro lado estaria a estação gerenciadora, que executaria um sistema de agente móvel que suportasse a capacidade descrita acima. Nesse caso, poder-se-ia avaliar não só o desempenho em termos de tempo de resposta da aplicação, comparada com os resultados obtidos neste trabalho, mas também a influência da extensão da MIB, quando comparada a utilização do JNI, e do suporte ao SNMP no próprio código do agente móvel.

Fazer uma avaliação utilizando mais de uma plataforma de agentes móveis para se encontrar o *overhead* que determinada plataforma tem sobre os resultados finais de tempo de resposta, até mesmo como forma de responder se a serialização de objetos implementada pela JVM ou a troca de mensagens do Aglets exerce influência significativa no tempo de resposta ao ponto de considerá-la principal responsável pelo baixo desempenho dos experimentos com agentes móveis.

Aplicação da abordagem utilizada com outros sistemas operacionais na NMS e também nas estações de trabalho para verificar a diferença no comportamento da plataforma de agentes móveis adotada.

Realizar a avaliação aplicando a prototipagem utilizada, implementando novos comportamentos nos agentes móveis como o retorno a estação de gerenciamento após alcançar determinado tamanho limite para a rede na qual ele está ou o envio das informações coletadas através de troca de mensagem com a NMS. Ou mesmo a criação de um “celeiro de informações” em determinada(s) localização(ões) na rede, consistindo no depósito das informações recolhidas, para que elas possam ser buscadas, posteriormente, por outros agentes ou através de requisições cliente “normais”.

7 Referências Bibliográficas

[AGLETS] URL: A plataforma de agentes móveis aglets: http://www.trl.ibm.com.jp/aglets, dezembro 2000.

[BALDI97] BALDI, M. Exploiting code mobility in decentralized and flexible network management. In 1st International Workshop on Mobile Agents (MA'97). Berlim, Alemanha, abril 1997, apud [BARAS01].

[BARAS01] BARAS, J. S., XI, H., YANG, S., LI, H. System designs for adaptative, distributed network monitoring and control. In 7th IFIP/IEEE International Symposium on Integrated Network Management (IM'01). Seattle, Washington, EUA, maio 2001, pp. 77-90.

[BROWN97] BROWN, M., NAJORK, M. Distributed active objects. Dr. Dobb's Journal, março 1997, pp 34-41.

[BRUSIC00] BRUSIC, I., HASSLER, V., LUGMAYR, W. Deployment of Mobile Agents in the Mobile Telephone Network Management. In 33rd Hawai International Conference on System Sciences (HICSS'00). Honolulu, Havaí, abril 2000. Disponível em htpp://citeseer.nj.nec.com/304161.html, maio de 2001.

[COSTA99] DA SILVA COSTA, T. F. Avaliação Analítica do uso de agentes móveis na gerência de redes. Dissertação de Mestrado em Ciências da Computação. Universidade Federal de Santa Catarina, CPGCC, Florianópolis, Brasil, outubro 1999.

[FENG01] FENG, N., AO, G., WHITE, T., PAGUREK, B. Dynamic evolution of network management software by software hot-swapping. In 7th IEEE/IFIP International Symposium on Integrated Network Management (IM'01). Seattle, Washington, EUA, maio 2001, pp. 63-76.

[GENESERETH94] GENESERETH, M.; KETCHPEL, S. Software agents. Communications of the ACM, v. 37, nº 7, julho 1994.

[GRAY00] GRAY, R. S., CYBENKO, G., KOTZ, D., RUS, D. Mobile agents: Motivations and State of the Art. Relatório Técnico TR2000-365. Department of Computer Science, Dartmouth College, abril 2000. Disponível em ftp://ftp.cs.dartmouth.edu/TR/TR2000-365.pdf, junho 2001.

[GRAY01] GRAY, R. S., KOTZ, D., PETERSON, R. A. Jr, et al. Mobile-agent versus client/server performance: Scalability in a information retrieval task. Relatório Técnico TR2001-386. Department of Computer Science, Dartmouth College, janeiro 2001. Disponível em ftp://ftp.cs.dartmouth.edu/TR/TR2001-386.pdf, junho 2001.

[GRAY95] GRAY, R. S. Agent Tcl: A transportable agent system. In the CIKM Workshop on Intelligent Information Agents, Fourth International Conference on Information and Knowledge Management (CIKM'95). Baltimore, Maryland, dezembro 1995. Disponível em http://www.infosys.tuwien.ac.at/Research/Agents/sp-dagents.html, setembro 2001.

[GOLDSZMIDT95] GOLDSZMIDT, G.; YEMINI, Y. Distributed management by delegation. In 15th International Conference on Distributed Systems, junho 1995. Disponível em http://citeseer.nj.nec.com/goldszmidt95distributed.html, junho 2001.

[ISMAIL99] ISMAIL, L., HAGIMONT, D. A performance evaluation of mobile agent paradigm. In International Conference on Object-Oriented Programming, Systems and Applications (OOPSLA'99). Denver, EUA, novembro 1999. Disponível em http://www.daimi.au.dk/dAOOP/papers/aoop25.pdf, maio 2001.

[JOHANSEN98] JOHANSEN, D., SCHNEIDER, B. F., van RENESSE, R. What Tacoma taught us. In MILOJICIC, D., FREDERICK, D., WHEELER, R., editores,

Mobility. Mobile Agents and Process Migration – An Edited Collection. Addison Wesley, 1998.

[JB98] Jumping Beans white paper. Ad. Astra Engineering, Inc.. September 1998. Disponível em http://www.JumpingBeans.com/, junho 2001.

[KOTZ00] KOTZ, D., JIANG, G. GRAY, R., CYBENKO, G., PETERSON, R. A. Performance analysis of mobile agents for filtering data streams on wireless networks. Relatório Técnico TR2000-366. Department of Computer Science, Dartmouth College, maio 2000. Disponível em ftp://ftp.cs.dartmouth.edu/TR/TR2000-366.pdf, junho 2001.

[KOTZ99] KOTZ, D., GRAY, R. S. Mobile code: The future of the Internet. Workshop on Mobile Agents in the Competition and Cooperation (MAC3) at Autonomous Agents. Seattle, Washington, maio 1999. (position paper). http://www.cs.dartmouth.edu/~dfk/papers/kotz:future/, abril 2001.

[MAGEDANZ96] MAGEDANZ, T., ECKARDT, T. Mobile software agents: A new paradigm for telecomunications management. In IEEE/IFIP Network Operations and Management Symposium (NOMS'96). Kyoto, Japão, abril de 1996, apud [RUBINSTEIN01]

[MILOJICIC99] MILOJICIC, Dejan et al. MASIF: The OMG Mobile Agent System Interoperability Facility. ACM Press, 1999.

[MOLE01] O *site* MOLE. Grupo de Sistemas Distribuídos, Universidade de Stuttgart, http://mole.informatik.uni-stuttgart.de/, junho 2001.

[MUHUGUSA98] MUHUGUSA, M. Implementing distributed services with mobile code: The case of Messenger environment. In IASTED International Conference on Parallel and Distributed Systems (Euro-PDS'98), Áustria, julho 1998. Disponível em http://citeseer.nj.nec.com/muhugusa98implementing.html, julho de 2001.

[NASCIMENTO99] NASCIMENTO, A., FRANKLIN, M., OLIVEIRA, M. Desenvolvendo agentes inteligentes para a gerência pró-ativa de redes ATM. In XVII Simpósio Brasileiro de Redes de Computadores (SBRC'99). Salvador, Brasil, maio 1999, pp. 681-696'.

[OBJ97] ObjectSpace Voyager core package technical overview. ObjectSpace, Inc. December 1997. Version 1.

[PAGUREK00] PAGUREK, B., WANG, Y., WHITE, T. Integration of Mobile agents with SNMP: Why and How. In 2000 IEEE/IFIP Network Operations and Management Symposium (NOMS'00). Honolulu, Havaí, abril 2000, pp. 609-622.

[PAPAIOANNOU99] PAPAIOANNOU, T. Mobile Agents: Are they useful for establishing a virtual presence in space? In 1999 American Association for Artificial Inteligence Spring Symposium (AAAI-SS'99). Stanford University, Palo Alto, California, EUA, março 1999. Disponível em http://www.luckyspin.org/Docs/Papers/aaai-ss99.pdf, agosto 2001.

[PAVLOU00] PAVLOU, G., BOHORIS, C., CRUICKSHANK, H. Using mobile agents for networking performance management. In 2000 IFIP/IEEE Network Operations and Management Symposium (NOMS'00). Honolulu, Havaí, abril de 2000, pp. 637-652.

[PEINE97] PEINE, H., STOLPMANN, T. The architecture of the ARA plataform for mobile agents. In First International Workshop on Mobile Agents (MA'97). Berlim, abril de 1997. Disponível em http://citeseer.nj.nec.com/peine97architecture.html, abril 2001.

[RFC1228] Carpenter, G. Wijnen, B. Simple Network Management Protocol Distributed Program Interface, RFC 1228, maio 1991.

[RUBINSTEIN01] RUBINSTEIN, M. G. Avaliação de Desempenho de Agentes Móveis no Gerenciamento de Redes. Tese de Doutorado em Engenharia Elétrica. Universidade Federal do Rio de Janeiro, COPPE, Rio de Janeiro, março 2001.

[SURI00] SURI, N., BRADSHAW, J., et al. NOMADS: Toward a strong and safe mobile agent system. In 4th International Conference on Autonomous Agents (Agents 2000). Barcelona, Espanha, junho 2000, apud [FENG01].

[VIGNA98] VIGNA, G. Mobile code technologies, paradigms and application. Tese de Pós-Doutorado (PhD) em Engenharia, Informática e Automação. Politecnico di Milano. Milão, Itália, fevereiro 1998. Disponível em http://www.cs.ucsb.edu/~vigna/pub/Vigna_PhDThesis97.ps.gz , maio 2001.

[WHITE97] WHITE, E. J., Mobile Agents. In BRADSHAW, J. editor, Software Agents. MIT Press, 1997, cap. 19, pp. 437-472.

[WONG97] WONG, D., PACIOREK, N., WALSH, T., DiCILIE, J., YOUNG, M., PEET, B. Concordia: An Infraestructure for collaborating mobile agents. In First International Workshop on Mobile Agents (MA'97), Springer-Verlag, 1997, pp 86-97.

[WOOLDRIDGE95] WOOLDRIDGE, M.; JENNINGS, N. Intelligent agens: theory and practice. In Knowledge Engineering Review, janeiro 1995. Disponível em http://www.csc.liv.ac.uk/~mjw/pubs/ker95.ps.gz.

[YEMINI91] YEMINI, Y.; GOLDSZMIDT, G.; YEMINI, S. Network management by delegation. In: 2th International Symposium on Integrated Network Management (IM'91).Washington, DC, EUA, abril 1991. Disponível em ftp://ftp.cs.columbia.edu/pub/german/cas91.ps, julho 2001.

[YEMINI93] YEMINI, Y. The OSI network management model. IEEE Communications, maio 1993, pp. 20-29.

Anexo A – Código fonte dos agentes móveis

Agente Móvel com suporte a SNMP

```
/*
 * @(#)AgenteColetor.java
 *
 */

import com.ibm.aglet.*;
import com.ibm.aglet.util.*;
import com.ibm.agletx.util.SeqPlanItinerary;
import java.util.Enumeration;
import java.io.*;
import com.adventnet.snmp.snmp2.*;
import com.adventnet.snmp.beans.*;
import java.lang.*;

/**
 * <tt> AgenteColetorSNMP </tt> Utilização no experimento com agentes móveis e
SNMP *
 * @version     1.00   $Date: 2001/08/01 05:10:39 $
 * @autor   Adriano Fiorese
 */

public class AgenteColetorSNMP extends Aglet {
    StringBuffer buffer;
    SeqPlanItinerary itinerary;
    Long totalTime, totalTaskTime;

    public void onCreation(Object ini) {
     itinerary = new SeqPlanItinerary(this);
    }

    public boolean handleMessage(Message msg) {
     if (msg.sameKind("getLocalInfo")) {
     System.out.println("Inicio execucao do agente =" + new
Long(System.currentTimeMillis()).toString());
       getLocalInfo(msg);
       return true;
     } else if (msg.sameKind("dialog")) {

          String nEstacoes = "1";
  //      DataInputStream in = new DataInputStream(new
          BufferedInputStream(System.in));
          BufferedReader in = new BufferedReader(new
          InputStreamReader(System.in));
          System.out.print("Digite o numero de estacoes a serem simuladas: ");

          try {
               nEstacoes = in.readLine();
          } catch (IOException e) {
            System.out.println("Problema de leitura!!");
            return false;
          }
```

```
          inicializaItinerario(new Integer(nEstacoes).intValue());
          start();
          return true;
      } else if (msg.sameKind("printResult")) {

      /*
      Armazena o tempo de chegada para que seja possível saber quanto tempo
      levou a travessia pelo gargalo, diminuindo este tempo do tempo apresentado
      na ultima estação antes da migração.
      */

      String tempo1 = new Long(System.currentTimeMillis()).toString();

      /*  Armazena o tempo total antes de mostrar os resultados*/
      String tempo = new Long(System.currentTimeMillis() -
(totalTime.longValue())).toString();

      /* Mostra as informações recolhidas */
      System.out.println(buffer);

      /* Mostra os tempos */
      System.out.println("Tempo de chegada apos passagem pelo gargalo = "+
tempo1);
      System.out.println("Tempo total em milisegundos, para conseguir as
informacoes: " + tempo);
      System.out.println("Tempo total para recuperar as informacoes =
"+totalTaskTime.toString());
      System.out.println("\n");

      totalTaskTime = null;
      totalTime = null;

        return true;
      }
      return false;
     }

   private void inicializaItinerario(int nEstacoes) {
   /* Monta o itinerario a ser percorrido com o endereco IP das estações
        que participarão do experimento
     */

        itinerary.clear();
      itinerary.addPlan("atp://200.200.6.2:4434/","getLocalInfo");
        itinerary.addPlan("atp://200.200.6.3:4434/","getLocalInfo");
        itinerary.addPlan("atp://200.200.6.4:4434/","getLocalInfo");

        int tam = itinerary.size();
      // Estações a simular é menor que o tamanho do itinerario
        // Elimino os destinos que nao desejo visitar
        if (nEstacoes < tam) {
        for (int i=(tam-1);i>=nEstacoes;i--) {
          itinerary.removePlanAt(i);
        }
      }
      // Estações a simular é maior que o tamanho do itinerario
      // Aumenta o itinerario na qtidade de estacoes que falta
      // com as estações já existentes
      else if (nEstacoes > tam){
        for(int i = 0; i< (nEstacoes - tam); i++) {

   itinerary.addPlan(itinerary.getAddressAt(i),itinerary.getMessageAt(i).getKin
d());
        }
      }
```

```
    // imprime o resultado na maquina gerenciadora (Network Management
Station)
    itinerary.addPlan("atp://192.168.2.2:4434/","printResult");
  }


   public void start() {
    buffer = new StringBuffer();
    if (totalTaskTime == null) { totalTaskTime = new Long(0); }
    if (totalTime == null) { totalTime = new Long(System.currentTimeMillis());
}
    System.out.println("Tempo antes do gargalo = " + new
Long(System.currentTimeMillis()).toString());
    try {
      itinerary.startTrip();
    } catch (Exception ex) {
      ex.printStackTrace();
    }
   }

   public void getLocalInfo(Message msg) {

  String inErrors = ".1.3.6.1.2.1.2.2.1.14.1"; //ifInErrors vai até 14, o 1
indica primeira ifInErrors
    String sysContact = ".1.3.6.1.2.1.1.4.0";
    String sysDescr   = ".1.3.6.1.2.1.1.1.0";

  SnmpTarget target = new SnmpTarget();

    target.setSnmpVersion(SnmpAPI.SNMP_VERSION_1);

    target.setCommunity("public");
    target.setTargetPort(161);
    target.setTimeout(60);

    target.setTargetHost("localhost");  // set host

    //buffer.append(getAgletContext().getHostingURL().toString());

    long t1 = System.currentTimeMillis();

    target.setSnmpOID(new SnmpOID(inErrors)); // seta a variável a ser
gerenciada
    buffer.append(target.snmpGet());

    target.setSnmpOID(new SnmpOID(sysContact)); // seta a variável a ser
gerenciada
    buffer.append(target.snmpGet());

    target.setSnmpOID(new SnmpOID(sysDescr)); // seta a variável a ser
gerenciada
    buffer.append(target.snmpGet());

    target = null;


    /*   Imprime o tempo de realização da tarefa propriamente dita mais o
     tempo de quando o agente móvel inicia a saida do nó sendo gerenciado.
      Com este ultimo dado é possível verificar quanto tempo o agente
     leva do último nó gerenciado até a estação gerenciadora (NMS).
      Assim é possível identificar qual a influencia da passagem pelo gargalo
     no tempo de resposta total, diminuindo do tempo de resposta total, o
     tempo apresentado no último nó gerenciado.
    */
```

```
    long t2 = System.currentTimeMillis() - t1;
    totalTaskTime = new Long(totalTaskTime.longValue() + t2);

    System.out.println("Tempo total de realizacao da tarefa (GET's) = "+new
Long(System.currentTimeMillis() - t1).toString());
    System.out.println("Tempo apos a tarefa (GET's) = "+new
Long(System.currentTimeMillis()).toString());
    System.out.println("\n");

  }

}
```

Agente Móvel com suporte a JNI

```
/*
 * @(#)AgenteColetor.java
*/

import com.ibm.aglet.*;
import com.ibm.aglet.util.*;
import com.ibm.agletx.util.SeqPlanItinerary;
import java.util.Enumeration;
import java.io.*;
/**
 * <tt> AgenteColetor </tt> usado no experimento com agentes móveis e JNI
 *
 * @version     1.00  $Date: 2001/08/02
 * @autor  Adriano Fiorese
 */
public class AgenteColetor extends Aglet {
    StringBuffer buffer;
    SeqPlanItinerary itinerary;
    Long totalTime, totalTaskTime;

    public native long getFreePhysicalMemory();
    public native long getTotalPhysicalMemory();
    public native long getFreeVirtualMemory();
    public native long getTotalVirtualMemory();
    public native long getFreeDiskSpace();

    static {
               System.loadLibrary("memory");
    }

    public void onCreation(Object ini) {
     itinerary = new SeqPlanItinerary(this);
    }

    public boolean handleMessage(Message msg) {
     System.out.println("Inicio execucao do agente =" + new
Long(System.currentTimeMillis()).toString());
     if (msg.sameKind("getLocalInfo")) {
       getLocalInfo(msg);
       return true;
     } else if (msg.sameKind("dialog")) {

         String nEstacoes = "1";
//     DataInputStream in = new DataInputStream(new
       BufferedInputStream(System.in));
       BufferedReader in = new BufferedReader(new
       InputStreamReader(System.in));
```

```
        System.out.print("Digite o numero de estacoes a serem simuladas: ");
        try {
             nEstacoes = in.readLine();
        } catch (IOException e) {
          System.out.println("Problema de leitura!!");
          return false;
        }
        inicializaItinerario(new Integer(nEstacoes).intValue());
        start();
        return true;
    } else if (msg.sameKind("printResult")) {
        /*
          Armazena o tempo de chegada para que seja possível saber quanto
tempo levou a travessia pelo gargalo, diminuindo este tempo do tempo
apresentado na ultima estação antes da migração.
        */
        String tempo1 = new Long(System.currentTimeMillis()).toString();

        /* Armazena o tempo total antes de mostrar os resultados*/
          String tempo = new Long(System.currentTimeMillis() -
totalTime.longValue()).toString();

        /* Mostra as informações recolhidas */
        System.out.println(buffer);

        /* Mostra o tempo total */
        System.out.println("Tempo total em milisegundos, para conseguir as
informacoes: " + tempo);
        System.out.println("Tempo de chegada apos passagem pelo gargalo =
"+tempo1);
        System.out.println("Tempo gasto na recuperacao das informacoes =
"+totalTaskTime.toString());
        System.out.println("\n");

        totalTaskTime = null;
        totalTime = null;
      return true;
    }
    return false;
   }

  private void inicializaItinerario(int nEstacoes) {
    /*
       Monta o itinerario a ser percorrido com o endereco IP das estações
       que participarão do experimento
    */
    itinerary.clear();
    itinerary.addPlan("atp://200.200.6.2:4434/","getLocalInfo");
      itinerary.addPlan("atp://200.200.6.3:4434/","getLocalInfo");
      itinerary.addPlan("atp://200.200.6.4:4434/","getLocalInfo");
    int tam = itinerary.size();
        // Estações a simular é menor que o tamanho do itinerario
      // Elimino os destinos que nao desejo visitar
      if (nEstacoes < tam) {
           for (int i=(tam-1);i>=nEstacoes;i--) {
             itinerary.removePlanAt(i);
           }
        }
        // Estações a simular é maior que o tamanho do itinerario
        // Aumenta o itinerario na qtidade de estacoes que falta
        // com as estações já existentes
        else if (nEstacoes > tam){
            for(int i = 0; i< (nEstacoes - tam); i++) {
```

```
  itinerary.addPlan(itinerary.getAddressAt(i),itinerary.getMessageAt(i).getKin
d());
             }
         }
         // imprime o resultado na maquina gerenciadora (Network Management
Station)
         itinerary.addPlan("atp://200.200.6.1:4434/","printResult");
  }


    public void start() {
     buffer = new StringBuffer();
     if (totalTaskTime == null) { totalTaskTime = new Long(0); }
     if (totalTime == null) { totalTime = new Long(System.currentTimeMillis());
}
     System.out.println("Tempo antes do gargalo = " + new
Long(System.currentTimeMillis()).toString());
     try {
       itinerary.startTrip();
     } catch (Exception ex) {
       ex.printStackTrace();
     }
    }

    public void getLocalInfo(Message msg) {
       long t1 = System.currentTimeMillis();
    buffer.append("\n");
    buffer.append("Machine : "+getAgletContext().getHostingURL().toString());
     buffer.append("\n");
     buffer.append("Total Physical Memory : "+new
Long(this.getTotalPhysicalMemory()).toString()+"kb");
     buffer.append("\n");
    buffer.append("Free Physical Memory : "+new
Long(this.getFreePhysicalMemory()).toString()+"kb");
     buffer.append("\n");
     buffer.append("Total Virtual Memory : "+new
Long(this.getTotalVirtualMemory()).toString()+"kb");
     buffer.append("\n");
    buffer.append("Free Virtual Memory : "+new
Long(this.getFreeVirtualMemory()).toString()+"kb");
    buffer.append("\n");
    buffer.append("Free Disk Space : "+new
Long(this.getFreeDiskSpace()).toString()+"kb");
    buffer.append("\n");
/*   Imprime o tempo de realização da tarefa propriamente dita mais o
      tempo de quando o agente móvel inicia a saida do nó sendo gerenciado.
       Com este ultimo dado é possível verificar quanto tempo o agente
      leva do último nó gerenciado até a estação gerenciadora (NMS).
       Assim é possível identificar qual a influencia da passagem pelo gargalo
      no tempo de resposta total, diminuindo do tempo de resposta total, o
      tempo apresentado no último nó gerenciado.
     */
     long t2 = System.currentTimeMillis() - t1;
     totalTaskTime = new Long(totalTaskTime.longValue() + t2);

     System.out.println("Tempo total de realizacao da tarefa (GET's) = "+new
Long(System.currentTimeMillis() - t1).toString());
     System.out.println("Tempo apos a tarefa (GET's) = "+new
Long(System.currentTimeMillis()).toString());
     System.out.println("\n");

    }
}
```

Anexo B – Código fonte do cliente SNMP

```
import com.adventnet.snmp.snmp2.*;
import com.adventnet.snmp.beans.*;
import java.lang.*;
import java.util.*;

public class snmpAplic
{

// Declaração dos endereços das máquinas que participarão dos experimentos

        private String target1 = "200.200.6.2"; //primeira maquina da rede
        private String target2 = "200.200.6.3"; //segunda maquina da rede
        private String target3 = "200.200.6.4"; //terceira maquina da rede

// Declaração da variável que será utilizada para simular um número maior de
// máquinas no experimento

  protected ArrayList stringTargets = new ArrayList();

  long ttotal = 0;

// Declaração das variáveis da MIB II que serão recuperadas

  protected String inErrors = ".1.3.6.1.2.1.2.2.1.14.1"; //ifInErrors vai até
14, o 1 indica primeira ifInErrors
  protected String sysContact = ".1.3.6.1.2.1.1.4.0";
  protected String sysDescr   = ".1.3.6.1.2.1.1.1.0";

  public snmpAplic(long nEstacoes)
  {
    stringTargets.add(target1);
    stringTargets.add(target2);

// Adicionar quantos targets houverem fixos aqui
//com stringTargets.add("dddd.ddd.dd.d")


// Aumenta o número de estações participantes dos experimentos conforme
// o número solicitado pelo operador do protótipo
  int tamTargets = stringTargets.size();
  if (nEstacoes > stringTargets.size()) {
    for(int i=0; i<= (nEstacoes - tamTargets); i++) {
      stringTargets.add((String)stringTargets.get(i));
    }
  }else {
      nEstacoes = stringTargets.size();
  }

// Declaração dos objetos que serão utilizados para recuperar as informações

    SnmpOID var1 = new SnmpOID(inErrors);
    SnmpOID var2 = new SnmpOID(sysContact);
    SnmpOID var3 = new SnmpOID(sysDescr);

    SnmpTarget target = new SnmpTarget();

    target.setSnmpVersion(SnmpAPI.SNMP_VERSION_1);
```

```
    System.out.println("SNMP version =
"+String.valueOf(target.getSnmpVersion()));

    target.setCommunity("public");
    target.setTargetPort(161);
    target.setTimeout(60);

// Inicializa contador de tempo para que o tempo total de resposta seja
// contabilizado pelo tempo no final do experimento menos o tempo inicializado
// aqui

    long tempo = System.currentTimeMillis();

// Realiza a coleta das informações das estações participantes

    for (int i=0; i < nEstacoes; i++) {
      target.setTargetHost((String)stringTargets.get(i));  // set host, or
other parameters

      target.setSnmpOID(var1);  // or 1.1.0 with standard prefix
      System.out.println();
      System.out.println("--Object Identifier = "+target.getObjectID()+" -
inErrors");
      String result = "";
      String result1 = "";

// Para cada variável solicitada é contado o tempo de recuperação do valor,
// em ttotal

      long t1 = System.currentTimeMillis();
      result1 = target.snmpGet();
     ttotal += (System.currentTimeMillis() - t1);
      if (result1 != null)
      {
       result = result + result1;
        System.out.println("HOST = "+(String)stringTargets.get(i)+", RESULT =
"+result);

        result1 = ";";
        target.setSnmpOID(var2);
       System.out.println("--Object Identifier = "+target.getObjectID()+" -
sysContact");

        t1 = System.currentTimeMillis();
        result1 = target.snmpGet();
       ttotal += (System.currentTimeMillis() - t1);
       if (result1 != null)
        {
          result = result + result1;
           System.out.println("HOST = "+(String)stringTargets.get(i)+",
RESULT = "+result);

           result1 = ";";

           target.setSnmpOID(var3);  // or 1.1.0 with standard prefix
           System.out.println("--Object Identifier = "+target.getObjectID()+"
- sysDescr");

            t1 = System.currentTimeMillis();
           result1 = target.snmpGet();
           ttotal += (System.currentTimeMillis() - t1);
           if (result1 != null)
            {
              result = result + result1;
```

```
                System.out.println("HOST = "+(String)stringTargets.get(i)+",
RESULT = "+result);
              }
             else
              {
                System.out.println("Nao foi possivel pegar a variavel
sysDescr");
                 System.out.println("Tempo total para o target
"+(String)stringTargets.get(i)+" =
"+String.valueOf(System.currentTimeMillis()-tempo)+" milisegundos");
                 System.exit(1);
               }
           }
           else
            {
              System.out.println("Nao foi possivel pegar as variaveis:
sysContact e sysDescr");
               System.out.println("Tempo total para o target
"+(String)stringTargets.get(i)+" =
"+String.valueOf(System.currentTimeMillis()-tempo)+" milisegundos");
              System.exit(1);
             }
       }
       else
       {
        System.out.println("Nao foi possivel pegar a variavel: inErrors");
        System.out.println("Experimento invalidado");
         System.exit(1);
       }
     }

// Mostra os resultados finais
// - Tempo total de resposta
// - Tempo de recuperação das variáveis (execução das tarefas get do SNMP)

     System.out.println();
     System.out.println("Tempo total do experimento =
"+String.valueOf(System.currentTimeMillis() - tempo));
    System.out.println("Tempo total de execucao das tarefas get do SNMP =
"+new Long(ttotal).toString());
  }

  public static void main(String args[])
  {
    long nEstacoes = new Long(args[0]).longValue();
    snmpAplic aplic = new snmpAplic(nEstacoes);
    System.exit(0);
  }

}
```

Anexo C - Código fonte da biblioteca para suporte ao agente móvel com JNI

```
#include <windows.h>
#pragma hdrstop
#include <condefs.h>
#include "jni.h"
#include "memory.hpp"
#include <stdio.h>

#pragma argsused

MEMORYSTATUS  mem;

/*extern "C" __declspec(dllexport) jlong getFreePhysicalMemory();
extern "C" __declspec(dllexport) jlong getFreeVirtualMemory();
extern "C" __declspec(dllexport) jlong getFreeDiskSpace();
*/

JNIEXPORT jlong JNICALL
Java_examples_itinerary_AgenteColetor_getFreePhysicalMemory(JNIEnv *, jobject)
{
        GlobalMemoryStatus(&mem);
        return mem.dwAvailPhys / 1024;
}

JNIEXPORT jlong JNICALL
Java_examples_itinerary_AgenteColetor_getTotalPhysicalMemory(JNIEnv *,
jobject)
{
        GlobalMemoryStatus(&mem);
        return mem.dwTotalPhys / 1024;
}

JNIEXPORT jlong JNICALL
Java_examples_itinerary_AgenteColetor_getFreeVirtualMemory(JNIEnv *, jobject)
{
        GlobalMemoryStatus(&mem);
        return mem.dwAvailVirtual / 1024;
}

JNIEXPORT jlong JNICALL
Java_examples_itinerary_AgenteColetor_getTotalVirtualMemory(JNIEnv *, jobject)
{
        GlobalMemoryStatus(&mem);
        return mem.dwTotalVirtual / 1024;
}

JNIEXPORT jlong JNICALL
Java_examples_itinerary_AgenteColetor_getFreeDiskSpace(JNIEnv *, jobject)
{
    _ULARGE_INTEGER freeBytesAvailable, totalNumberofBytes,
totalNumberFreeBytes;
        __int64 totalFreeKbytes = 0;


  if (GetDiskFreeSpaceEx(NULL, &freeBytesAvailable, &totalNumberofBytes,
&totalNumberFreeBytes))
```

```
        {
                totalFreeKbytes =  totalNumberFreeBytes.HighPart;
                totalFreeKbytes <<= 32;
                totalFreeKbytes |= totalNumberFreeBytes.LowPart;
                totalFreeKbytes /= 1024;

                return (totalFreeKbytes);
        }
        else
                return -1;
}


int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason, void*)
{
        return 1;
}
```