UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Taís Freire da Silva Costa

AVALIAÇÃO ANALÍTICA DO USO DE AGENTES MÓVEIS NA GERÊNCIA DE REDES

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

Orientador: Prof. Dr. Carlos Becker Westphall
Co-orientador: Prof. Dr. Ricardo Felipe Custódio

Florianópolis, Outubro de 1999.

AVALIAÇÃO ANALÍTICA DO USO DE AGENTES MÓVEIS NA GERÊNCIA DE REDES

Taís Freire da Silva Costa

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

Prof. Dr. Fernando Álvaro Ostuni Gauthier
Coordenador

Banca Examinadora:

Prof. Dr. Carlos Becker Westphall
Orientador

Prof. Dr. Ricardo Felipe Custódio
Co-orientador

Prof. Dr. Bernardo Gonçalves Riso

Prof. Dr. João Bosco Mangueira Sobral

Prof. Dr. Luís Carlos Zancanella

"A possibilidade de arriscar é que nos faz homens.
Vôo perfeito no espaço que criamos.
Ninguém decide sobre os passos que evitamos.
Certeza de que não somos pássaros e que voamos.
Tristeza de que não vamos por medo dos caminhos."

Damário da Cruz

AGRADECIMENTOS

Devo confessar que este trabalho foi realizado principalmente a 8 mãos... A cada passo dado, a cada obstáculo vencido e a cada vitória conquistada, sentia as mãos de Deus, da minha mãe e do meu pai dadas às minhas mãos. A vocês, eternas fontes de inspiração, meus sinceros e infinitos "Obrigadas!!!!" O sucesso desta jornada é somente o começo da recompensa que devo a vocês!!!!

Aos meus irmãos e amigos quase irmãos, valeu a força e o apoio necessário para superar o desânimo, as frustrações e sobretudo a saudade da terrinha querida...

Ao meu namorado, agradeço pelo seu amor, seu carinho e seu incentivo que muito me ajudaram a chegar até aqui. TAMMM, and don't you ever forget it!!!

Aos Fluffies, Mary e Lê, o meu eterno agradecimento por todos os momentos felizes que passamos juntos. O que seria de mim aqui sem vocês?? "É hora de dar tchau...", mas levarei sempre comigo todas as coisas que aprendi com vocês. Até breve...

Ao Prof. Westphall e Prof. Custódio, meu sincero e gigantesco "Obrigada!". Confesso que não havia nada mais confortante do que ouvir um "Não te preocupas!!" ao entrar na sua sala, Westphall! Ricardo, sorte sua que você chegou mais tarde, senão eu ia te alugar ainda mais do que eu te aluguei neste pouco tempo em que trabalhamos juntos! Agradeço a atenção, a confiança e o incentivo de vocês, sem os quais essa conquista não seria possível.

À Verinha e Val, verdadeiros anjinhos da guarda, milhões de obrigada por tanto carinho e por tanta simpatia!! Espero vocês atrás do trio elétrico no carnaval da Bahia!

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS, SIGLAS E SÍMBOLOS

AC – Ambiente Computacional
AM – Agentes Móveis
API – Application Programmer Interface
ASDE – Agent Systems Developing Environment
AWT- Abstract Window Toolkit
C/S – Cliente/Servidor
CMIP – Common Management Information Protocol
CoD – Code on Demand
CORBA – Common Object Request Broker
HTML – Hypertext Markup Language
IA – Inteligência Artificial
IETF – Internet Engineering Task Force
ISO – International Organization for Standardization
JDK – Java Development Kit
JVM – Java Virtual Machine
MA – Mobile Agents
MCL – Mobile Code Language
MCS – Mobile Code Systems
MIB – Management Information Base
MIT – Management Information Tree
NMS – Network Management Station
ORB – Object Request Broker
PGP – Pretty Good Privacy
REV – Remote Evaluation
RMI – Remote Method Invocation
RMON – Remote Monitoring
RPC – Remote Procedure Call
SDT – Sistemas Distribuídos Tradicionais
SNMP – Simple Network Management Protocol
SO – Sistema Operacional
SOR – Sistema Operacional de Rede
Tcl – Tool Command Language
UE – Unidade de Execução
WAN – Wide Area Network
WWW – World Wide Web

RESUMO

A dificuldade em realizar o gerenciamento eficiente de redes de computadores a partir de um modelo centralizado, motivou o desenvolvimento deste trabalho. Aqui, é apresentado um esquema de gerência descentralizada baseado nos conceitos da mobilidade de código. A gerência de redes de computadores é uma atividade que sofre impactos diretos com o crescimento do número de recursos e, consequentemente, de informações a serem processadas. A gerência centralizada, solução adotada atualmente, tem se mostrado inflexível e ineficiente, chegando algumas vezes a ser inadequada para o sucesso da própria atividade. Este trabalho apresenta os conceitos fundamentais para a compreensão da mobilidade de código, uma vez que existem ambigüidades e contradições nessa área. É apresentado ainda, uma análise de desempenho de agentes móveis no domínio da gerência de redes através de deduções analíticas.

ABSTRACT

The problems faced with a centralized network management motivated the development of this work. Thus, a descentralized management model is presented based on the concepts of code mobility. Network management is a process which efficiency is strictly associated with the number of resources to be managed and therefore the number of information to be processed. A centralized management, although it is the mostly used solution, has been considered inflexible and unefficient. Sometimes, it is even unappropriate for its main goal. This work presents the fundamental concepts for the comprehension of code mobility, once a fuzzy terminology is commonly found and used in this community. It then presents a quantitative analysis of the performance of mobile agents in the network domain.

1 INTRODUÇÃO

O conceito de mobilidade de código não é tão novo quanto parece ser. Desde o início, a possibilidade de se interligar sistemas computacionais distintos formando uma rede de comunicação de dados propiciou a idéia de mobilidade de código. No princípio, as redes eram constituídas por equipamentos com baixo ou nenhum poder de processamento. Já nessa época, encontravam-se alguns indícios de mobilidade de código com, por exemplo, a freqüente submissão de arquivos de lote (*batch*) para os super-computadores. Com o surgimento da área de sistemas distribuídos, transferências mais ousadas e complexas de informações foram desenvolvidas. Neste contexto, o objetivo principal é prover o suporte para a migração de processos ativos e/ou objetos, mantendo a integridade do seu estado de execução.

A migração de processos é caracterizada pela transferência de um processo que está sendo executado no sistema operacional de uma máquina qualquer para outra máquina distinta, onde a execução deverá ter prosseguimento. Um processo é uma abstração do sistema operacional composto por código, dados e o estado do próprio sistema operacional associado a uma instância de uma aplicação em execução [MILOJICIC99]. Os mecanismos que favorecem este tipo de migração devem gerenciar os *links* e as referências associados ao processo migrante, como por exemplo as variáveis de ambiente, a fim de reconstituir o mesmo "cenário" na máquina destino onde, então, a execução deve continuar a partir do ponto onde foi interrompida. A migração de processos foi, de início, a solução encontrada para o problema do balanceamento de carga entre os nós de uma rede e, adotada como uma solução para implementar a tolerância a falhas.

A migração de objetos, por sua vez, consiste na possibilidade de mover objetos entre espaços de endereços distintos. Esta abordagem permite maior granularidade na transferência de código uma vez que o conceito de objeto abrange desde uma estrutura de dados simples até grandes e complexos objetos.

Estas duas abordagens foram desenvolvidas quando os sistemas distribuídos se limitavam às redes locais. É evidente que tem-se hoje um novo panorama quando se trata das necessidades e problemas da computação distribuída. A experiência obtida com a migração de processos e de objetos favoreceu a concepção de um novo tipo de sistema, o qual ficou conhecido como sistema de código móvel (MCS). Quando comparados às abordagens anteriores, os MCSs apresentam as seguintes características:

- **O escopo da distribuição ultrapassa fronteiras** – os sistemas distribuídos, cuja implementação se dá pela migração de processos e/ou objetos, foram projetados especialmente para as redes locais. Esta topologia geralmente apresenta uma grande largura de banda, uma latência máxima previsível e, quando comparada às redes de longa distância, apresenta maior segurança, ou menos vulnerabilidade. Sistemas de código móvel abrangem ambientes maiores, onde as redes são interconectadas com links de largura de banda e velocidades extremamente contrastantes. Além disso, estas redes geralmente espalham-se por cidades, estados, países e continentes, apresentando diferenças no modo como são gerenciadas, administradas e disponibilizadas para os usuários. Esta diversidade de "culturas" gera impactos diretos na questão da segurança, já que é difícil garantir que o código móvel será executado da forma como foi escrito.
- **O sistema não é transparente quanto à localização dos recursos** – uma das propriedades fundamentais dos sistemas distribuídos tradicionais é a transparência com relação aos recursos. No MCS, as aplicações devem estar cientes do local onde elas estão sendo executadas, até porque o conhecimento desse local pode influenciar na seqüência lógica do programa.
- **A mobilidade não se restringe apenas à tolerância a falhas e à distribuição do processamento** – a migração de processos e objetos, como já foi dito anteriormente, visa equilibrar a carga nos processadores, otimizando assim, a performance geral da aplicação. Os objetivos do MCS vão além disso, providenciando a customização de serviços, modularidade da aplicação, autonomia e o suporte para a computação móvel.

As considerações feitas acima servem de base para a elaboração de um modelo de classificação apresentado neste trabalho. Esta classificação apresenta conceitos, termos e abstrações, no intuito de caracterizar os diversos aspectos do paradigma de mobilidade de código existentes, assinalando semelhanças, diferenças e suas aplicações. A classificação é feita levando-se em conta três aspectos:

- Tecnologia

O aspecto tecnologia envolve as linguagens e ambientes que oferecem suporte para a mobilidade de código. Estes sistemas são consideradas como ferramentas de desenvolvimento na fase de implementação de um sistema. Estas tecnologias serão analisadas no âmbito da mobilidade de código.

- Paradigmas de desenvolvimento

Os paradigmas de desenvolvimento estão relacionados ao estilo usado pelo analista de sistemas para definir a arquitetura de sua aplicação. Este estilo identifica a configuração dos componentes de uma aplicação e estabelece como as interações entre estes componentes serão implementadas. Entre os paradigmas de desenvolvimento, o Cliente-Servidor é atualmente o mais usado.

- Tipos de aplicação

O último aspecto está ligado aos tipos de aplicação. Ele ressalta classes de aplicação (comércio eletrônico, gerência de redes,...) a fim de apresentar situações reais onde a mobilidade de código pode ser utilizada.

A contribuição que este trabalho pretende trazer é a de analisar o paradigma da mobilidade de código, apresentando seus fundamentos, benefícios, problemas e dificuldades, baseando-se em um amplo levantamento bibliográfico. Este cuidadoso estudo propõe uma uniformização dos termos e conceitos do paradigma da mobilidade de código.

A parte prática deste trabalho, realizada no domínio da gerência de redes, apresenta uma análise de desempenho do uso de agentes móveis na gerência de redes.

O trabalho está organizado da seguinte forma:

O capítulo dois apresenta os conceitos relacionados com o paradigma da mobilidade de código. Em seguida, é feita uma comparação entre o paradigma dos sistemas de objetos distribuídos com o paradigma da mobilidade de código, ressaltando as suas semelhanças e diferenças. Ainda neste capítulo é sugerida uma nova terminologia para a área de código móvel.

O capítulo três contém uma classificação dos principais aspectos da mobilidade de código: mobilidade, segurança e comunicação. A taxonomia apresentada é importante para o entendimento das propriedades de sistemas de código móvel.

O capítulo quatro analisa algumas das ferramentas, linguagens e ambientes de desenvolvimento de sistemas de código móvel. Este estudo é feito levando-se em consideração os aspectos discutidos no capítulo anterior: mobilidade, segurança e comunicação.

O capítulo cinco discute as arquiteturas de código móvel, destacando a forma como os componentes de um sistema se comportam em cada arquitetura. A discussão se inicia com a arquitetura Cliente-Servidor e, baseada no grau de mobilidade, estende-se até a arquitetura de Agentes Móveis.

No capítulo seis são apresentadas algumas considerações sobre a aplicabilidade do paradigma de código móvel no desenvolvimento de sistemas distribuídos. Seus benefícios são ressaltados, analisando-se domínios de aplicações reais tais como a gerência de redes, o comércio eletrônico, entre outros.

O capítulo sete descreve as dificuldades da gerência centralizada, demonstrando a necessidade de se criar um mecanismo de gerência descentralizado. Os problemas apresentados neste capítulo constituem importante motivação para o desenvolvimento deste trabalho, assim como de realizações futuras.

O capítulo oito realiza uma análise qualitativa do uso da mobilidade de código na gerência de redes, de acordo com as arquiteturas discutidas no capítulo cinco.

O capítulo nove apresenta a dedução matemática, com a qual foram extraídos os resultados práticos deste trabalho. Esta dedução matemática foi criada de forma a analisar o uso de agentes móveis e do SNMP em redes, cuja estrutura seja hierárquica, ou seja, os segmentos devem estar conectados com roteadores cascateados.

O capítulo dez apresenta três estudos de caso, considerando três topologias diferentes: a gerência remota de uma rede, a gerência local de uma rede e, por fim, a gerência de inter-redes, onde realiza-se a gerência de dois segmentos. Os efeitos analisados são: a variação da latência e da banda passante, o tamanho inicial do agente móvel e o tamanho de três diferentes tarefas de gerenciamento no tempo de resposta.

O capítulo onze traz a conclusão deste trabalho, junto com as dificuldades encontradas e as perspectivas futuras. Finalmente, no capítulo doze encontra-se a bibliografia.

2 FUNDAMENTOS DA MOBILIDADE DE CÓDIGO

Neste capítulo são apresentados conceitos relevantes para a compreensão do paradigma da mobilidade de código.

Existem diversas propostas de Sistemas de Código Móvel (em inglês: Mobile Code Systems) disponíveis tanto comerciais quanto acadêmicas. Este ambiente plural acabou propiciando a ambigüidade de conceitos, abstrações, termos e semântica deste paradigma. A inexistência de padrões dificulta a compreensão, a avaliação e a discussão de técnicas, metodologias, ferramentas e aplicações de MCSs. [BIC96] analisa este problema ao fazer uma comparação entre assuntos e conceitos incompatíveis, classificando-os como Objetos Autônomos. Nesse artigo, mecanismos como o *Remote Procedure Call* (RPC) e o *Remote Evaluation* (REV) são comparados a *Web browsers*, aplicações de correio eletrônico e até a algoritmos de descobrimento de grafos.

A falta de uma terminologia padronizada também gera muitos problemas e falsas expectativas. Alguns sistemas, por exemplo, alegam permitir a transferência do código juntamente com o seu estado de execução. Tal alegação está baseada no fato do programador poder, através de alguns comandos, transferir o valor de algumas das variáveis para o ambiente onde o processo será executado. Neste caso, é tarefa do programador reconstituir o ambiente de execução no destino, sempre que isso for possível. Tal reconstituição pode ser necessária uma vez que, em algumas circunstâncias, todo o estado de execução (contador de instrução, *heap*, etc...) não pode ser enviado junto com o código. O ambiente de execução onde os MCSs serão executados/interpretados deve prover suporte para esta tarefa de reconstituição de estado.

Uma expectativa exagerada é produzida pelo uso excessivo do termo "agentes móveis". O mesmo termo é encontrado, com diferentes significados, na área de inteligência artificial e na área de sistemas distribuídos, chegando algumas vezes a ser empregado com significados contraditórios. Dentro da comunidade de sistemas distribuídos, o

termo "agente móvel" é usado para denotar um componente de software que apresenta propriedades de mobilidade, ou seja, seu código pode ser transferido para diferentes sistemas computacionais desde que entre eles haja alguma comunicação, permanente ou não. A inteligência artificial, por outro lado, utiliza este mesmo termo para definir um componente de software orientado a objetivos, capaz de reagir e executar ações em um ambiente dinâmico [WOOLDRIDGE94]. O agente da IA 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 confundir a noção de agentes inteligentes (própria de IA) com a noção de agente móvel (encontrada na área de sistemas distribuídos). Ao mesclar estes dois conceitos, produz-se a idéia de que todo agente móvel é inteligente e vice-versa.

A confusão se estende ainda na especificação dos requisitos de linguagens para o desenvolvimento de sistemas de agentes móveis (Mobile Code Language – MCL). Em [KNABE95], enumeram-se as características essenciais de uma linguagem que suporte a mobilidade de código. Segundo Knabe, estas linguagens devem prover mecanismos para a manipulação, transmissão, recepção e execução de objetos móveis. Em nenhum momento, ele se refere à necessidade de se oferecer meios para a transferência do estado do objeto móvel. Em contrapartida, outros autores [PEINE97, HARRISON97] acreditam que uma linguagem só é considerada MCL no caso dela suportar tanto a transferência do código quanto do estado do objeto.

A inexistência de padrões é um fato comum e típico em áreas de pesquisas novas e imaturas. Sabe-se porém, que o andamento e o avanço das pesquisas podem prosperar com mais facilidade quando uma base sólida de conceitos e idéias é firmada e aceita como ponto de partida, facilitando futuras descobertas. As pesquisas na área de mobilidade de código não fogem a esta regra. O fator que está inibindo a popularização e a disseminação do paradigma de mobilidade de código não se limita a aspectos técnicos como questões de segurança e performance, embora eles imprescindíveis. É evidente a necessidade de uma base que possibilite a compreensão do multi-facetado

mundo da mobilidade, fornecendo aos pesquisadores um conjunto de conceitos e idéias padrões que sirvam como referências concisas na hora de analisar e comparar as soluções existentes.

2.1 A Essência da Mobilidade de Código

A figura 2.1 ressalta as diferenças entre sistemas distribuídos tradicionais[1] e os sistemas de código móvel. Os sistemas distribuídos tradicionais, (figura 2.1(a)), são compostos por uma camada responsável por oferecer transparência ao usuário, conhecida como *Object Request Broker* (ORB). A arquitetura de um sistema de código móvel, no entanto, é caracterizada pela necessidade de se especificar o local onde cada componente será executado. Este ambiente é chamado de ambiente computacional (AC).

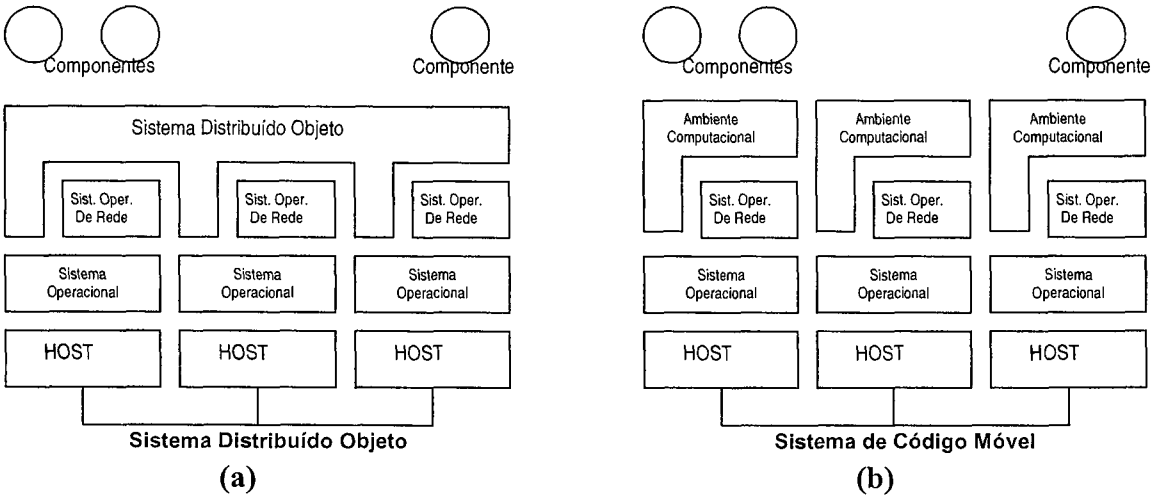


Figura 2.1 – Arquitetura de sistemas distribuídos tradicionais X Arquitetura de sistemas de código móvel.

Analisando a Figura 2.1, tem-se a camada mais inferior, chamada de *Host,* representando o hardware da máquina. A camada logo acima, Sistema Operacional

[1] Na verdade, os sistemas distribuídos que utilizam a camada ORB são chamados de Sistemas Distribuídos Orientados a Objetos. Como este tipo de sistema é atualmente muito conhecido e amplamente utilizado, este trabalho considera-o como um Sistema Distribuído Tradicional (STD).

(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. O *sockets* é um dos serviços mais utilizados desta camada ao se desenvolver sistemas distribuídos. Assim como em uma pilha de protocolos, a camada do SOR utiliza serviços da camada imediatamente inferior a ela, o SO, e assim por diante.

Até a camada SOR, a arquitetura de ambos os paradigmas é idêntica. A última camada, distingue um sistema distribuído tradicional (SDT) de um sistema de código móvel (Mobile Code System - MCS). Na última camada do figura 2.1(a) é oferecida uma plataforma, onde componentes localizados em máquinas distintas são vistos e utilizados pelo usuário como se estivessem na sua própria máquina. Através dessa transparência, não é necessário o conhecimento da topologia da rede onde o sistema é executado. Em um SDT, ao invocar um serviço qualquer, não se pode determinar ao certo qual o nó da rede que irá oferecer e executar este serviço. 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 paradigma 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. Na figura 2.1(b) nota-se esta diferença, onde a camada SDT é substituída por diversos ambientes computacionais (ACs) de acordo com o número de nós da rede que irá participar do sistema. Nesta estrutura, a topologia da rede deixa de ser transparente para o usuário 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. Da mesma forma que a arquitetura dos SDTs e da pilha de protocolos, ela é consumidora dos serviços do SOR para realizar a comunicação entre as máquinas e

consumidora também dos serviços do SO para preparar o código e os dados para uma transferência qualquer.

No intuito de apresentar uma classificação mais aprofundada, os componentes foram subdivididos em unidades 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-threaded.* Entre os recursos, são consideradas as entidades que podem ser compartilhadas entre dois ou mais componentes. Exemplos típicos de recursos são sistema de arquivo, variáveis do sistema operacional, etc. A figura 2.2 ilustra o modelo de uma unidade de execução, composta por um segmento de código (que descreve o comportamento do componente e de um estado).

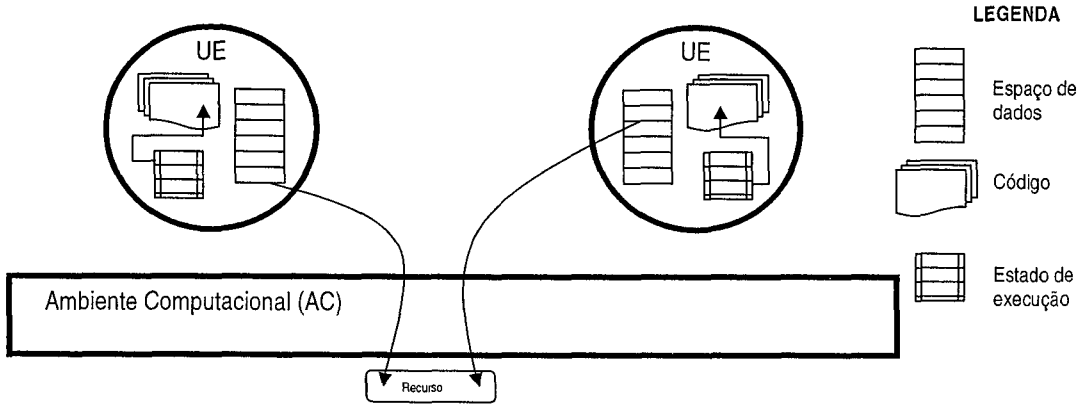


Figura 2.2 – Modelo de uma unidade de execução (UE).

O estado de um componente é composto de um espaço de dados e de um estado de execução. O espaço de dados é o conjunto das referências dos componentes (UE + recursos) que podem ser acessados e utilizados. Este 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.

Uma vez adotados estes conceitos, pode-se, então, classificar um *script,* escrito em uma MCL qualquer, como uma unidade de execução. Seu espaço de dados é composto pelas variáveis usadas pelo interpretador dessa linguagem executado em um nó da rede. O estado de execução desse nó é constituído pelo contador de instrução e pelo *call stack,* mantidos pelo interpretador, juntamente com as outras variáveis referenciadas no *script.*

3 UMA TAXONOMIA DOS ASPECTOS DA MOBILIDADE DE CÓDIGO

Uma vez estabelecidas a terminologia e a conceituação dos seus elementos, é feita uma classificação dos aspectos de mobilidade, segurança e comunicação comuns a qualquer sistema baseado no paradigma de código móvel. Essa classificação é utilizada na análise de algumas das ferramentas, disponíveis para o desenvolvimento de sistemas de código móvel, no final deste capítulo.

3.1 Aspectos da mobilidade

Em sistemas tradicionais, ou seja em sistemas, que não apresentam propriedades de um MCS, cada UE é associada a um único ambiente de execução durante todo o seu ciclo de vida. Além disso, a ligação entre a UE e seu segmento de código é geralmente estática. Inclusive em ambientes que suportam o *link* dinâmico, o código dinamicamente compilado pertence ao AC local. Isto não acontece nos sistemas de código móvel. Nos MCSs, o segmento de código, o estado de execução e o espaço de dados de uma unidade de execução podem ser transferidas para um ambiente computacional disponível na rede de comunicação.

Os MCSs existentes oferecem basicamente duas formas de mobilidade caracterizadas de acordo com os elementos das unidades de execução que podem ser transferidas. Na primeira, estão os MCSs que permitem a mobilidade tanto do código quanto do estado de execução de uma UE para outro ambiente computacional são classificados como MCS forte, possuindo a propriedade de mobilidade forte. Por outro lado, a mobilidade fraca é a propriedade de transferir apenas o código de uma unidade de execução. Na mobilidade fraca pode-se ainda transferir junto com o código alguns dados e variáveis de inicialização, mas em nenhum momento o estado de execução de uma UE é movido.

A mobilidade forte é implementada através de dois mecanismos: migração e clonagem remota. O mecanismo de migração suspende a unidade de execução e transmite-a para o ambiente computacional destino onde então, é retomada a sua execução exatamente

do ponto onde foi interrompida. Esta migração pode ser do tipo pró-ativa ou reativa. Em uma migração do tipo pró-ativa, a unidade de execução decide de maneira autônoma, o momento e o destino da migração. A reação é considerada reativa quando a transferência é invocada por outra unidade de execução desde que essa tenha permissão para este tipo de operação. A migração reativa é comum nos sistemas que delegam o gerenciamento de todos os componentes móveis a uma entidade. O mecanismo de clonagem remota cria uma cópia da unidade de execução em um outro ambiente computacional. Este mecanismo difere da migração uma vez que a unidade de execução original não é desvinculada do ambiente computacional origem. Assim como na migração, existem a clonagem pró-ativa e a reativa.

Ao transferirem o código para o ambiente computacional destino, os mecanismos que implementam a mobilidade fraca associam este código, dinamicamente, a uma unidade de execução ativa no ambiente computacional destino (*code fragment*) ou aproveitam o código 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*).

A figura 3.1 apresenta um resumo da classificação apresentada nesta seção.

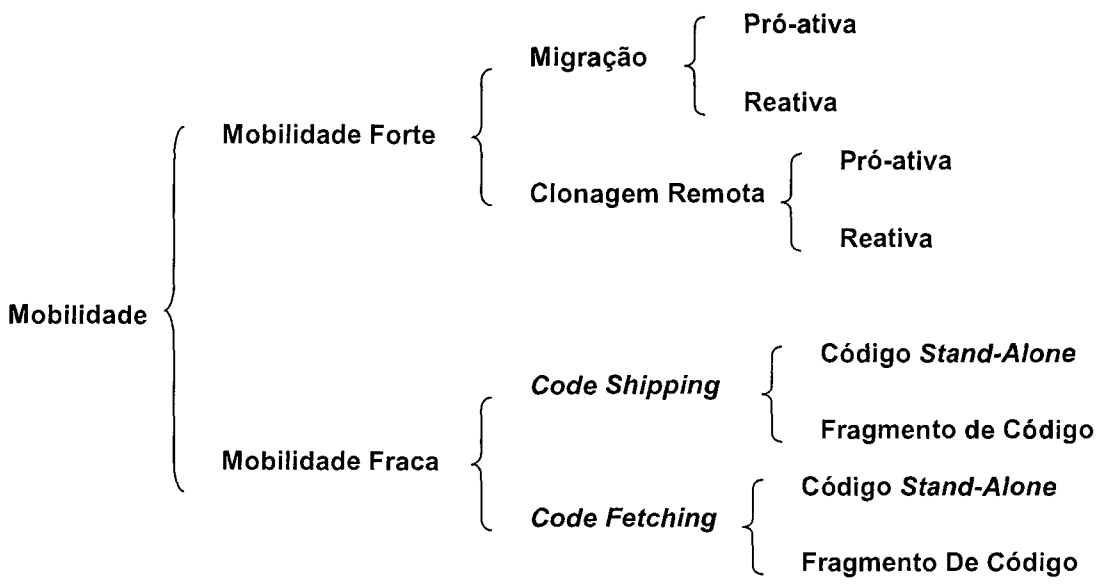


Figura 3.1 – Uma classificação dos mecanismos de mobilidade

3.2 Aspectos de Segurança

A segurança é uma das principais preocupações no uso de sistemas de código móvel, sendo ainda considerada como um dos principais entraves para a popularização e uso de sistemas deste tipo [CHESS95].

MCSs proporcionam um ambiente distribuído, onde aplicações pertencentes a usuários distintos e, portanto, com níveis de confiabilidade distintos, podem ser executadas concorrentemente. Além disso, os ambientes computacionais que abrigam as unidades de execução podem estar localizados em diferentes redes com políticas de segurança, vulnerabilidades e outras características divergentes. O problema se agrava quando o território da aplicação abrange a Internet, herdando assim não só as facilidades, mas também as vulnerabilidades inerentes à grande rede mundial.

Neste vasto cenário, ataques variados podem ser cometidos por qualquer elemento do sistema (ambiente computacional, unidade de execução,...) ou até mesmo por um elemento externo ao sistema, como um usuário espionando o conteúdo dos pacotes

transmitidos na rede. A figura 3.2 ilustra os possíveis pontos de falha de segurança dos sistemas de mobilidade de código; nos pontos marcados com a figura de um raio existem vulnerabilidades que devem ser consideradas.

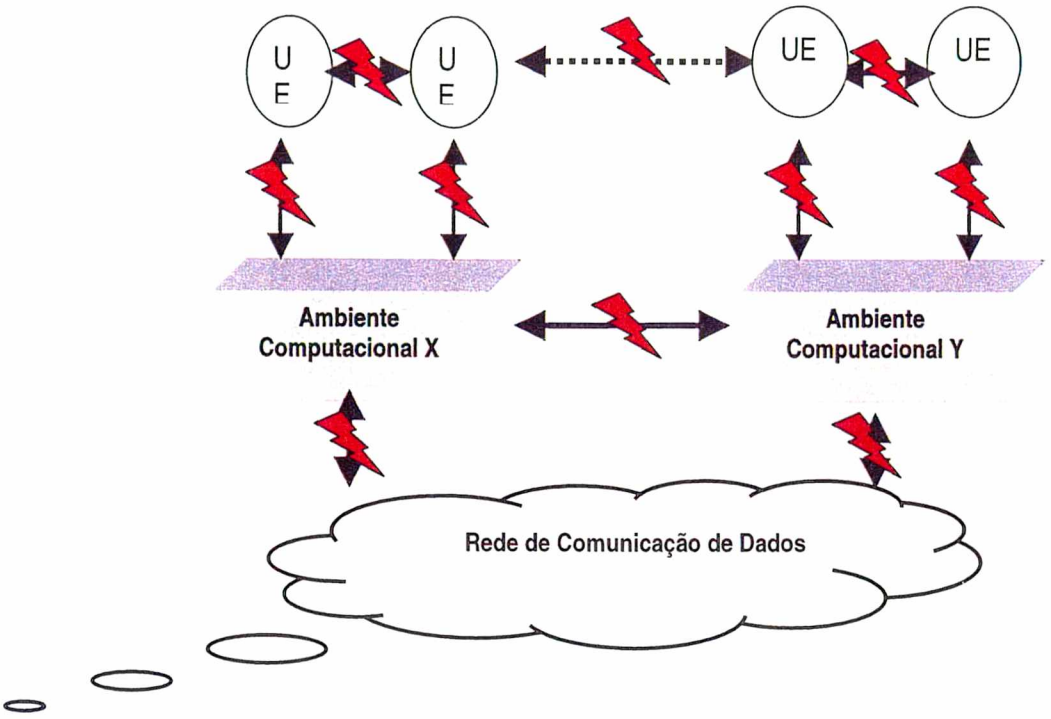


Figura 3.2 – Pontos de falhas de segurança nos sistemas de código móvel

Por serem sistemas intrinsecamente distribuídos, os sistemas de código móvel apresentam todas as vulnerabilidades deste tipo de sistema, além de introduzir novas falhas. Entre os ataques mais comuns dos sistemas distribuídos tradicionais, tem-se:

- Como em todo ambiente de rede, pessoas não-autorizadas podem espionar o tráfego da rede a fim de acessarem ou até modificarem as informações em trânsito;
- aplicações podem tentar obter mais privilégios do esquema de controle de acesso alegando pertencerem a usuários privilegiados do AC destino no intuito de acessarem recursos e informações críticas do sistema;

- um ambiente computacional pode tentar alterar sua identidade, se fazendo passar por outro AC com o objetivo de alojar aplicações alheias e, assim acessar seu código ou dados;
- Uma vez que a aplicação foi recebida por um ambiente computacional, ela pode tentar atacar o seu próprio *host* ou até outros *hosts*. Na primeira hipótese, a aplicação pode tentar acessar informações confidenciais do seu hospedeiro como chaves secretas, arquivos de senhas, etc. Na outra hipótese, a aplicação tira proveito dos recursos do ambiente computacional hospedeiro para realizar ataques em outros *hosts* da rede. Um exemplo comum deste tipo de ataque é o uso do serviço de *sockets* do ambiente computacional para atacar outras máquinas protegidas por *firewalls*;
- Ataques do tipo interrupção de serviços são comuns neste tipo de sistema. Uma unidade de execução maliciosa, por exemplo, pode monopolizar um ou mais recursos de um ambiente computacional ultrapassando seu limite de memória, espaço em disco, etc, fazendo com que a máquina suspenda a execução de todos seus processos;

Além destas falhas de segurança, conhecidas dos sistemas distribuídos em geral, é importante ressaltar uma nova categoria de ataques, introduzida pelos sistemas de código móvel. Considerando que uma unidade de execução percorre diversos ambientes computacionais, estes ACs podem, perfeitamente, manipular o código que está para ser executado no seu processador e, assim, efetuar perigosos ataques. Ao receber um código de uma UE, o ambiente computacional tem poder absoluto sobre ele. Desta forma, o AC pode desobedecer o fluxo do programa modificando sua lógica interna e até alterando o código para que operações ilícitas sejam executadas na próxima máquina a ser visitada. Estes ataques podem anular o objetivo final da aplicação, considerando por exemplo, uma aplicação de compra de CDs composta de uma unidade de execução com a função de se mover pela rede visitando os sites de loja de discos a procura do melhor preço para o pedido do usuário. Um ambiente computacional malicioso pode acessar a estrutura de dados da UE a fim de verificar o melhor preço oferecido até o momento para então fazer sua oferta. Um ataque mais complexo, porém com melhores resultados para o *host* mal intencionado seria o de modificar o fluxo de

execução do código, forçando-o a aceitar o seu preço como o melhor preço, ainda que isso não seja verdade.

O estudo dos aspectos de segurança da mobilidade de código implica na subdivisão de áreas a fim de identificar e isolar os problemas e soluções característicos de cada ambiente. A classificação dos aspectos de segurança, neste trabalho, é feita sob dois prismas: segurança intra-AC que abrange as falhas e mecanismos de segurança dos elementos localizados em um mesmo ambiente computacional; e a segurança inter-AC, onde são apresentados as vulnerabilidades das interações entre elementos remotos. A taxonomia desses aspectos é ilustrada na figura 3.3.

3.2.1 Segurança Inter-AC

Todos as vulnerabilidades e falhas de segurança envolvendo elementos remotos, ou seja, de ambientes computacionais distintos são classificados como aspectos de segurança inter-AC. Entre os aspectos endereçados neste grupo, estão a privacidade, integridade e autenticação da comunicação entre dois ambientes computacionais, uma unidade de execução e um ambiente computacional remoto e duas unidades de execução localizadas em dois ACs distintos.

Ao interagirem, dois ambientes computacionais precisam autenticar-se, entre si, de maneira a ficarem prevenidos de um ataque comum chamado *spoofing* [CHESS95]. Mecanismos de autenticação podem basear-se em simples identificadores, como endereços ou nomes de rede, oferecendo um nível de segurança mínima ou, no caso da segurança ser um fator crítico, exigir que a autenticação seja feita com chaves criptográficas simétricas ou assimétricas, assinaturas digitais, entre outros [SCHNEIER96].

Além da autenticação, é importante implementar mecanismos que ofereçam a integridade e a privacidade das informações transmitidas, principalmente do código das unidades de execução que é freqüentemente transferido de AC em AC. Mecanismos de

integridade são basicamente algoritmos de *checksum*, como o MD5, tornando-se fundamentais para garantir que as informações não sejam modificadas, seja por má intenção ou mesmo por erro no processo de transmissão. Para garantir a privacidade da comunicação, evitando que pessoas não-autorizadas compreendam o conteúdo das mensagens enviadas, tem-se uma gama de soluções divididas entre algoritmos de criptografia simétricos e assimétricos [SCHNEIER96].

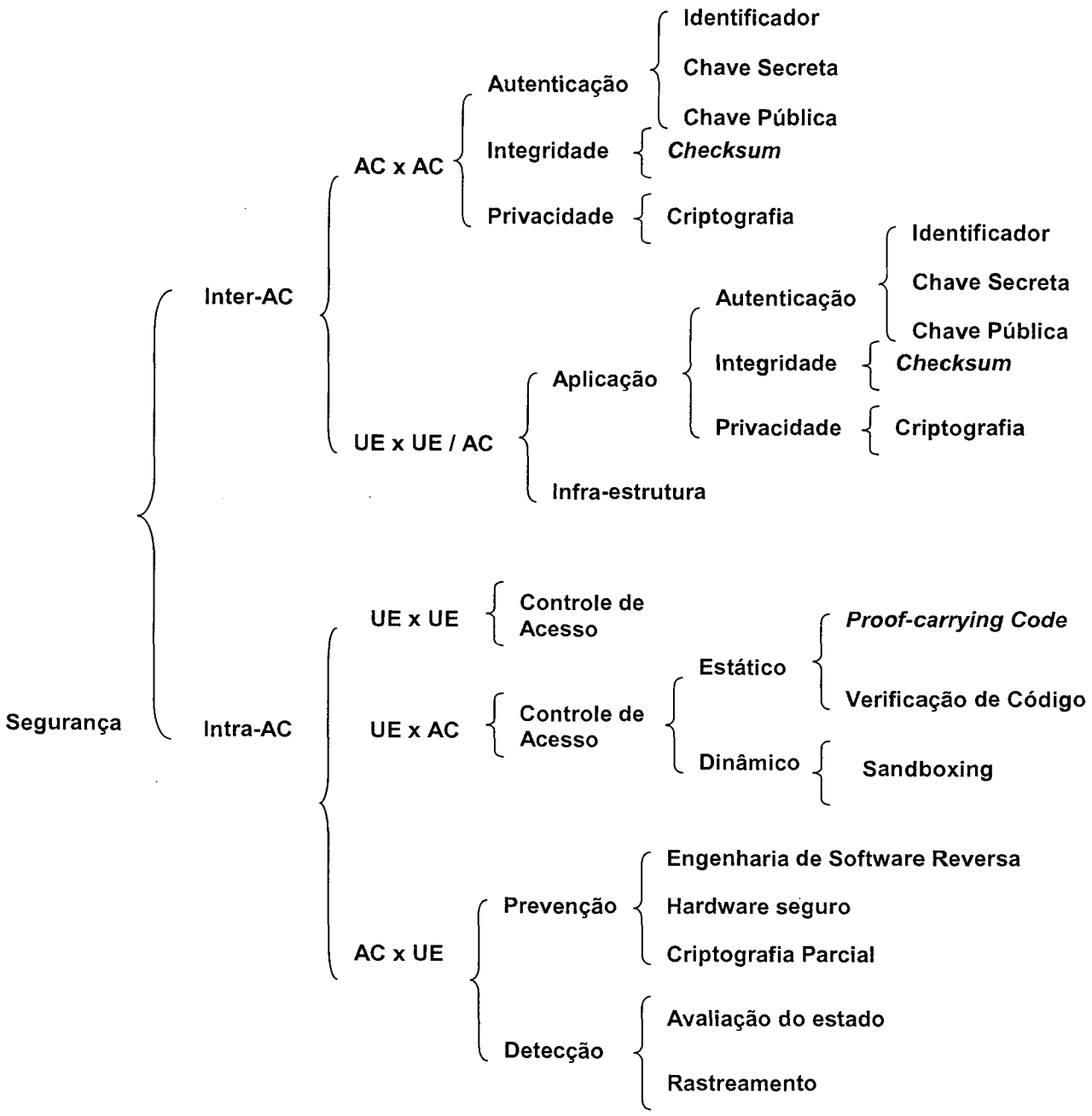


Figura 3.3 - Uma classificação dos aspectos de segurança.

3.2.2 Segurança Intra-AC

A segurança intra-AC relata os problemas e vulnerabilidades presentes no cenário de um único ambiente computacional e seus componentes. Ela abrange aspectos de segurança entre:

- UE x UE – que consiste nos ataques entre duas ou mais unidades de execução localizadas em um mesmo ambiente computacional;
- AC x EU – que se baseia na proteção dos recursos do AC contra os ataques provenientes das unidades de execução;
- EU x AC – que consiste na proteção das unidades de execução contra ataques do ambiente computacional hospedeiro.

A maioria destas falhas de segurança pode ser solucionada com esquemas de autorização, determinando quais as operações que podem ser executadas por um elemento, com base na sua autenticação. Duas ou mais unidades de execução, em um mesmo ambiente computacional, devem estar protegidas uma das outras, de forma a impedir o acesso indevido aos seus dados. Esta proteção é, geralmente, obtida através de esquemas de controle de acesso, que especificam quais os recursos que podem ser utilizados, e em que quantidade cada recurso pode ser explorado. Estas listas de controle de acesso podem ser:

- Estáticas – mecanismos de controle de acesso verificam, antes da UE ser executada, se o seu código respeita a política de controle de acesso estabelecida para os recursos do AC hospedeiro. Se essa política for respeitada, então nenhuma outra verificação será feita durante a execução da UE. O mecanismo do *Proof-Carrying Code* [NECULA97] se baseia neste esquema, uma vez que associa pedaços do código da UE à provas formais que garantem a correção do código. Outro mecanismo, utilizado para a verificação estática do controle de acesso, é constituído de verificadores de código que usam técnicas de prova de teoremas para determinar se o código preserva alguns itens de segurança.

- Dinâmicas – neste esquema, cada UE possui um conjunto de direitos de acesso aos recursos do AC determinados dinamicamente pela sua política de segurança. Cada tentativa de acesso aos recursos do AC no decorrer da execução é interceptada e verificada na lista de controle de acesso local. Desta forma, cada operação no código da UE pode ser aceita ou negada. Este mecanismo é geralmente chamado de execução segura ou, *sandboxing*.

A maior falha nos sistemas de código móvel encontra-se nos perigosos ataques dos ambientes computacionais realizados contra as unidades de execução. O problema consiste na forma como o processamento é realizado, em qualquer *host* hospedeiro ou máquina, sendo ele um ambiente computacional ou não. Para que uma unidade de execução, no caso de sistemas de código móvel, seja processada, o ambiente computacional precisa acessar o código e o estado de execução da UE. Portanto, é muito difícil implementar mecanismos de proteção contra esta vulnerabilidade.

Entre os ataques mais perigosos, encontram-se: modificação do código/dados da UE, leitura de dados confidenciais da UE e a interrupção do serviço, onde o AC simplesmente deixa de oferecer mais determinado serviço às UEs.

A classificação dos mecanismos de proteção para as UEs contra o ambiente computacional é feita inicialmente partindo do seu propósito principal: prevenção ou detecção. Os mecanismos de prevenção tentam tornar mínima a possibilidade de acesso e/ou modificação.

- Mecanismos de prevenção – tentam impedir que acessos e/ou modificações indevidas sejam feitas nas UEs. O mecânismo mais fácil para este objetivo é o uso de um hardware seguro que traz algumas funções criptográficas. Uma outra abordagem menos eficiente é a engenharia de software reversa [HOHL98], onde o código é escrito contrariando-se as regras da engenharia de software: nomes de variáveis sem sentido, códigos redundantes, etc, de forma a tornar a lógica da UE quase que incompreensível para a pessoa que efetuou o ataque. O funcionamento desse mecanismo é limitado, já que, com o tempo, o código poderá ser decodificado.

Mecanismos de criptografia parcial são mais limitados. Eles cifram apenas alguns dados de forma que eles só sejam utilizados por um ambiente computacional escolhido. Estes dados são cifrados com a chave pública do AC destino, tornando-os ilegíveis para qualquer outro AC por onde passar.

- Mecanismos de detecção – tentam descobrir se e quando um ataque foi realizado após a execução da UE. O mecanismo de análise do estado de execução [FARMER96b] estabelece algumas constantes para a UE e após ser executada, verifica o seu estado de execução de forma a descobrir se o valor de alguma dessas constantes foi modificado. Estas constantes podem ser incluídas em algumas funções *hash* tornando o mecanismo ainda mais eficiente. Um outro mecanismo de detecção de ataques é o rastreamento [VIGNA97], ou *tracing*, que através da criptografia e assinaturas digitais permite que qualquer modificação ilegal feita na UE seja detectada.

3.3 Aspectos da comunicação

As primeiras pesquisas e desenvolvimentos na área de sistemas de código móvel [STAMOS90b, FALCONE87] tiveram, como ponto de partida, o mecanismo de *Remote Procedure Call*. A propriedade de se transferir unidades de execução através dos nós de uma rede por si só, exige serviços de comunicação entre os elementos que compõem o sistema. Porém, esta seção se restringe a discursar sobre os aspectos da comunicação entre unidades de execução, uma vez que existem peculiaridades quando comparados aos sistemas distribuídos tradicionais.

Para implementar a comunicação entre as unidades de execução móveis de um sistema, a princípio, tentou-se aproveitar os mecanismos já utilizados e encontrados na literatura. Concluiu-se, porém, que eles não eram adequados para viabilizar as principais propriedades do paradigma de código móvel. A comunicação por troca de mensagens, por exemplo, exige que o destinatário seja identificado por um endereço único associado à máquina onde ele está armazenado. É evidente que no caso de um MCS, onde é possível ao destinatário mudar de hospedeiros, recebendo novos endereços

físicos, este mecanismo não é satisfatório. Outro contexto, que apresenta problemas semelhantes, é o da computação móvel, onde o roteamento de pacotes deve acontecer de forma dinâmica, de acordo com o movimento do *host*. Hoje, já é possível estabelecer comunicação entre uma ou mais unidades de execução, mesmo quando estas se encontram em *hosts* sem uma conexão permanente à rede. Nesta situação, utiliza-se esquemas semelhantes à aplicação de correio eletrônico, *store and forward*, sendo as mensagens armazenadas em um servidor qualquer até que o *host* onde se encontra a UE volte a estar conectado à rede. Com esta característica, sistemas de código móvel tornam-se perfeitos para a computação móvel.

O estudo das soluções para a comunicação em um ambiente móvel ainda está em fase inicial, pois, os esforços até então, estavam voltados para a busca de mecanismos que viabilizassem a mobilidade das unidades de execução. Uma vez que consideráveis avanços foram realizados nesta área, pesquisadores começaram a investigar formas mais complexas de interações entre UEs.

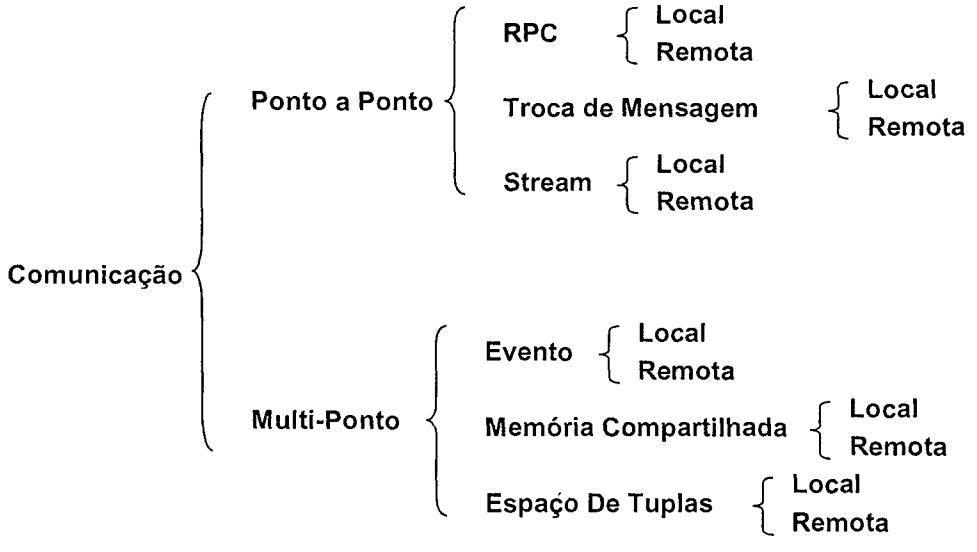


Figura 3.4 – Uma classificação dos mecanismos de comunicação

A classificação dos mecanismos de comunicação presentes nos Sistemas de Código Móvel existentes é feita com base no número de UEs envolvidas e no âmbito onde a comunicação ocorre. Uma comunicação entre duas unidades de execução é dita ser do tipo ponto a ponto, enquanto que comunicações envolvendo mais UEs são chamadas de

multi-ponto. Além dessa classificação, uma comunicação pode ser do tipo local, onde as UEs encontram-se no mesmo AC; ou remota, envolvendo dois ou mais *hosts* da rede. Em uma primeira análise, a comunicação remota pode parecer contraditória ao objetivo principal do paradigma de código móvel que consiste na minimização do tráfego da rede. Este paradigma, porém, como será discutido no decorrer deste trabalho, é uma alternativa aos paradigmas tradicionais e não uma solução revolucionária. Desta forma, a comunicação remota pode ser útil nos MCSs como uma forma de se obter a cooperação entre as unidades de execução ou até mesmo para integrar aplicações MCSs com sistemas distribuídos tradicionais. O gráfico da taxonomia dos aspectos da comunicação deste paradigma é apresentado na figura 3.3.

3.3.1 Mecanismos de comunicação ponto-a-ponto

Os mecanismos de comunicação ponto-a-ponto possibilitam a interação entre duas unidades de execução apenas. Uma das soluções mais utilizadas e conhecidas é o RPC. Tal solução implica usar tanto a invocação de métodos primitivos em objetos remotos ou utilizar esquemas mais robustos como o CORBA. Os MCSs, geralmente, impedem que as unidades de execução se movam durante a execução de uma chamada remota. Mecanismos de RPC são utilizados na comunicação remota e local, de forma a torná-la uniforme.

A comunicação por *stream* possibilita a abertura de um canal de transferência de dados entre duas unidades de execução de maneira a transferir um fluxo contínuo de informações. Assim como no RPC, ela pode ser usada tanto na comunicação remota quanto local, sendo comumente implementada através do serviço de *sockets*.

3.3.2 Mecanismos de comunicação multi-ponto

Os mecanismos de comunicação multi-ponto viabilizam a interação entre mais de duas unidades de execução. Entre os mecanismos mais freqüentemente utilizados encontra-

se o compartilhamento de memória. Neste mecanismo, todas as unidades de execução recebem referências (ponteiros) para as informações que se pretende distribuir, sendo percebidas quaisquer atualizações nas mesmas por todas as partes envolvidas na comunicação. Nos MCSs analisados, este mecanismo é geralmente utilizado para implementar uma comunicação local, apesar que ele pode ser aproveitado também para a comunicação remota.

A comunicação baseada em eventos, por sua vez, funciona de forma semelhante ao serviço de listas de discussões (*mailing-lists*), pois ele estabelece um canal lógico através do qual os eventos são distribuídos e/ou recebidos por todas as unidades de execução que se associaram ao canal. Variações deste mecanismo são encontradas nos MCSs. Muitos deles utilizam-no apenas para realizar a comunicação local, ou seja, os eventos só são distribuídos entre as unidades de execução localizadas no mesmo AC onde foi disparado o evento. Na intenção de viabilizar a comunicação remota com este mecanismo, o usuário encontraria os mesmos problemas apresentados pelo mecanismo da troca de mensagem – o problema da mudança de endereço. Algumas propostas estão surgindo no intuito de solucionar este problema [BAUMANN97, WHITE96].

A outra forma de comunicação multi-ponto encontrada em alguns MCSs é o espaço de tuplas. Neste mecanismo, as unidades de execução trocam informações entre si a partir de tuplas em um espaço de memória compartilhado. Este espaço, chamado espaço de tuplas, pode também ser varrido na tentativa de buscar uma certa informação. O escopo deste mecanismo limita-se a aplicações de pequena escala, uma vez que torna-se difícil gerenciar e manter os espaços de tupla em um ambiente de rede muito extenso.

4 AMBIENTES, LINGUAGENS E SISTEMAS PARA O DESENVOLVIMENTO DE SISTEMAS DE CÓDIGO MÓVEL

Os ambientes, linguagens e sistemas de desenvolvimento de sistemas de código móvel existentes na comunidade acadêmica e na indústria, têm por objetivo prover suporte, facilitar e impulsionar a implementação de sistemas deste tipo. Por motivos já apresentados anteriormente, como a falta de padronização, eles expõem algumas divergências, se analisados sob os aspectos discutidos no capítulo anterior. Nas próximas seções, as linguagens (Mobile Code Language -MCL) e ambientes de desenvolvimento de sistemas de agentes (ASDE) mais conhecidos são apresentados e suas características mais relevantes são discutidas. O levantamento das amostras consideradas, apesar de representar diversas tendências dos MCS, não deve, em nenhum momento, ser considerado exaustivo. Em [MAL], uma listagem completa e atual dos MCSs existentes pode ser encontrada.

4.1 D'Agents [GRAY95]

Desenvolvido no *Dartmouth College* (EUA), o então *Agent Tcl* surgiu, basicamente, de uma extensão do interpretador *Tool Command Language* (Tcl) a fim de desenvolver uma ferramenta que facilitasse a concepção de sistemas de mobilidade forte. Com a pretensão de suportar outras linguagens, como *Phyton*, *Scheme* e Java, ele então passou a se chamar D'Agents.

Neste MCS, uma unidade de execução, também chamada de agente, é um *script* em Tcl. Uma vez que as unidades de execução são executadas em espaços de endereço distintos, elas somente conseguem compartilhar recursos do sistema operacional, como os arquivos.

O ambiente computacional no D'Agents, é composto pela camada do sistema operacional da máquina junto com o interpretador. Neste MCS, as unidades de execução podem mover-se para outros ambientes computacionais, disparar a clonagem

remota ou enviar parte do seu código para um ambiente computacional remoto, através dos comandos **jump**, **fork** e **submit**, respectivamente. Ao migrar para um novo AC, um mecanismo de migração pró-ativo possibilita a transferência não só do código e estado de execução, como também do interpretador Tcl. Para possibilitar a clonagem remota, mecanismos reativos foram disponibilizados. A operação *submit*, por sua vez, é classificada como um mecanismo de *code shipping* para código *stand-alone*, permitindo que uma nova unidade de execução seja criada no ambiente computacional destino. Uma cópia das variáveis associadas ao estado de execução da UE que invocou o comando deve ser transferida explicitamente como parâmetros.

O D'Agents é disponibilizado também na linguagem Java, não somente no intuito de se popularizar, como também para tirar proveito dos recursos (interoperabilidade, segurança,...) desta nova linguagem.

A segurança do D'Agents consiste na possibilidade de criptografar a unidade de execução enquanto esta estiver sendo transferida, provendo assim um certo nível de privacidade no ambiente inter-AC. Ao migrar para um AC qualquer, a unidade de execução pode ser autenticada através do *Pretty Good Privacy* (PGP) [ZIMMERMAN93]. O controle de acesso dos recursos do hospedeiro é estabelecido a partir da identidade autenticada. Para reforçar a segurança destes recursos, a linguagem *SafeTcl* pode ser usada.

A comunicação nos sistemas desenvolvidos com o D'Agents ocorre através de mensagens e de reuniões. Unidades de execução podem trocar mensagens, estando elas em uma mesma máquina ou não. Mecanismos para a comunicação do tipo *stream*, chamado aqui de reunião (*meetings*), estão presentes nesta ferramenta, possibilitando duas unidades de execução estabelecerem um canal de comunicação via *sockets*. Uma unidade de execução pode requisitar uma reunião com outra UE, situada no mesmo hospedeiro ou remotamente. A UE convidada "tem o direito" de aceitar ou rejeitar a conexão.

4.2 Java [SUN94]

A linguagem Java, desenvolvida pela *Sun Microsystems*, trouxe grandes benefícios para a área de sistemas de código móvel. O Java, além de servir como linguagem fonte para o desenvolvimento dos MCSs, é também, por si só, uma poderosa ferramenta para a implementação de aplicações de código móvel.

Com o objetivo original de construir uma linguagem orientada a objetos, de fácil uso, e principalmente portável, os criadores do Java não imaginavam o quanto ela contribuiria para o crescimento da Internet e vice-versa.

O compilador Java transforma o código fonte dos programas escritos nesta linguagem em uma linguagem intermediária independente de plataforma, chamada *bytecode*. O *bytecode* pode então ser interpretado e executado em qualquer plataforma (PC, *Macintosh*, *Workstations*,...) desde que esta contenha o *Java Virtual Machine* (JVM). Ao considerarmos o Java como um MCS, associa-se o conceito de AC ao JVM.

O Java ainda oferece um mecanismo de busca e *link* de classes dinâmicas chamado de *Class Loader*. Este mecanismo é invocado pelo JVM hospedeiro toda vez que o código em execução contém referências para uma classe desconhecida desse ambiente de execução. O *Class Loader* então, busca esta classe, geralmente de um *host* remoto, para que a execução do código prossiga normalmente. Este mecanismo traz somente classes, ou seja, linhas de código, não havendo portanto transferência de variáveis e de estado de execução. De acordo com a classificação apresentada anteriormente neste trabalho, pode-se dizer que o Java suporta a mobilidade fraca utilizando o mecanismo de busca de código (*code fetching*).

Um dos fatores de sucesso do Java, como dito acima, foi a integração desta linguagem com a tecnologia da *World Wide Web*(WWW). A fim de obter maior interatividade e maior riqueza na representação das informações, os browsers incorporaram um JVM permitindo assim que páginas em HTML trouxessem, além de texto, imagens e sons,

códigos executáveis. A combinação de browsers com JVM pode ser considerada como um MCS, já que este suporta aplicações de código móvel. Neste cenário, o JVM de cada browser seria um ambiente computacional (AC) e os applets as unidades de execução (UEs).

Quanto à segurança oferecida neste MCS, o Java traz consigo conceitos como *Security Manager*, *Code Verifier*, entre outros. A unidade de execução, ou applet, não é imediatamente executada ao chegar no ambiente computacional destino; ela é, antes de mais nada, "revistada" pelo módulo do JVM chamado *Code Verifier* para garantir que o código não contém construções perigosas. Ao ser aprovado pelo *Code Verifier*, o código é executado sob supervisão do *Security Manager*.

Como o Java foi desenvolvido principalmente para aplicações em ambientes de rede, diversos mecanismos de comunicação são suportados na linguagem. Bibliotecas de funções (Application Programmer Interface – API) para a comunicação por mensagens, *stream* e até uma plataforma para sistemas distribuídos semelhante ao CORBA estão presentes no Kit de Desenvolvimento Java (JDK). Os mecanismos para implementar a comunicação por eventos e por espaço de tuplas estão ainda em fase de estudo.

4.3 Java Aglets [LANGE97]

O Java Aglets é uma API desenvolvida pelo Laboratório de Pesquisa da IBM em Tóquio para o desenvolvimento de sistemas de agentes móveis. As unidades de execução, chamadas de Aglets, nada mais são do que *threads* que serão interpretados por um *Java Virtual Machine*. O ambiente computacional neste MCS é denominado de contexto.

Um Aglet pode sofrer as seguintes operações durante seu ciclo de vida:

- **create**() – um novo Aglet é criado, onde seu estado é inicializado e sua *thread* principal é executada;

- **clone**() – uma cópia de um Aglet é feita. O novo Aglet terá seu estado de execução idêntico ao estado de execução do Aglet original no momento que a clonagem foi executada;
- **dispatch**() – o Aglet migra para um novo AC, levando consigo seu estado;
- **retract**() – um Aglet que já tenha sido despachado, é trazido de volta junto com seu estado, ao AC de onde ele partiu;
- **deactivate**() – um Aglet é colocado em modo *sleep*, seu estado de execução é armazenado no disco do AC hospedeiro;
- **activate**() – um Aglet que esteja em modo *sleep* é reativado, sendo seu estado restaurado do disco rígido;
- **dispose**() – um Aglet é finalizado, seu estado é perdido para sempre.

Com exceção das operações **create**() e **dispose**(), todas as operações envolvem duplicação, transmissão de objetos pela rede ou ainda o armazenamento do estado de execução em disco. O mecanismo para manipular o estado de execução de um Aglet é o método do Java chamado serialização.

O AC, ou contexto, utiliza a serialização de objetos, disponível no JDK ou com o RMI para exportar o estado de um Aglet em um stream de bytes. Através deste processo, um Aglet e suas referências são escritos como um vetor de bytes e transmitidos pela rede. Ao chegar no contexto destino, o processo inverso ocorre: o estado do Aglet é reconstituído a partir da série de bytes.

O mecanismo de serialização permite apenas que a imagem do *heap* do Aglet seja traduzida em bytes. O *stack* e o contador de instruções das *threads* de um Aglet não podem ser serializadas. Assim, quando um Aglet é despachado, clonado ou desativado, os dados do *stack* e o contador de instrução são perdidos. Essa limitação ocorre devido à arquitetura do *Java Virtual Machine*, uma vez que ela não permite o acesso e a manipulação direta do *stack* com o objetivo de reforçar a segurança da linguagem. Desta maneira, qualquer MCS baseado em Java puro, como o Java Aglets, será incapaz de suportar a mobilidade forte.

A segurança no Java Aglets está intrinsecamente asssociada à segurança da linguagem Java. Todas as vulnerabilidades apresentada pelo Java são consequentemente herdadas pelo Java Aglets. Alguns ataques como a interrupção de serviços, (*denial of service*) que é realizado através da utilização de toda a memória do *host,* são triviais e devem ser cuidadosamente considerados no desenvolvimento de sistemas com esta linguagem.

Os Aglets podem interagir através da troca de mensagens, sendo possível um Aglet invocar operações em outro Aglet. Um mecanismo de comunicação multi-ponto baseado em eventos também é oferecido, permitindo que um Aglet envie mensagens para todos os Aglets que se inscreveram para aquele evento, desde que eles estejam no mesmo contexto.

4.4 MOLE [STRASSER96, BAUMANN97]

Desenvolvida na Universidade de Stuttgart, o MOLE é um API Java para o desenvolvimento de sistemas de agentes móveis restringidos à mobilidade fraca.

As unidades de execução do MOLE, conhecidos como agentes, são objetos Java executados como *threads* pelo JVM. O ambiente computacional neste MCS é chamado de lugar (*place*). Um sistema desenvolvido com o MOLE possui dois tipos de agentes: o agente de usuário e o agente de serviço. Os recursos do ambiente computacional podem ser acessados através de um agente de serviço, que ao contrário dos agentes de usuário, são estacionários.

A migração é feita baseada no conceito de ilha [HOOG91], que consiste no agrupamento de todas as referências feitas pelo agente. As ilhas não conseguem manter referências a objetos localizados em outro ambiente computacional na migração. Referências a objetos remotos recebem o valor nulo em uma migração.

O projeto MOLE enfatizou bastante a comunicação entre agentes cooperativos, onde, de acordo com [STRASSER96], compreende três conceitos básicos: grupos de agentes,

sessões e eventos. Os grupos de agentes podem ser criados através da idéia de crachás, os quais consistuem em identificadores que agentes podem apresentar ou retirar, durante a vida da sua aplicação. Desta forma, os valores destes crachás podem ser usados em uma aplicação a fim de estabelecer a comunicação entre agentes com os mesmos objetivos. Uma sessão corresponde ao estabelecimento da comunicação entre agentes, comunicação esta que é feita pela troca de mensagem ou pelo RPC. Durante o estabelecimento de uma sessão, dois parâmetros são passados: o endereço do ambiente computacional do agente que quer se comunicar (*placeId*) e o identificador do destinatário (*PeerQualifier*), podendo este último se referir a um único agente ou a um identificador de grupo de agentes.

A segurança do MOLE, por enquanto, se limita aos mecanismos de segurança oferecidos pela linguagem Java.

4.5 TACOMA [JOHANSEN95a, JOHANSEN95b]

O TACOMA (*Tromso and Cornell Mobile Agents*) assemelha-se ao D'Agents, uma vez que ele também é considerado uma extensão da linguagem Tcl para prover suporte à mobilidade fraca. As unidades de execução neste MCS também são chamadas de agentes e são implementadas em Tcl.

Neste MCS, a mobilidade oferecida é do tipo *code shipping* de código *stand-alone*, permitindo também que dados de inicialização do agente móvel sejam transferidos junto com ele, encapsulados em uma estrutura de dados chamada de *briefcase* (mala). Os recursos do ambiente computacional hospedeiro referenciados pelo agente são chamados de *cabinets* (armários). Estás referências não são automaticamente transferidas pelo MCS, ficando sob responsabilidade do programador.

No TACOMA, a segurança é implementada, basicamente, por dois mecanismos. Cada ambiente computacional pode estabelecer uma lista de *hosts* dos quais ele aceitará agentes, limitando o intercâmbio de agentes apenas aos *hosts* confiáveis. A segurança dos recursos do ambiente computacional contra ataques dos agentes é obtida através do

confinamento da execução do agente a uma parte reservada do sistema de arquivos da máquina. Além destes dois mecanismos, nenhum outro suporte à segurança é oferecido no TACOMA.

No TACOMA, os agentes se comunicam através dos *cabinets*, ou seja, espaço de dados compartilhado no sistema operacional do *host* hospedeiro, portanto envolvendo apenas agentes localizados no mesmo ambiente computacional. Para compartilhar informações contidas no *briefcase* de cada agente, a comunicação deve ser realizada durante a inicialização dos agentes envolvidos.

O TACOMA suporta aplicações desenvolvidas com agentes em diversas linguagens como Tcl, Python, Scheme, Perl e C. Os agentes escritos em linguagens interpretadas são transferidos em seu código original, sem a necessidade de traduções. Códigos compilados, como o C, muitas vezes precisam sofrer alguns ajustes no AC destino para então ser compilado e executado.

4.6 Odyssey

Desenvolvido pela General Magic, o Odyssey é considerado o sucessor do Telescript [WHITE96].

O Telescript foi o primeiro MCS da General Magic, sendo conhecido, principalmente, por seu poderoso suporte à segurança. Apesar da sua eficiência, o Telescript foi retirado do mercado, pois a General Magic concluiu que a popularização do Java estava dificultando o sucesso do Telescript.

O Odyssey foi, então criado como um conjunto de bibliotecas de classes do Java no intuito de substituir o Telescript. Neste MCS, tanto as unidades de execução quanto os ambientes computacionais são *threads* em Java, implementados através da instanciação das classes oferecidas pelo sistema.

Como o Java não oferece mecanismos para a transferência do estado de execução de um *thread*, a mobilidade do Odyssey é fraca. Ao ser transferida, a unidade de execução é reiniciada no destino. O Telescript, em todo o caso, possibilitava a migração do código e do seu estado de execução, implementando a mobilidade forte, com o comando ***GO***.

A comunicação no Odyssey é obtida através da definição de uma interface de transporte. O Odyssey suporta apenas o RMI, sendo que estudos para a implementação nas plataformas CORBA e DCOM estão sendo realizados. O RMI do Odyssey exigiu uma adaptação do *Security Manager*, já que o *Security Manager* original proíbe o estabelecimento de conexões *Sockets* e interação com a classe AWT (Abstract Window Toolkit).

A segurança do Odyssey consiste apenas na segurança oferecida pelo Java. A *General Magic* pretende incorporar outros mecanismos de segurança do Telescript ao Odyssey.

5 ARQUITETURAS

Neste capítulo, algumas arquiteturas, que suportam sistemas de código móvel, serão discutidas. É importante ressaltar que as arquiteturas apresentadas não são dependentes de um único MCS, podendo ainda serem implementadas com outras tecnologias que não necessariamente a tecnologia de código móvel [GHEZZI97].

Para o desenvolvimento de aplicações de código móvel, existe uma variedade de arquiteturas, que irá determinar os elementos que compõem o sistema e a forma pela qual eles interagem. Essas interações, como em todo sistema distribuído, são complexas, exigindo um cuidado maior no tratamento de questões como latência, performance, tolerância a falhas, etc...

Antes de analisarmos as arquiteturas, os seguintes conceitos devem ser estabelecidos visando uma compreensão concisa: componentes, interação e sites.

Componentes, são os elementos que constituem uma arquitetura de software. Eles podem ainda ser classificados em:

- componentes de código - que representam as regras para a execução de uma computação qualquer;
- componentes de recursos - simbolizando os dados ou recursos invocados durante a computação e,
- componentes computacionais - são os elementos ativos responsáveis por processar a computação e gerenciar os recursos.

As interações são eventos de comunicação envolvendo dois ou mais componentes de qualquer tipo, como por exemplo uma mensagem trocada entre duas unidades de execução.

O conceito de *sites* intuitivamente implica na noção de local. Os *sites* hospedam componentes além de suportar a execução de componentes computacionais, exigindo

recursos de processamento. As interações entre componentes localizados em um mesmo *site* são consideradas menos complexas do que as interações entre componentes remotos, uma vez que não envolvem os serviços de rede. É necessário compreender que uma computação somente ocorre quando todos os componentes de recurso e computacionais referenciados pelo componente de código estiverem localizados no mesmo site.

As arquiteturas descritas a seguir, são analisadas em termos das interações ocorridas entre componentes, considerando a coordenação e localização destes, ao prestar um serviço qualquer. O cenário é montado de forma que um componente computacional A, localizado no site S_A, requisita a execução de um serviço qualquer. Em se tratando de um sistema distribuído, assume-se também a existência de um outro site S_B, com componente computacional B, necessário para a realização do serviço. As arquiteturas são analisadas levando-se em conta o local dos componentes antes e depois da prestação do serviço, o componente computacional responsável pela execução do serviço e, finalmente, o local onde a computação é efetuada.

A tabela 5.1 resume, com base nestes fatores, as quatro arquiteturas apresentadas neste capítulo. Cada linha da tabela 5.1 representa uma arquitetura e a localização dos componentes antes e após a realização do serviço. Os componentes computacionais foram sublinhados de maneira a indicar o local onde o processamento ocorreu. Assim, na arquitetura de código sob demanda por exemplo, o *site* S_A ao requisitar um serviço qualquer, detém o componente computacional, representado na tabela pela letra A, e os recursos necessários para efetuar o serviço. O código contendo as instruções do serviço no entanto, está localizado em outro *site*, S_B, responsável por enviar estas informações para qualquer outro site que requisite este serviço. Uma vez que o código é recebido pelo *site* A, o serviço é então executado nele, usufruindo dos recursos e dos processadores do próprio *site* A.

Tabela 5-1 – Arquiteturas para sistemas de código móvel.

Arquitetura	Antes do Serviço		Depois do Serviço	
	S_A	S_B	S_A	S_B
Cliente/Servidor	A	B Código Recurso	A	**B** Código Recurso
Execução Remota	A Código	B Recurso	A	**B** Código Recurso
Código sob Demanda	A Recurso	B Código	**A** Código Recurso	B Código
Agentes Móveis	A Código Recurso	B Recurso	-	**B** Código Recurso

A seguir, as arquiteturas mencionadas na tabela 5.1 serão mais detalhadas.

5.1 Cliente-Servidor (C/S)

A arquitetura cliente-servidor é muito conhecida e utilizada, atualmente. Neste paradigma, um componente computacional B, localizado no *site* S_B, recebe requisições para prestação de alguns serviços, sendo chamado de servidor. O código para prestação do serviço assim como os recursos utilizados para tal, estão localizados no *site* S_B. Um componente A, chamado de cliente, localizado em um *site* S_A, requisita a execução de um serviço qualquer, desde que oferecido pelo servidor, interagindo com o componente B. Ao receber esta requisição, B executa o código referente à prestação do serviço, utilizando seus próprios recursos. Normalmente, a realização de um serviço fornece algum tipo de resultado que é enviado para o cliente que fez a requisição.

5.2 Remote Evaluation (REV)

No paradigma REV, um componente A, localizado no *site* S_A, detém o código para a prestação de um serviço, porém carece dos recursos necessários para a sua execução.

Os recursos necessários estão localizados no *site* S_B. Conseqüentemente, A envia o código para o componente computacional B, remoto, onde ele é executado usufruindo dos recursos próprios do *site* S_B. Os resultados da prestação do serviço são enviados de volta para A.

5.3 Código sob Demanda (CoD)

Nesta arquitetura, o componente A possui todos recursos necessários para a prestação do serviço, faltando-lhe porém, as instruções de como os recursos devem ser usados para que o serviço possa ser realizado. Desta forma, A interage com o componente B, localizado no site S_B, requisitando o código para tal serviço. Ao receber o código de B, o componente A executa-o utilizando seus próprios recursos.

5.4 Agentes Móveis (MA)

O paradigma de agentes móveis difere de todos os outros paradigmas citados anteriormente, pois a prestação de um serviço nem sempre envolve uma interação entre dois componentes. Nesta arquitetura, o componente A, localizado inicialmente no *site* S_A, detém o código e os recursos necessários para a execução do serviço. O serviço pode ainda necessitar de recursos extras localizados em um *site* remoto, S_B. Neste caso, o componente de código migra para o *site* S_B, transferindo consigo o código, estado e recursos móveis, como dados intermediários. Ao chegar no *site* S_B, B completa o serviço usufruindo dos recursos disponibilizados neste site. Assim, enquanto que no REV e no CoD o foco é na transferência de código entre os componentes, no paradigma de agentes móveis, o elemento a ser transferido é composto do seu código, estado e alguns recursos necessários para a realização da tarefa.

5.5 Uma análise das arquiteturas

O paradigma de código móvel mostrou-se eficiente no desenvolvimento de sistemas distribuídos [GHEZZI97]. A maioria dos paradigmas tradicionais é estático em relação à estrutura do código de seus componentes e do local onde estes são executados. Nestes

paradigmas, uma vez que um componente é criado, nenhuma modificação é feita no seu código nem na definição do local de execução. Além disso, os tipos de interação e suas complexidades são fixos, uma vez que é impossível mover os componentes a fim de otimizar as interações durante a vida do sistema.

O paradigma de código móvel é caracterizado pela transitoriedade dos componentes de um sistema. As arquiteturas REV e CoD, por exemplo, permitem a execução de código em processadores remotos, tornando as interações entre os componentes envolvidos na computação, simples interações locais. O paradigma CoD, ainda oferece uma certa flexibilidade aos componentes ao permitir que eles busquem códigos em outros sites, estendendo seu comportamento.

6 APLICABILIDADE DA MOBILIDADE DE CÓDIGO

Como o paradigma de código móvel ainda é insipiente, o domínio de suas ferramentas é pequeno, principalmente, quando comparado ao domínio das aplicações de sistemas distribuídos tradicionais. Esta diferença, porém, tende a diminuir. O interesse na mobilidade de código é motivado não somente pelos grandes benefícios que ele pode oferecer às aplicações distribuídas tradicionais, mas também por possibilitar o desenvolvimento de aplicações inusitadas.

Este capítulo discute alguns destes benefícios prometidos pelo paradigma de código móvel e apresenta alguns domínios de aplicação adequados para o uso da mobilidade de código.

6.1 Os benefícios do paradigma de código móvel

Um dos maiores benefícios do paradigma de código móvel é a possibilidade de alcançar a customização dos serviços oferecidos em uma aplicação. Em sistemas distribuídos tradicionais baseados no paradigma Cliente/Servidor, os serviços oferecidos são estabelecidos a *priori*, disponibilizados através de uma interface estática. É comum que essas interfaces ou estes serviços não sejam adequados para suprir as necessidades dos usuários, seja a curto ou a longo prazo. A solução viável neste caso é modificar a estrutura do servidor, incluindo as novas funcionalidades, aumentando assim a sua complexidade sem, no entanto, ganhar flexibilidade.

A possibilidade de realizar a execução remota de código contribui para o ganho de flexibilidade do servidor, sem afetar sua complexidade. Neste caso, o servidor precisa oferecer apenas um conjunto de serviços simples que não precisarão ser constantemente modificados. Os serviços, na verdade, são confeccionados pelo cliente, a fim de alcançar a customização de acordo com suas necessidades específicas.

Benefícios deste paradigma também estão presentes em uma área da engenharia de software conhecida como manutenção e atualização de sistemas. Em um ambiente distribuído, o processo de instalação ou atualização da aplicação normalmente é realizado localmente em cada *host*, sendo necessário a intervenção humana. O tempo necessário para uma instalação ou atualização é proporcional ao número de *hosts* da rede e à distância física entre estes *hosts*. Este processo pode e deve ser automatizado. Alguns *softwares*, como o *Real Player* da *Real Networks*[2], já oferecem mecanismos de *upgrade* automático, buscando o código necessário da Internet, bastando ao usuário apenas a autorização para este processo.

O paradigma de código móvel oferece recursos para uma automação mais sofisticada nos processos de instalação e atualização. Um esquema poderia ser montado onde procedimentos de instalação seriam codificados em uma unidade de execução, cujo itinerário incluísse todos os *hosts* de uma rede. Desta forma, o programa poderia analisar as características de cada máquina e realizar, de acordo com as características encontradas, a configuração e instalação mais adequadas.

Um esquema mais ousado envolve situações onde a aplicação precisa de novas funcionalidades, como por exemplo, no caso de um novo módulo da aplicação ser necessário ao clicar certo botão. As técnicas tradicionais exigem que esse novo módulo seja adicionado em cada *host*, através de uma reinstalação do *software* ou de um mecanismo conhecido como *patching*. Este processo geralmente é trabalhoso e longo, além de sobrecarregar as máquinas com *softwares* pesados sem muitas vezes haver necessidade. O novo módulo requisitado pela aplicação em um site S_A, poderá nunca ser utilizado por um site S_B. Com isso, a possibilidade de se realizar o *link* dinâmico *on demand* do código que traz a implementação da nova funcionalidade oferece inúmeros benefícios tanto para o usuário quanto para o programador do sistema:

- a atualização da aplicação estaria centralizada em um servidor repositório de código onde a última versão está sempre disponível, evitando inconsistências.
- as atualizações não serão mais feitas pelo programador de forma pró-ativa em cada máquina; ao invés disso, estes *upgrades* serão exigidos pela própria aplicação de

[2] http://www.real.com

forma reativa, ficando ela com a tarefa de buscar automaticamente o novo código no repositório central.

- as aplicações só são estendidas se houver necessidade para tanto.

Esta noção de autonomia, característica do paradigma de código móvel, é muito útil também nas aplicações executadas em uma rede heterogênea com links de velocidade e performance distintos. Estas diferenças devem ser consideradas desde a fase de projeto da aplicação. Pesquisas recentes no desenvolvimento de sistemas de computação móvel ressaltaram que ambientes com conexões de baixa largura de banda e de baixa confiabilidade exigem novas metodologias de projeto [FORMAN94, IMIELINSKY94]. Estes sistemas devem tratar peculiaridades como as freqüentes desconexões e o redirecionamento de tráfego para os links de alta velocidade.

O paradigma Cliente/Servidor tenta resolver estes problemas aumentando o nível da granularidade dos serviços oferecidos pelo servidor. Assim, uma única interação entre cliente e servidor pode disparar uma série de operações que serão executadas no servidor, de forma a poupar o meio de comunicação. Com isso, o número de interações entre o cliente e o servidor pode ser reduzido a fim de poupar o meio. Por outro lado, a complexidade do servidor aumenta, uma vez que ele deve oferecer mecanismos de tolerância a falhas, como por exemplo se, a conexão cair em algum momento.

O paradigma de código móvel, por sua vez, promete benefícios para a operação de sistemas em ambientes com essas características. Os serviços a serem executados em um servidor remoto, conectado à rede por meios físicos de baixa velocidade de transmissão e de baixa confiabilidade, são escritos em um programa. O meio é utilizado uma única vez para o envio do programa ao servidor e outra vez para a transmissão dos resultados para o cliente, não sendo necessária uma conexão permanente durante o processamento do serviço.

A autonomia dos componentes de uma aplicação propicia um esquema de tolerância a falhas mais eficiente. Em sistemas Cliente/Servidor, o estado do processamento é distribuído entre o cliente e o servidor. O programa cliente é composto por comandos

que são executados localmente junto com comandos que invocam serviços em um servidor remoto. O servidor, por sua vez, recebe cópias de alguns dados que pertencem ao ambiente do cliente, podendo, eventualmente, enviar resultados ao cliente que irão sobrepor esses dados. Esta estrutura apresenta graves problemas com o surgimento de falhas parciais, uma vez que é quase impossível determinar onde e quando intervir para reconstruir um estado consistente. Um componente autônomo encapsula todo o estado de uma computação distribuída e, assim, pode ser facilmente verificado e reconstituído, sem o conhecimento dos estados das partes remotas.

6.2 Domínios de Aplicação

O domínio de aplicação para o paradigma de código móvel é vasto, porém, ainda pouco explorado. Existem diversas áreas onde, se aplicado este novo paradigma, ganhos em performance e eficiência podem ser alcançados. A seguir, os principais domínios de aplicação que podem ser explorados por este novo paradigma são apresentados.

a) Consulta de informação distribuída

As aplicações de consulta de informações distribuídas realizam consultas em fontes de informação dispersas na rede. As fontes de informação a serem "visitadas" podem ser definidas de modo estático ou determinadas dinamicamente durante a execução da consulta. Este é um domínio que engloba uma vasta gama de aplicações. Os objetivos da aplicação podem variar, por exemplo, do conjunto de publicações de um acadêmico até o endereço de todos os *hosts* conectados em uma rede qualquer.

A mobilidade de código neste domínio pode trazer grandes benefícios. Ao transferir o código para a fonte de informação, por exemplo, é possível obter ganhos em eficiência uma vez que a consulta e filtragem das informações serão realizada localmente na própria base de informação. Este domínio de aplicação foi um dos primeiros domínios explorados pelo paradigma de agentes móveis, reforçando a idéia da mobilidade de código.

b) Serviços de telecomunicações

Atividades de gerenciamento e contabilização de alguns serviços de telecomunicações como a vídeo-conferência, *video-on-demand* e tele-conferência, exigem mecanismos para reconfiguração e customização dinâmicas. A mobilidade de código pode facilitar a implementação destas atividades ao desenvolver componentes com funções de gerência de *setup*, sinalização, etc. Uma outra aplicação neste domínio que deve aproveitar os benefícios da mobilidade de código é a computação móvel. Neste caso, componentes autônomos podem prover suporte para usuários com conexões temporárias [GRAY97].

c) Controle e monitoração de equipamentos

O controle remoto de equipamentos visa facilitar as atividades de configuração de equipamentos de uma rede e de monitoração dos seus comportamentos.

Este domínio engloba, principalmente, as aplicações de gerência de redes e de controle de processos industriais. A maioria dessas aplicações foi desenvolvida com base no paradigma Cliente/Servidor, onde a monitoração é feita através do mecanismo chamado *polling* e a configuração é limitada a um conjunto de serviços ou primitivas. Esta abordagem apresentou uma série de problemas comprometendo a própria atividade de gerência [YEMINI96].

A mobilidade de código foi vista como a solução para alcançar uma gerência mais eficiente. A idéia é desenvolver componentes que serão alocados nos próprios equipamentos a serem gerenciados, favorecendo ganhos em performance e flexibilidade [GOLDSZMIDT95].

d) Workflow

Aplicações do tipo *Workflow* possibilitam a cooperação de pessoas e a integração de ferramentas na execução de um processo. O fluxo de controle dessas aplicações é

determinado por um roteiro de atividades delegadas a certos elementos (grupos de funcionários, departamento, etc...) e que devem ser executadas dentro do período estabelecido no roteiro.

Estas aplicações são, geralmente, modeladas representando-se as atividades como entidades autônomas, que no decorrer do tempo, fluem entre as entidades envolvidas no *workflow*. A mobilidade de código parece ser perfeitamente adequada para o desenvolvimento deste tipo de aplicação, já que o próprio paradigma sugere a transferência de entidades. Além disso, o paradigma de código móvel permite incluir o estado das atividades nas entidades móveis. Uma aplicação de revisão de textos, por exemplo, poderia compor as entidades como o texto a ser revisado junto com o estado das revisões já realizadas e a próxima operação a ser executada. Em [CAI96] apresenta-se uma aplicação prática deste domínio.

e) Redes Ativas

A idéia de redes ativas [TENNENHOUSE97, YEMINI96] surgiu com o objetivo de oferecer mais flexibilidade à rede de comunicação de dados, permitindo que ela seja "programada" dinamicamente de acordo com as diferentes necessidades das aplicações que nela trafegam. Apesar de pesquisadores, como [BHATTACHARJEE96], apresentarem esta idéia sem associá-la à idéia de mobilidade de código, a maioria das aplicações segue este paradigma. Em [TENNENHOUSE97], encontra-se um levantamento de pesquisas feitas na área de redes ativas, encontrando resultados extremos: o *switch* programável e a abordagem de cápsula. O conceito de *switch* programável pode ser considerado como uma instanciação da arquitetura CoD, implementando a rede ativa com o *link* dinâmico de código. No outro extremo, tem-se o conceito de cápsula, onde a proposta é incluir em todo pacote transmitido na rede, instruções de como o pacote deve ser processado ao chegar em cada nó.

Em [WETHERAL97], um protocolo para o uso dinâmico da rede é apresentado, utilizando conceitos da arquitetura CoD. Redes ativas poderiam ser compostas de roteadores "inteligentes", onde pacotes de protocolos desconhecidos e não-suportados

pelo roteador poderiam ser roteados, desde que o pacote trouxesse o código com o protocolo necessário.

f) Comércio Eletrônico

Aplicações de comércio eletrônico permitem a realização de negócios em um ambiente de rede. O ambiente dessa aplicação geralmente é composto por diversas entidades comerciais independentes e competidores entre si. Assim, a segurança é um dos fatores mais importantes para o sucesso e aceitação da aplicação.

A realização de um negócio eletrônico envolve negociações com entidades remotas e pode exigir informações que são atualizadas constantemente, como os índices da bolsa de valores. Neste contexto, é importante padronizar o comportamento das partes envolvidas na tentativa de formular um protocolo de negociação e permitir que novas entidades entrem no negócio.

Hoje, o comércio eletrônico é muito comum na Internet, através do paradigma CoD. A mobilidade de código promete uma reestruturação do comércio eletrônico, uma vez que os componentes "consumidores" farão o papel de pessoas, pesquisando o melhor preço antes de efetuar a compra. O MCL Telescript [WHITE94] foi concebido, principalmente, visando o desenvolvimento de aplicações para o comércio eletrônico.

7 CÓDIGO MÓVEL NA GERÊNCIA DE REDES

O mundo da gerência de redes hoje pode ser dividido em duas facções predominantes: o *Simple Network Management Protocol* (SNMP) [CASE90] proposto pela IETF, e o *Common Management Information Protocol* (CMIP). Ambos adotam uma arquitetura similar, apesar de divergirem quanto ao modo de operação [JONES94]. O SNMP e o CMIP implementam uma gerência centralizada, baseada na arquitetura Cliente/Servidor.

A inteligência do sistema de gerenciamento concentra-se em uma ou mais estações de gerenciamento (Network Management Station - NMS) responsáveis por interagir com agentes[3] que armazenam as informações dos diversos recursos da rede: roteadores, *workstations*, impressoras, etc. Estes agentes de gerência são entidades computacionais com o propósito de prover uma interface padronizada para o acesso às informações dos recursos onde eles estão localizados. Cada agente armazena os dados do recurso correspondente em uma base de informação de estrutura hierárquica chamada de *Management Information Base* (MIB). A comunicação segue um protocolo de gerência, que especifica o formato do pacote para cada operação.

7.1 As desvantagens da centralização

Na gerência SNMP e na CMIP, todo o processamento das informações relacionadas à gerência é realizada na NMS. As operações disponíveis para a NMS acessar a MIB são poucas. No SNMP, por exemplo, a NMS só dispõe de duas operações para acessar a MIB: consultar (*GET*) ou atualizar (*SET*). Com isso, a gerência de redes torna-se limitada e ineficiente.

Em [YEMINI93], Yemini afirma que a centralização limita a escalabilidade da gerência de redes. A partir do momento em que a rede cresce, a NMS precisa interagir com um maior número de recursos, assim como armazenar e processar uma crescente quantidade de dados. Isso implica, muitas vezes, na necessidade de se comprar mais hardware,

elevando os custos da atividade. Sem um hardware mais poderoso, a performance baixa, implicando em uma gerência lenta ou ineficiente. A gerência centralizada pode, ainda, tornar-se inviável para lidar com o crescente tamanho da rede, e, contribuir para o seu mau funcionamento, ao gerar maior quantidade de tráfego, por exemplo.

Além de questões de performance e balanceamento de carga, a gerência centralizada pode gerar problemas de tráfego e congestionamento dos meios de transmissão. Se todas as informações fluem para a NMS, é provável que os segmentos de rede dos quais a NMS está ligado, sustente tráfegos muito altos estando sujeitos à colisões. Em alguns casos, uma segunda rede é montada somente para trafegar dados relativos à gerência. Esta solução é cara e gera problemas de recursividade: para gerenciar esta nova rede, cria-se outra rede, e assim por diante.

O maior problema ocorre quando a rede apresenta congestionamento. Neste período, a atividade de gerência é fundamental para contornar o problema rapidamente. Porém, ao contrário do que se espera, a gerência centralizada implica em:

a) a NMS aumenta o número de interações com os recursos para descobrir o motivo do congestionamento, aumentando assim o congestionamento;
b) o acesso aos 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.

Problemas semelhantes a estes já foram diagnosticados, há algum tempo atrás, nos algoritmos de roteamento. Inicialmente, eles eram estruturados de forma centralizada, existindo um centro de controle de roteamento responsável por direcionar todos os pacotes da rede. Este centro concentrava as funções para descobrir a topologia da rede, e a partir dela, calcular a tabela de roteamento para cada roteador da rede e, ainda, armazenar esta tabela nos respectivos roteadores. O alto tráfego gerado ao redor do centro de controle de roteamento e a dificuldade de acessar os equipamentos mais

[3] O termo "agente" no contexto da gerência de redes é usado para caracterizar programas convencionais;

distantes levaram ao desenvolvimento de mecanismos de roteamento distribuídos, os quais são utilizados até hoje. De forma análoga, a centralização deve ser evitada na gerência de redes, distribuindo as atividades de controle e de diagnóstico pela rede sempre que for possível.

As desvantagens e problemas da gerência centralizada já foram reconhecidos pela comunidade de gerência de redes e tanto a IETF quanto a ISO introduziram aspectos de descentralização nas especificações mais recentes de suas arquiteturas de gerenciamento.

A notificação de eventos foi o primeiro passo para a descentralização da gerência, uma vez que possibilitou aos agentes enviarem mensagens, chamadas de *traps*, à NMS sem que este tenha requisitado. Como a idéia da IETF é de manter os agentes simples, estes só conseguem notificar alguns eventos que exijam 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 resetado, etc. A proposta da ISO especifica agentes mais complexos, capazes de executar processamentos um pouco mais refinados nos dados coletados, gerando assim um esquema de notificação mais complexa baseado em um maior número de parâmetros. As notificações no jargão OSI são chamadas de eventos ou alarmes.

Independentemente da arquitetura de gerenciamento utilizada, SNMP ou CMIP, os agentes ficam responsáveis apenas pela notificação de alguns eventos, não possuindo meios de realizar operações em resposta a eles. Ou seja, o conhecimento ainda reside na NMS.

Outra tentativa de distribuir a gerência partiu da ISO, ao implementar a operação *ACTION*, onde a NMS pode invocar ações em alguns objetos armazenados na *Management Information Tree* (MIT). A execução destas ações pode reduzir tanto a carga de processamento da NMS quanto o tráfego associado à atividade de gerência. A

---

não havendo qualquer relação com o conceito de agentes móveis ou agentes inteligentes.

IETF não provê nenhum mecanismo direto que permita a chamada remota de operações nos recursos.

O SNMPv2 busca a descentralização ao criar o conceito de *proxy agents*. Um agente *proxy* é responsável por gerenciar um conjunto de recursos. Um sistema de gerenciamento pode conter diversos *proxy agents*, cada um agindo como cliente em relação aos recursos gerenciáveis e como servidor, ao reportar informações para a NMS.

Além dos *proxy agents*, o IETF propôs outra abordagem ainda mais descentralizada, chamada de *Remote Monitoring* (RMON). 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. Desta forma, o RMON é considerado como uma abordagem orientada ao tráfego uma vez que as conclusões são tiradas dos pacotes que trafegam na rede, e não da inspeção de cada recurso. O protocolo SNMP é utilizado pelos agentes RMON na comunicação com a NMS. Parte do processamento das informações de gerência é realizada localmente nos *probes* RMON. Eles conseguem detectar falhas, problemas e identificar situações críticas como no caso de congestionamento da rede. Semelhante ao conceito de alarme no CMIP, o agente RMON é configurado de forma a analisar amostras periódicas de parâmetros relevantes sempre observando que os limites aceitáveis não sejam ultrapassados. O agente RMON ainda pode realizar a compressão semântica de dados antes de enviá-los à NMS, contribuindo para a redução de tráfego na rede.

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

7.2 Gerenciamento por delegação

As pesquisas do uso da mobilidade de código em aplicações de gerência de redes é, geralmente, classificada na área de gerenciamento por delegação (*Management by Delegation*). Esta área surgiu muito antes da popularização dos MCS na Internet. A proposta original [YEMINI91] assinala uma arquitetura onde os recursos da rede buscam e atualizam seus *scripts* de gerenciamento de forma dinâmica, usando uma combinação de REV e Cliente/Servidor. Esta arquitetura de gerência ainda inclui um protocolo de gerência, agentes e um ambiente de *run-time* para cada recurso. Ao invés de trocar simples mensagens, a estação gerenciadora pode enviar programas contendo uma série de operações aos recursos gerenciados, delegando a eles a execução da tarefa. Esta execução é assíncrona, liberando a estação gerenciadora para executar outras tarefas, além de introduzir um certo grau de paralelismo no sistema de gerenciamento. Uma vez que o envio de programas aos recursos é dinâmico, a atualização de código é fácil e simples. Isto torna o sistema mais flexível, pois a estação gerenciadora pode customizar 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*. Esta proposta, porém, pode ser considerada como a precursora das idéias discutidas neste trabalho. Ela está em vias de padronização pela IETF e pela ISO.

8 A MOBILIDADE DE CÓDIGO NA GERÊNCIA DE REDES: UMA ANÁLISE QUALITATIVA

A mobilidade de código está sendo considerada como uma solução para otimizar a gerência de redes, uma vez que ela oferece um nível de flexibilidade necessária para lidar com os problemas mencionados nos capítulos anteriores. [GOLDSZMIDT95] afirma que as funções de gerenciamento devem se deslocar para os dados, ao invés de mover os dados para as funções.

Diante da diversidade de abordagens para se implementar a gerência baseada na mobilidade de código, neste capítulo é feita uma análise qualitativa e informal sobre as vantagens de cada uma destas abordagens.

8.1 Código sob Demanda: Flexibilidade

No esquema de gerenciamento da IETF, a estrutura da MIB é implementada a partir do código do agente e não pode ser alterada durante a execução (*run-time*). Esta falta de flexibilidade acaba por trazer algumas restrições no processo de notificação de eventos. A definição de eventos significativos está atrelada à estrutura da MIB não permitindo que novos eventos customizados 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.

A extensão de um agente de gerência pode ser feita através de alguns artifícios do paradigma Cliente-Servidor. As funcionalidades do CORBA podem ser aproveitadas pelos agentes de gerência para reforçar propriedades de segurança, tolerância a falhas, etc. A ISO, através de uma especificação mais refinada, permite que a MIT seja alterada dinamicamente.

A adição de complexidade aos agentes é, geralmente, feita de forma estática, incluindo as novas funcionalidades em um grupo de agentes, ainda que a maior parte deles jamais

necessite usufruir destas funções. Ainda que necessária, a função pode ser utilizada com baixa freqüência, como no caso de procedimento para resolver períodos de congestionamento nos agentes monitorando portas de roteadores de pouca utilização. Assim, a adição de complexidades nos agentes nem sempre compensa. A própria natureza dinâmica da gerência de redes requer agentes customizados dinamicamente.

A adição de complexidade nos agentes sempre foi evitada pela comunidade de gerência de redes. A argumentação apresentada pelos pesquisadores é a de que um agente deve 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, torna o processo de gerência mais lento, podendo ainda gerar mais tráfego na rede.

A arquitetura do 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. Toda vez que seja necessário 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 "reconstituídos" 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.

Esta arquitetura pode então ser empregada para atingir uma gerência "adaptativa" ou "*on-demand*", já que as primitivas de gerenciamento só serão invocadas no recurso quando realmente necessitado. A arquitetura CoD é muito útil também na gerência de configuração, onde as atualizações e a configuração dos *softwares* da rede podem ser feitos automaticamente.

8.2 *Remote Evaluation*: distribuição de processos e economia da banda passante

Na gerência de redes tradicional, os dados são transferidos dos agentes para a estação gerenciadora onde, então, ocorre o processamento das informações. Este esquema visa manter os agentes compactos e simples, deixando o código pesado para a NMS executar.

Para ressaltar os benefícios do paradigma REV, considera-se a situação onde a NMS necessita procurar um certo valor em uma tabela qualquer. De acordo com o paradigma tradicional da gerência de rede, esta operação pode ser implementada de duas formas:

1. A tabela inteira, é transferida para a NMS para que a busca seja feita. Desta maneira, quanto maior a tabela, mais largura de banda será consumida.
2. A função de busca pode ser implementada no próprio agente. A inclusão de funções nos agentes de gerência nem sempre compensa, já que os agentes podem ficar muito "pesados" com funções que serão invocadas esporadicamente.

O paradigma REV pode ser usado como uma solução para este problema. Para a consulta em tabelas, por exemplo, 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 ser executada localmente. Após a execução, somente o resultado esperado é enviado de volta à NMS, alcançando assim a compressão dos dados. Com este paradigma, a economia da banda é maximizada proporcionalmente ao tamanho das tabelas e das funções de consulta. Além da compressão semântica, parte do processamento é distribuído entre os agentes, liberando a NMS.

A solução obtida com a REV implica, ainda, em uma maior modularidade na arquitetura de gerenciamento. Diante da possibilidade de se construir funções e procedimentos de gerência customizados, o gerente pode criar uma espécie de biblioteca de gerência, utilizando-a sempre que preciso.

8.3 Agentes móveis: autonomia e novas funcionalidades

No paradigma REV, cada função deve ser invocada explicitamente pela NMS em cada recurso. No paradigma de agentes móveis, a NMS inicializa apenas uma função e delega a esta um caminho a ser percorrido, de recurso em recurso. A diferença básica entre o REV e o MA está na capacidade de um agente móvel reter seu estado de execução, independente de quantos *hosts* sejam utilizados na execução.

Além de aumentar a flexibilidade e facilitar a distribuição de carga, como nos outros paradigmas, este esquema ainda reduz o tráfego nas proximidades da NMS, já que as consultas realizadas nos recursos são feitas localmente, sem a necessidade de se trocar mensagens pela rede.

Ao visitar um certo número de recursos, o agente móvel aplica também a compressão semântica de dados. A compressão neste caso é dita compressão semântica global dos dados, ao contrário dos paradigmas REV e COD onde a compressão é limitada a apenas um recurso. Apesar disso, não se pode afirmar que esse fato implica na otimização do tráfego da rede, pois o tamanho do agente móvel a trafegar na rede é diretamente proporcional ao tamanho das informações coletadas e do número de recursos visitados.

Ainda que o tamanho do agente móvel seja grande o suficiente de forma a desconsiderar as vantagens alcançadas com a compressão semântica dos dados, existem situações onde, ainda assim, é recomendável o uso deste paradigma.

Como exemplo, considera-se um cenário onde os recursos a serem gerenciados estão localizados em uma rede local diferente da rede onde encontra-se a estação gerenciadora; sendo estas duas redes interligadas por um link não-confiável e extremamente lento. Neste caso, a gerência tradicional centralizada seria quando não inviável, ineficiente e lenta, pois este esquema implica em uma troca de mensagens excessiva entre a NMS e os recursos gerenciados (figura 8.1). Assim, o processo de gerência estaria sujeito a uma elevada latência e riscos de falhas parciais.

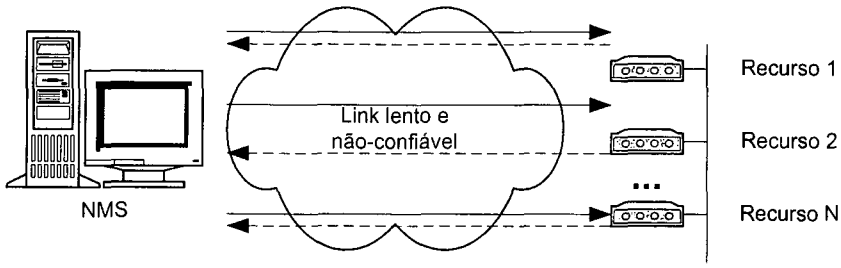


Figura 8.1 – Gerência de redes baseada no paradigma Cliente/Servidor.

Com o paradigma dos agentes móveis, o componente móvel é injetado na rede onde estão os recursos, coletando assim as informações de todos os recursos, sem a necessidade de haver qualquer comunicação com a NMS (figura 8.2). Uma vez coletada todas as informações, o agente móvel retornaria à estação de gerenciamento. Ainda que o tamanho do agente móvel aumente de forma considerável, a gerência é realizada com mais eficiência e com maior rapidez.

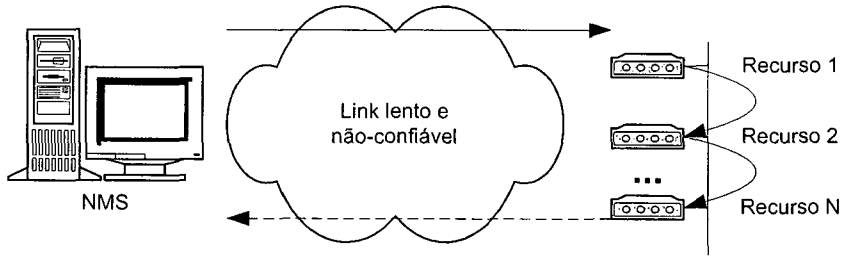


Figura 8.2 – Gerência de redes baseada no paradigma de agentes móveis.

O paradigma MA é mais eficaz, também, na detecção e recuperação de *loops* de roteamento. Se o gerente da rede percebe a existência de um *loop* em alguma das tabelas dos roteadores, ele pode enviar um agente móvel para examiná-las. O agente tenta se aproximar do roteador causador do problema enquanto armazena a lista dos roteadores já visitados. Se, durante este caminho, ele passa por um roteador já visitado, então um *loop* é detectado e o agente pode então tentar:

1. Interromper o *loop* escolhendo uma outra interface randomicamente ou
2. Solucionar o *loop* através, por exemplo, da desativação da interface do *loop* e monitorando o comportamento do protocolo de roteamento.

A princípio, a NMS pode, por si só, detectar o *loop* lendo as tabelas de roteamento. Porém, uma vez que o roteamento apresenta problemas, é provável que a NMS não possua conectividade com alguns roteadores, tornando a informação coletada incompleta ou desatualizada.

8.4 Solução heterogênea: maior viabilidade

A escassez de equipamentos com suporte aos MCSs ainda é o maior obstáculo para a popularização da mobilidade de código na gerência de redes. Contudo, as vantagens oferecidas por esta arquitetura na área de redes desperta grande interesse de pesquisadores e organizações. Muitas pesquisas estão sendo realizadas na tentativa de se criar arquiteturas e plataformas que incluam o conceito de mobilidade de código no próprio equipamento de rede [TENNENHOUSE97]. Paralelamente a isso, alguns fabricantes já estão anunciando a comercialização de equipamentos com suporte à linguagem Java implementada no próprio *chip* visando performance.

É fato que existe uma tendência dos fabricantes adicionarem mais poder de processamento direcionado para a gerência de redes nos seus produtos. Os *switches* com suporte ao RMON, por exemplo, são equipados com um processador capaz de analisar os pacotes além de outras funcionalidades do RMON que descentralizam o processo de gerência até um certo nível. Essa tendência de estabelecer uma gerência distribuída no entanto, poderá gerar problemas quanto a inexistência de um padrão que garanta a interoperabilidade entre os equipamentos de fabricantes distintos. Este problema já é evidente na comunidade da mobilidade de código, onde a possibilidade de se trabalhar com linguagens interpretadas dificulta a interoperabilidade entre ambientes e sistemas escritos com diferentes MCSs.

Diante da tecnologia atualmente disponível, a proposta deste trabalho é de implementar uma gerência heterogênea de redes, ou seja, criar um cenário onde agentes móveis e agentes SNMP realizarão o gerenciamento dos recursos. Esta solução, apesar de não ser a mais eficiente, é mais factível e fácil de implementar, já que não exige que todos os recursos de uma rede ofereçam o suporte a MCSs.

Os resultados obtidos com esta experiência irão servir para testar a eficiência da gerência de redes com a mobilidade de código e, no caso destes serem positivos, poderão ainda contribuir para despertar o interesse dos fabricantes em oferecer o suporte a MCSs em seus produtos.

9 UM MODELO ANALÍTICO PARA A AVALIAÇÃO DE DESEMPENHO DE AGENTES MÓVEIS NA GERÊNCIA DE REDES

Como toda tecnologia emergente, o paradigma de agentes móveis ainda não foi amplamente utilizado, nem formalmente testado, para que seja considerado uma alternativa mais eficiente na atividade da gerência de redes. Nos capítulos anteriores, afirmou-se que este paradigma oferece uma série de benefícios quando comparado ao SNMP. A fim de investigar e comparar o desempenho destas duas tecnologias na atividade de gerência de redes, um modelo matemático é proposto e utilizado no desenvolvimento de estudos de caso.

Uma avaliação de desempenho é realizada através de métricas, que constituem critérios para a comparação do desempenho entre sistemas ou entre diferentes situações a que um sistema pode ser submetido. A métrica utilizada neste trabalho é o tempo de resposta resultante da obtenção de variáveis SNMP de diversos recursos distribuídos por uma ou mais redes. As variáveis utilizadas nesta análise matemática e suas respectivas unidades de medida são descritas na tabela 9.1:

Tabela 9-1 – Variáveis utilizadas no modelo matemático proposto.

Variável	Unidade de medida	Descrição
L_i	Segundos	Latência da rede i
B_i	Mbps	Largura de banda da rede i
k	Bits	Tamanho do código do agente
P	Bits	Tamanho do pedido
R	Bits	Tamanho da resposta
n_u	Inteiro	Número total de elementos na rede u
q	Inteiro	Número total de sub-redes

9.1 Caracterização do ambiente modelado

O cenário modelado neste trabalho consiste na existência de uma estação central, gerenciadora (NMS), responsável por realizar consultas em um grupo de elementos gerenciáveis distribuídos em um ou mais segmentos de rede.

A análise da performance é feita com base no cálculo do tempo total necessário para realizar determinada operação de gerência.

9.1.1 Modelo SNMP

No caso do SNMP, esse tempo é a diferença entre dois instantes: o instante em que a primeira mensagem de requisição foi enviada ao primeiro elemento gerenciável e o instante do recebimento da última mensagem de resposta do último elemento gerenciável pela NMS. A figura 9.1 ilustra um recurso sendo gerenciado através do SNMP. Desta forma, o tempo de resposta proveniente da gerência de n recursos com o SNMP, é obtido multiplicando o tempo de resposta da gerência de um recurso pelo número de elementos que se pretende gerenciar. De acordo com a figura 9.1 e as variáveis descritas na tabela 9.1, tem-se:

Tempo de Repostas$_{SNMP}$ = (L_0+(P/B_0)+ L_1+(P/B_1)+ L_1+(R/B_1)+ L_0+(R/B_0))*n

Ou seja, o tempo de resposta do SNMP é calculado a partir da latência de cada segmento e da divisão do tamanho da informação trafegada no segmento pela sua banda passante.

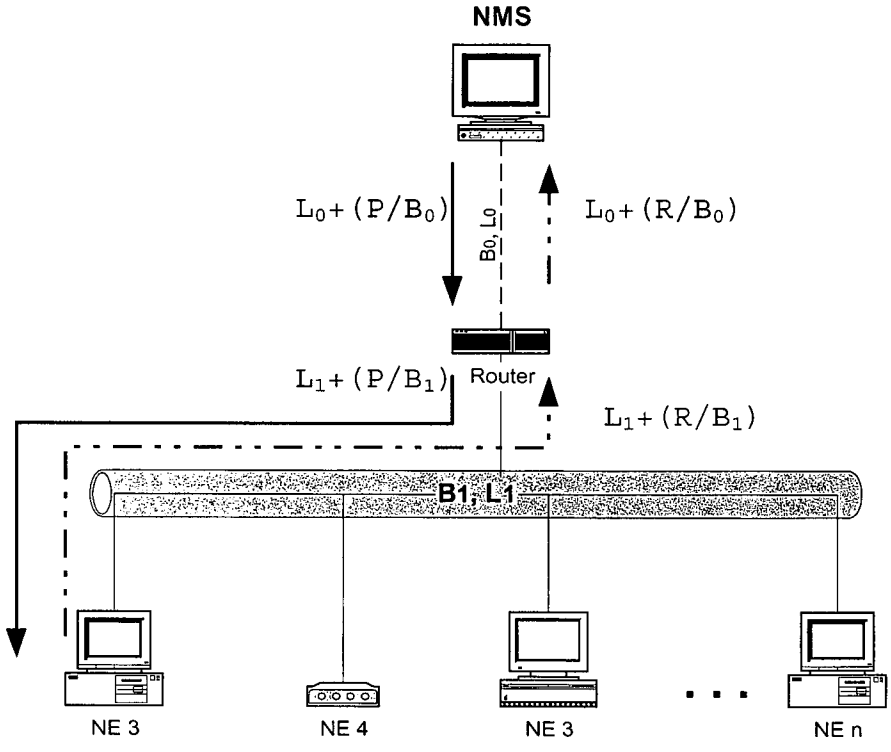


Figura 9-1 – Modelo de gerência do SNMP.

Este modelo foi refinado e concluiu-se que o tempo de resposta, resultante das consultas às MIBs dos diversos recursos distribuídos, por um ou mais segmentos através do protocolo centralizado SNMP, é obtido com a equação 1:

$$t_{snmp} := \sum_{z=1}^{q} n_z \left(\left(\sum_{i=0}^{z} (2 L_i) \right) + \left(\sum_{i=0}^{z} \frac{P+R}{B_i} \right) \right)$$

(1)

9.1.2 Modelo Agentes Móveis

Quando a mesma operação de gerência é feita com o paradigma de agentes móveis, o tempo é obtido a partir da realização de uma série de passos:

1. O agente móvel parte da NMS localizada na rede j para o primeiro recurso da rede (j+1);
2. O agente móvel percorre todos os recursos da rede (j+1);
3. O agente móvel retorna do último recurso do último segmento de rede (q) à NMS.

O passo 2 deve ser expandido no caso de haver mais recursos a gerenciar em outros segmentos de rede. Neste caso, o agente móvel terá que percorrer cada segmento de rede, visitando todos os seus recursos. Desta forma, o passo 2 envolve os seguintes "sub-passos":

2.1 O agente móvel, após gerenciar todos os elementos da rede (j), parte do último elemento gerenciado para o primeiro elemento gerenciável da rede j+1;
2.2 O agente móvel percorre todos os elementos da rede (j+1);

Estes sub-passos devem ser executados tantas vezes quanto forem o número de sub-redes a serem percorridas. O modelo de gerência do agente móvel é apresentado na figura 9.2.

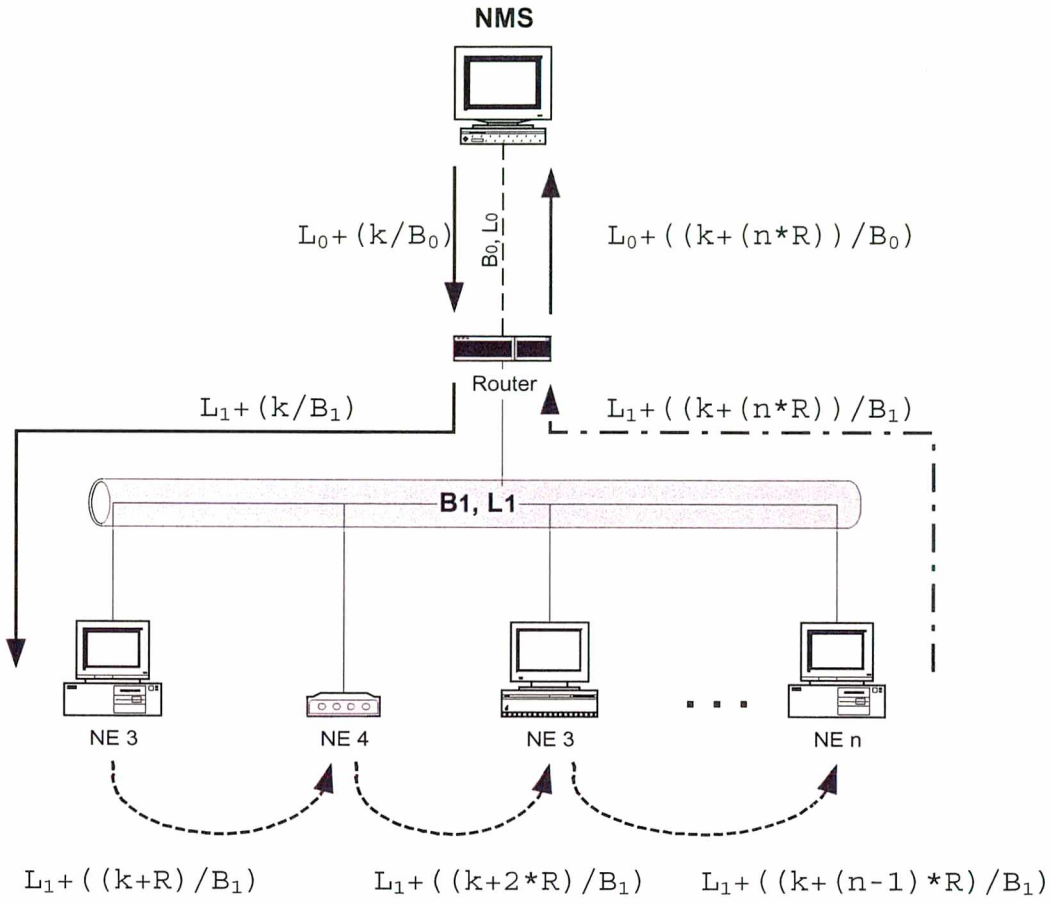


Figura 9-2 – Modelo de gerência do agente móvel.

Uma vez que os agentes móveis realizam a atividade de gerência através de uma série de tarefas, seu tempo de resposta é obtido com a soma das seguintes variáveis:

- Tempo de ida do agente móvel do último recurso do segmento (j -1) [4]para o primeiro recurso no segmento j:

$$T\_IDA_j := \left(\sum_{i=j-1}^{j} L_i \right) + \left(\sum_{i=j-1}^{j} \frac{k + R\left(\sum_{u=1}^{j-1} n_u \right)}{B_i} \right)$$

(2)

- Tempo do percurso do agente móvel por todos os recursos do segmento j:

$$T\_PERCORRE_j := \left(\sum_{i=2}^{n_j} \frac{k + R\left(\sum_{u=1}^{j-1} n_u \right) + (i-1)R}{B_j} \right) + (n_j - 1) L_j \tag{3}$$

- Tempo de retorno do agente móvel do último recurso gerenciado do segmento de maior índice para a NMS, localizada no segmento de índice zero:

$$T\_VOLTA\_FINAL := \left(\sum_{i=0}^{q} L_i \right) + \left(\sum_{i=0}^{q} \frac{k + R\left(\sum_{u=1}^{q} n_u \right)}{B_i} \right) \tag{4}$$

- O tempo total da operação de gerência realizada com o agente móvel é obtido com a soma das equações anteriores, ou seja:

$$T_{Ag\_movel} := \left(\sum_{j=1}^{q} (T\_IDA_j + T\_PERCORRE_j) \right) + T\_VOLTA\_FINAL \tag{5}$$

Onde q é o número de LANs interconectadas a serem gerenciadas.

---

[4] Quando j = 1, significa que o agente móvel está partindo da NMS localizada no segmento zero.

10 ESTUDOS DE CASO

A avaliação de desempenho, dentro do universo da gerência de redes, é um tema bastante abrangente, uma vez que diversos aspectos e protocolos estão envolvidos nesta atividade. No intuito de manter a homogeneidade com o trabalho de [RUBINSTEIN99], os experimentos apresentados a seguir assumem que:

- enlaces e nós não possuem carga;
- enlaces não têm perdas;
- o tempo de processamento na camada de aplicação não é considerado;

Neste trabalho, os valores das variáveis utilizadas para a latência, largura de banda, tamanho de pacotes e de agentes móveis, não são nossos. Ainda com o propósito de homogeneidade, foram incorporados os valores apresentados no trabalho de [RUBINSTEIN99].

10.1 CASO A: Gerência remota de uma LAN

Neste experimento, ilustrado na figura 10.1, os elementos de rede estão dispostos em uma rede local *Ethernet* com banda passante igual a 10Mbps e latência de 10μs. O gerenciamento é realizado por uma estação de gerenciamento que se encontra em um outro segmento, conectado à rede local. Este segmento apresenta latência maior e menor banda passante, quando comparado ao segmento da rede local e, por isso, pode ser chamado de enlace de gargalo. As características do enlace de gargalo variam de acordo com o efeito a ser estudado.

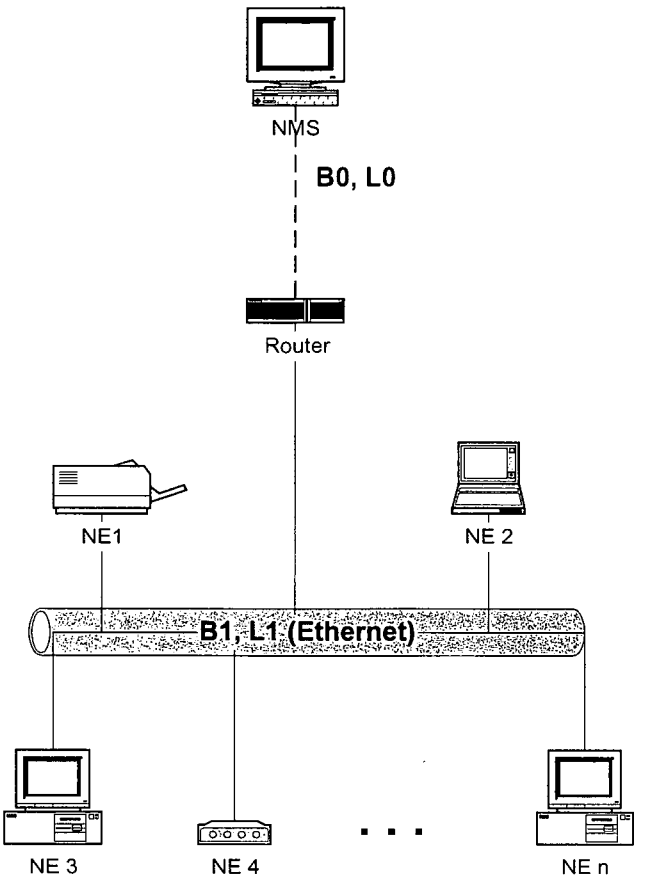


Figura 10-1 – Topologia do estudo de caso A.

- **Efeito da latência no enlace de gargalo**

A fim de analisar o efeito da latência no desempenho da atividade de gerência com SNMP e agente móvel, três diferentes valores foram testados, para a latência do enlace de gargalo, expostos na tabela 10.1. Neste caso, a banda passante do enlace de gargalo é, invariavelmente, de 2Mpbs, assim como o valor de 5k-octetos para o tamanho inicial do agente móvel. O número de elementos gerenciados varia de 0 até 250 nós. A tarefa realizada é a consulta à variável *ifInErrors* da MIB SNMP cujo pacote de pedido e resposta tem, respectivamente, 71 e 76 octetos.

Tabela 10-1 – Valores para a latência do enlace de gargalo.

Variáveis para o enlace	Latência (ms)
V_1	10
V_2	4
V_3	0,1

O gráfico da figura 10.2 ilustra o comportamento do SNMP e do agente móvel em cada um dos três valores associados ao enlace de gargalo. Nesta figura, observa-se que o comportamento do agente móvel é invariável em relação à variação da latência do enlace de gargalo. As três curvas que representam o uso de agentes móveis nos enlaces V_1, V_2 e V_3, praticamente passam pelos mesmos pontos no plano cartesiano. Em relação ao número de elementos gerenciados, é de se esperar que, quanto maior a quantidade de nós visitados, maior será o tamanho do agente móvel e, conseqüentemente, mais tempo ele levará para retornar à NMS.

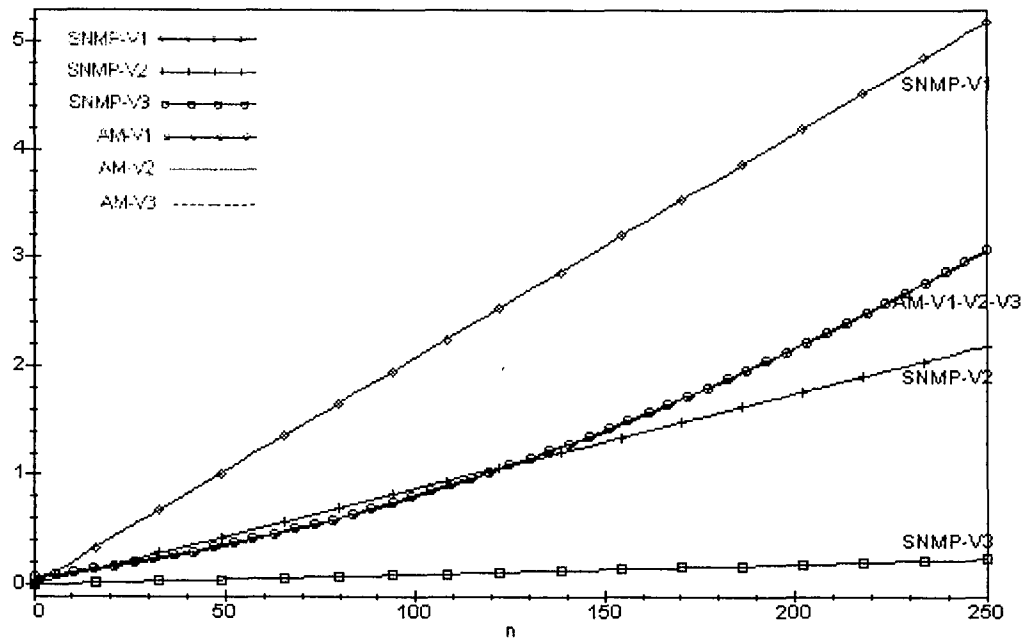


Figura 10-2 – Tempo de resposta para diferentes latências.

Já, no comportamento do SNMP, no mesmo gráfico, a diferença é visível para os diferentes enlaces de gargalo. Uma vez que, para cada recurso gerenciado, o enlace de gargalo é usado duas vezes. A discrepância entre os tempos de resposta do SNMP, para diferentes enlaces, aumenta proporcionalmente ao número de nós gerenciados.

Ainda, observando a figura 10.2, é possível concluir que, quando a NMS encontra-se em um enlace com baixa latência (V_1) e uma banda passante não muito baixa, é melhor utilizar o SNMP na gerência dos recursos de redes remotas.

- **Efeito da banda no enlace de gargalo**

A fim de avaliar os efeitos da banda passante do enlace de gargalo, os valores da tabela 10.2 foram utilizados neste experimento:

Tabela 10-2 – Valores para a banda passante do enlace de gargalo.

Variáveis para o enlace	Banda passante (Mpbs)
V_1	0.05
V_2	2
V_3	5

A latência do enlace de gargalo e o tamanho inicial do agente são fixados em 4ms e 5k-octetos respectivamente. A tarefa de gerência realizada é a mesma do experimento anterior com um pedido de 71 octetos e uma resposta de 76 octetos. O resultado deste experimento é apresentado na figura 10.3.

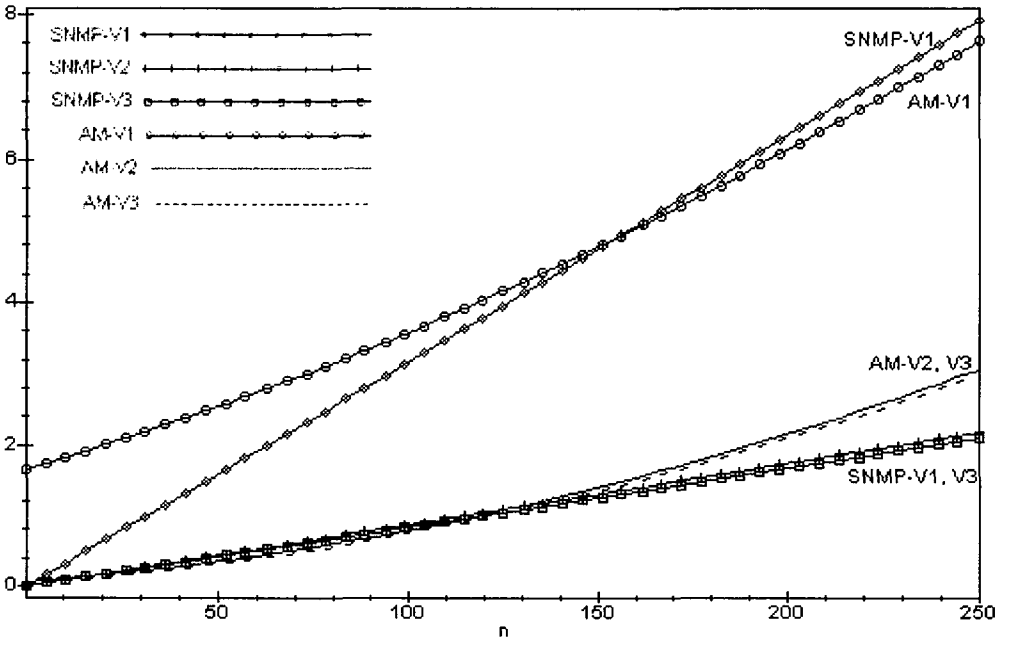


Figura 10-3 – Tempo de resposta para diferentes banda passante no enlace de gargalo.

Observando a figura 10.3, é possível concluir que ambos os tempos de resposta, do SNMP e agente móvel, sofrem alta elevação quando o valor da banda passante é muito pequeno (V_1). A variação da banda com os valores de V_1 e V_2 não influenciam muito no comportamento dos dois paradigmas, uma vez que estas atividades não exigem muita largura de banda para serem executadas.

- **Efeito tamanho inicial do agente móvel**

Outro efeito analisado neste trabalho é o tamanho inicial do agente móvel e sua influência no tempo de resposta. Neste experimento, os valores da banda passante e da latência foram fixados em 2Mpbs e 4ms respectivamente. A tarefa realizada consiste na consulta à variável *ifInErrors* com o tamanho do pedido de 71 octetos e uma resposta de 76 octetos. Os valores utilizados para o tamanho inicial do agente móvel foram de 1, 3, 5, 7 e 9 k-octetos. O resultado deste teste é apresentado na figura 10.4.

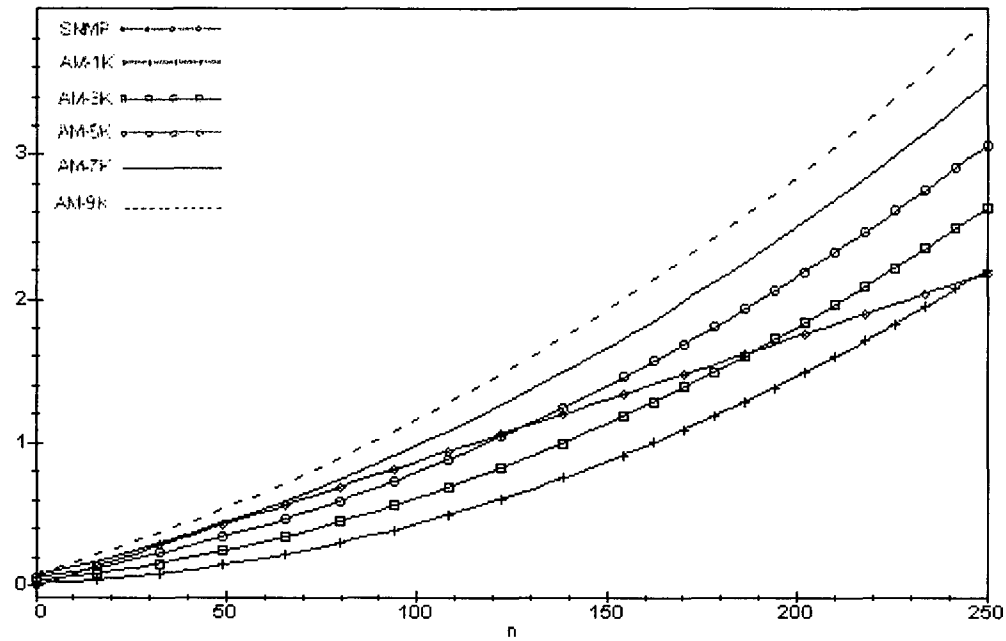


Figura 10-4 – Tempo de resposta para o SNMP e para diferentes tamanhos de agente móvel.

Neste experimento, observou-se que para um enlace de gargalo com valores médios de latência (4ms) e de banda passante (2Mbps), o SNMP apresenta um menor tempo de resposta quando o número de recursos gerenciados é maior que 250, independente do tamanho inicial do agente móvel. Para um valor de até aproximadamente 110 recursos gerenciados, o uso de agentes móveis de tamanho inicial de até 5k-octetos resulta em um tempo de resposta menor do que o SNMP.

- **Efeito da tarefa a ser realizada**

A análise do efeito do tamanho da tarefa de gerência no tempo de resposta também foi estudado. Três tarefas foram usadas neste experimento: a tarefa t1 consiste na consulta à variável *ifInErrors;* a tarefa t2 representa a consulta à variável *sysContact;* e a tarefa t3 busca a variável *sysDescr.* Todas estas variáveis pertencem à MIB SNMP. Os tamanhos dos pedidos e das respostas, incluindo os cabeçalhos da PDU *GetRequest* do UDP e do IP, para cada uma destas tarefas são apresentados na tabela 10.3.

Tabela 10-3 – Tamanho dos pacotes de pedido e resposta para as tarefas.

Tarefa	Pedido (octetos)	Resposta (octetos)
T1	71	76
T2	69	91
T3	66	173

O resultado deste experimento está ilustrado na figura 10.5. Os valores da latência e banda passante do enlace de gargalo utilizados neste experimento são de 4ms e 2Mpbs respectivamente.

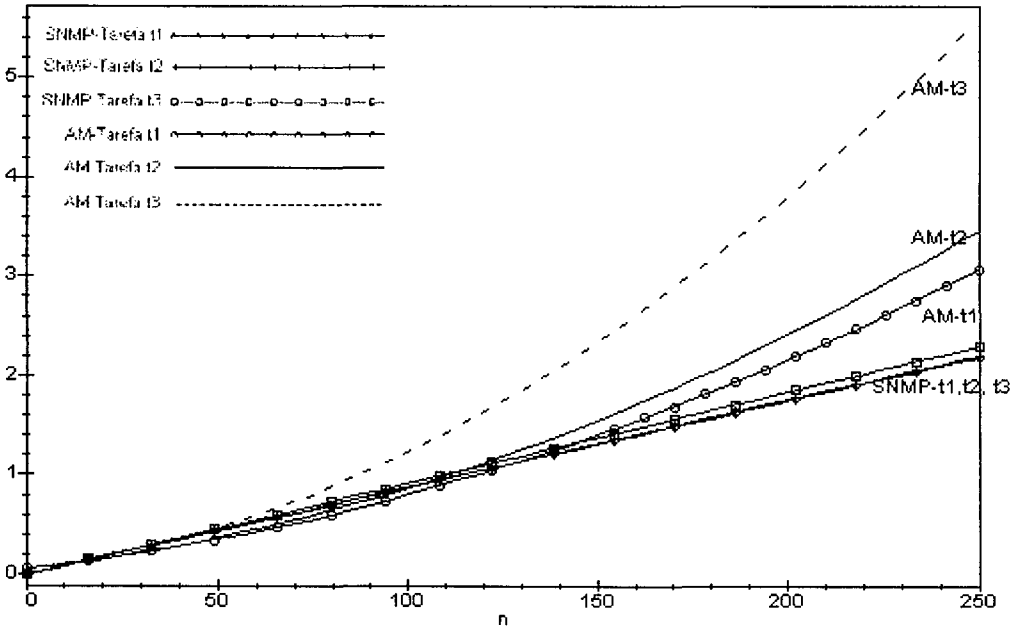


Figura 10-5 – Tempo de resposta para diferentes tarefas.

Como pode se observar na figura 10.5, o tempo de resposta do SNMP não sofre muita alteração com a variação das tarefas. Isto ocorre porque o SNMP realiza uma consulta de cada vez, ou seja, a variação da quantidade de octetos que trafega no enlace de gargalo é muito pequena para que implique em uma alteração significante no tempo de resposta. Já, no caso do agente móvel, quando o número de octetos do pedido e, principalmente, da resposta, aumentam, o tempo de resposta total sofre grande elevação. Este fato deve-se à característica dos agentes móveis de armazenar as

informações, (neste caso os pacotes contendo as respostas), de cada nó visitado pela rede. Desta forma, quanto maior o número de nós visitados, maior a influência no tempo de resposta, variando-se o tamanho das tarefas.

- **Conclusão do estudo do Caso A**

Para redes com topologias semelhantes ao do Caso A, o número de nós gerenciados e o tamanho da tarefa de gerência a ser executada são fatores importantes, a serem levados em consideração, na escolha do esquema de gerenciamento. Quando o número de nós gerenciados e o tamanho da tarefa realizada for alto o desempenho do SNMP, em relação ao tempo de resposta, é melhor. Deve-se observar também as características do enlace de gargalo. Para enlaces com latência muito alta e banda passante muito baixa, é sugerido o uso de agente móvel.

10.2 CASO B: Gerência local de uma LAN

A topologia deste estudo de caso, ilustrada na figura 10.6, apresenta a estação de gerenciamento (NMS) inserida na própria rede local, junto com os elementos a serem gerenciados. Esta topologia é considerada como um caso específico da topologia apresentada no Caso A (figura10.1), onde a banda da rede de índice zero tende a infinito ($B_0 \rightarrow \infty$), e a latência da mesma tende a zero ($L_0 \rightarrow 0$).

Quando possível, cada efeito é analisado em dois modelos de rede local: *Ethernet* e *Fast Ethernet*, com uma banda passante de 10Mpbs e 100Mpbs respectivamente.

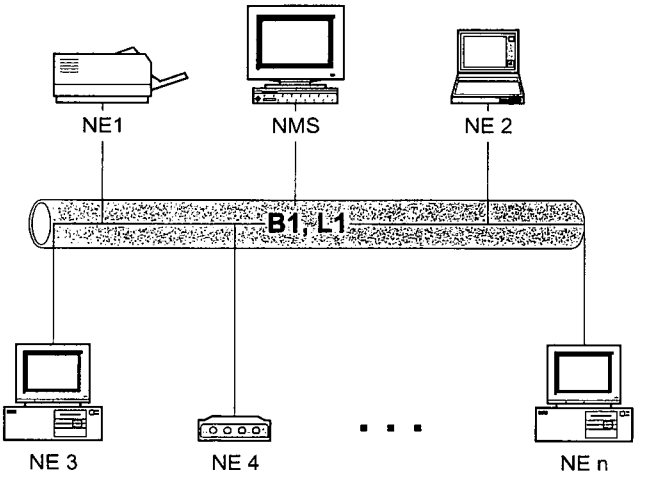


Figura 10.6 – Topologia de uma rede local.

- **Efeito da banda passante**

Neste experimento, o objetivo é analisar o efeito da variação da banda passante, de uma rede *Ethernet* (V1) para uma rede *Fast Ethernet* (V2), sobre o tempo de resposta. A atividade de gerência executada é a consulta à variável *ifInErrors*, cujo tamanho do pedido é de 71 e de resposta é de 76 octetos. A latência de ambas topologias foi fixada em 10μs. O tamanho inicial do agente móvel é de 5k-octetos. O resultado deste experimento é apresentado na figura 10.7.

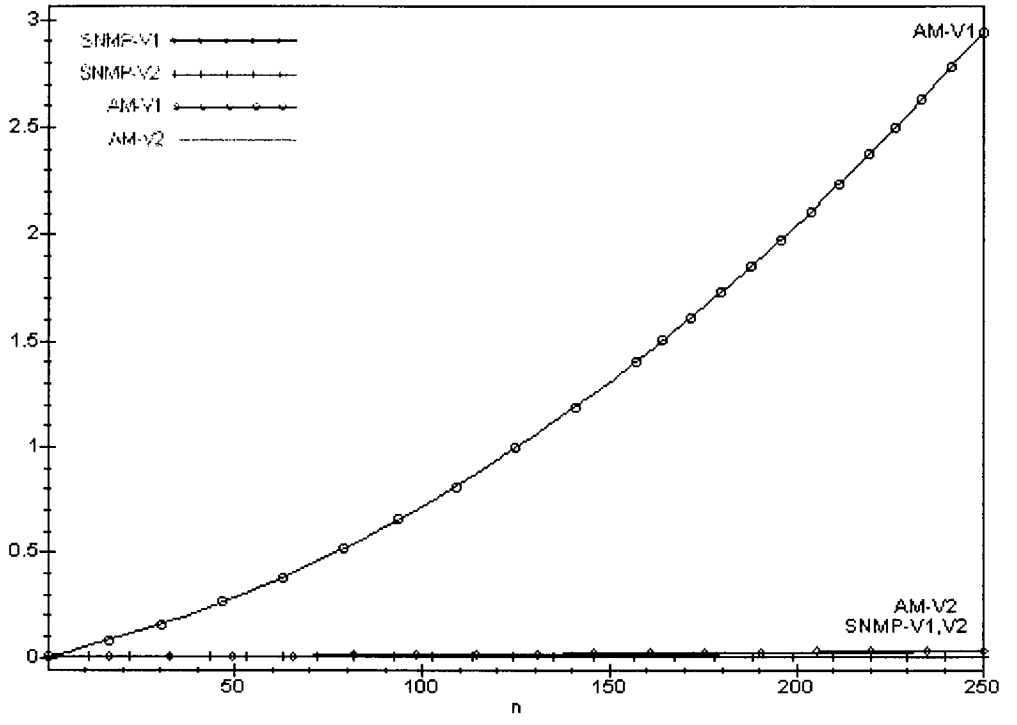


Figura 10.7 – Tempo de resposta para diferentes bandas passantes.

Como já afirmado anteriormente, o SNMP não exige muita largura de banda para realizar seus serviços. Logo, seu comportamento é, praticamente, igual nas duas topologias, *Ethernet* (V1) e *Fast Ethernet* (V2). Uma vez que não existe mais um enlace de gargalo com uma latência relativamente alta, o tempo de resposta do SNMP é muito baixo, tornando-o a opção mais eficiente para gerenciar redes com esta topologia. Uma vez que o tamanho do agente móvel cresce proporcionalmente ao número de nós visitados, seu comportamento é altamente influenciado ao percorrer uma *Ethernet* nas condições apresentadas. Quando trafegando em uma rede *Fast Ethernet*, com uma banda passante alta, sua performance praticamente se iguala à performance do SNMP.

- **Efeito do tamanho inicial do agente móvel**

O efeito do tamanho inicial do agente móvel, no Caso B, é analisado sob duas perspectivas: seu comportamento em uma rede *Ethernet* e em uma rede *Fast Ethernet*. A latência de ambas as redes é fixa em 10μs e o tempo de resposta foi calculado para agentes móveis com tamanho iniciais de 1, 3, 5, 7 e 9k-octetos. A tarefa executada é a mesma do experimento anterior: a consulta à variável *ifInErrors*.

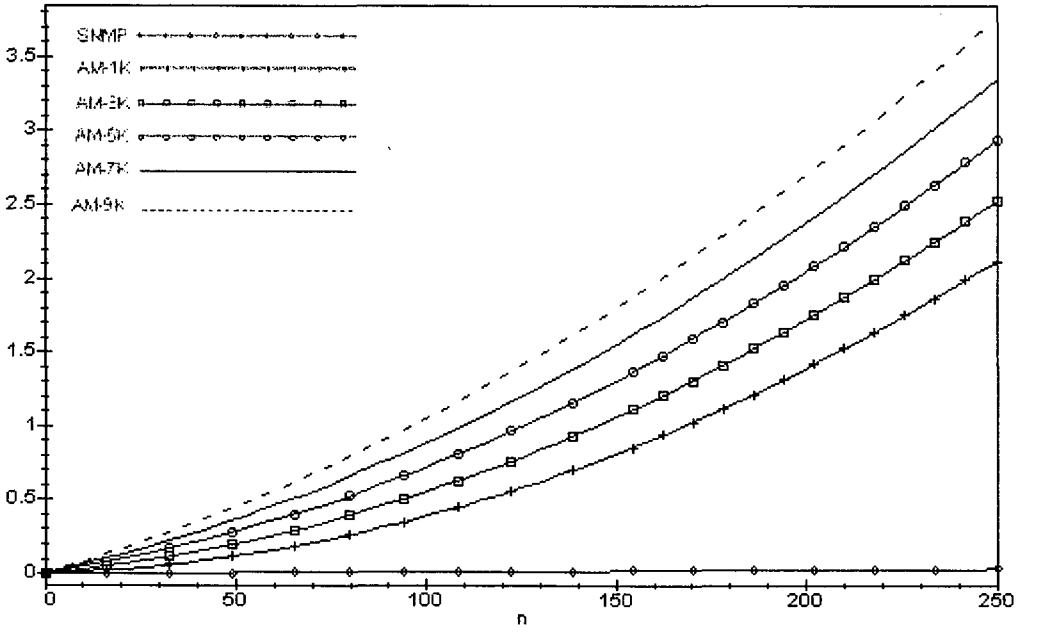


Figura 10.8 – Tempo de resposta para diferentes tamanhos iniciais do agente móvel em uma rede Ethernet.

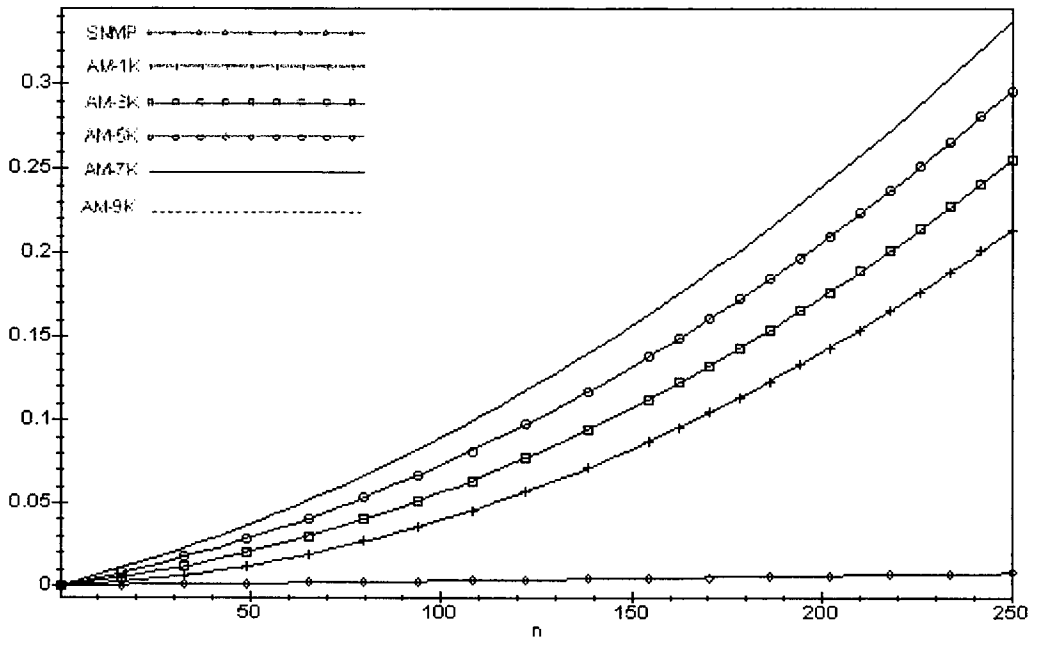


Figura 10.9 – Tempo de reposta para diferentes tamanhos iniciais do agente móvel em uma rede Fast Ethernet.

Observando a figura 10.8 e a figura 10.9, percebe-se uma semelhança na disposição dos dois gráficos. A única diferença entre as figuras é o eixo das coordenadas, onde o tempo de resposta é muito menor para redes *Fast Ethernet*. Isto é explicado pela alta banda passante (100Mpbs), tornando possível a passagem de informação mais rapidamente. O comportamento do SNMP não sofre alteração em nenhuma das duas topologias por motivos já explicados no estudo do efeito da banda passante.

- **Efeito da tarefa a ser realizada**

O efeito da tarefa a ser realizada também é estudado para redes *Ethernet* e *Fast Ethernet*, de forma a verificar o comportamento do SNMP e do agente móvel nestas topologias. As tarefas testadas foram as mesmas das usadas no Caso A, na tabela 10.3.

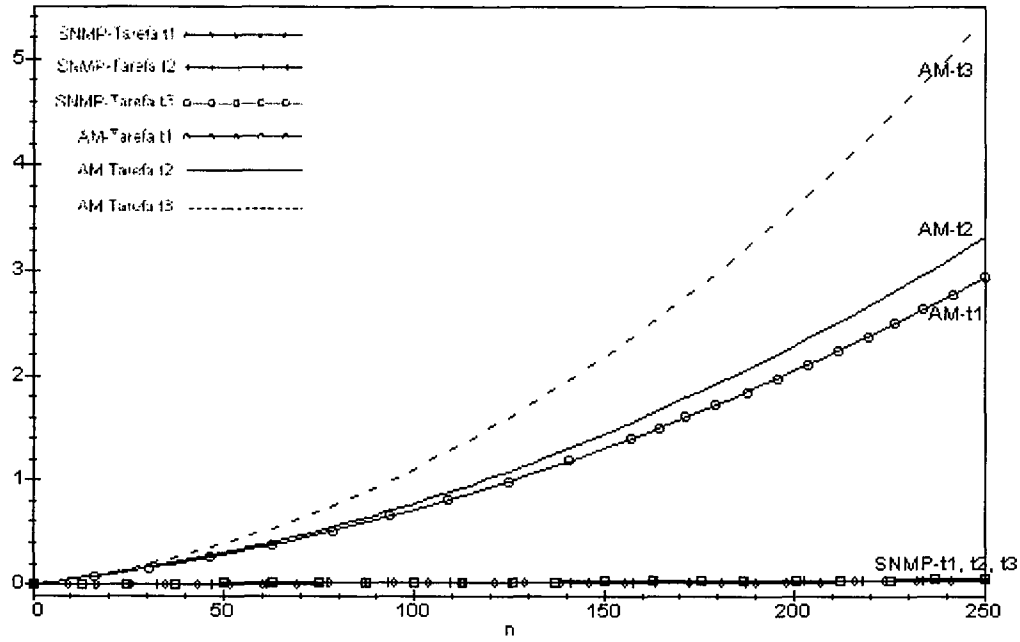


Figura 10.10 – Tempo de resposta para diferentes tarefas em uma rede Ethernet.

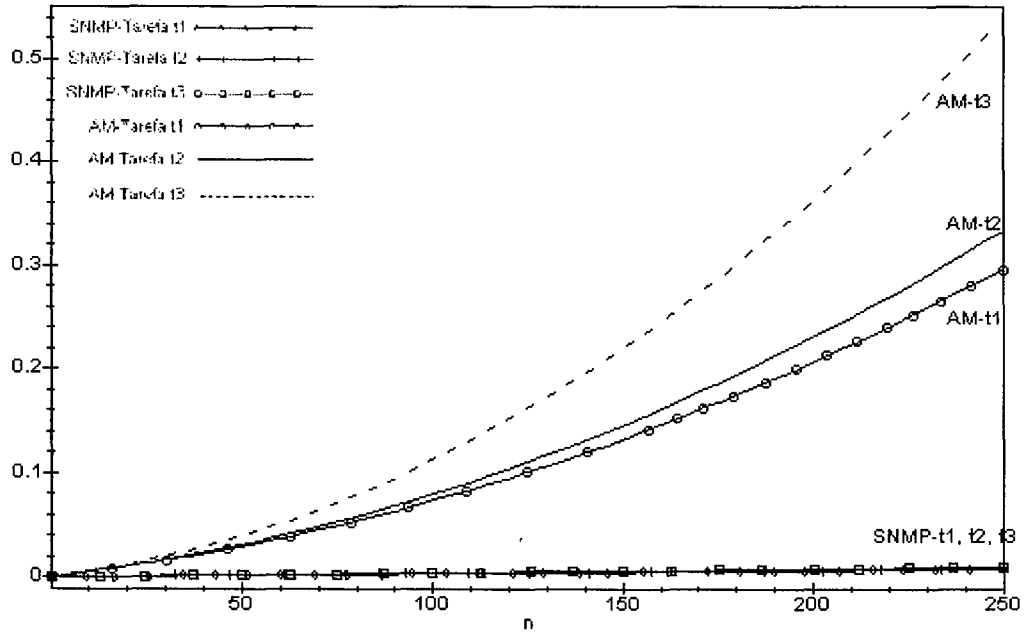


Figura 10.11 – Tempo de resposta para diferentes tarefas em uma rede Fast Ethernet.

As figuras 10.10 e 10.11 ilustram os resultados deste experimento. A disposição dos dois gráficos é semelhante, diferindo apenas nos valores do tempo de resposta. A rede *Fast Ethernet*, por possuir maior banda passante, possibilita que as atividades de gerência sejam realizadas em menor tempo.

Da mesma forma que na figura 10.5, é possível notar, nas figuras 10.10 e 10.11, que o tempo de resposta do SNMP não sofre nenhuma alteração significante em relação ao tamanho das tarefas realizadas, pois a consulta em cada recurso é realizada separadamente das outras. Portanto, o aumento de alguns octetos de uma tarefa para outra, não influi muito no tempo de resposta final do SNMP. Por outro lado, o desempenho do agente móvel é diretamente proporcional ao tamanho da tarefa executada. Ou seja, quanto maior o tamanho da tarefa e quanto maior o número de elementos gerenciados ou percorridos, maior será o tempo de resposta.

- **Conclusão do estudo do Caso B**

Diante dos experimentos realizados, é possível concluir que, para uma topologia onde a NMS está inserida na própria rede a ser gerenciada, o uso do SNMP na atividade de gerência é mais eficiente, em termos de tempo de resposta, do que o uso de agente móvel. Isto ocorre, pois as redes locais têm uma latência muito baixa e uma largura de banda bastante alta, favorecendo o desempenho do SNMP. Em todos os efeitos estudados neste caso, o tempo de resposta do SNMP foi consideravelmente menor do que o do agente móvel.

10.3 CASO C – Gerência de inter-redes

A topologia apresentada na figura 10.12 é mais complexa que nos casos anteriores. Aqui, trata-se de dois segmentos gerenciados por uma NMS localizada em um terceiro segmento, conhecido como enlace de gargalo, uma vez que apresenta latência alta e banda passante baixa.

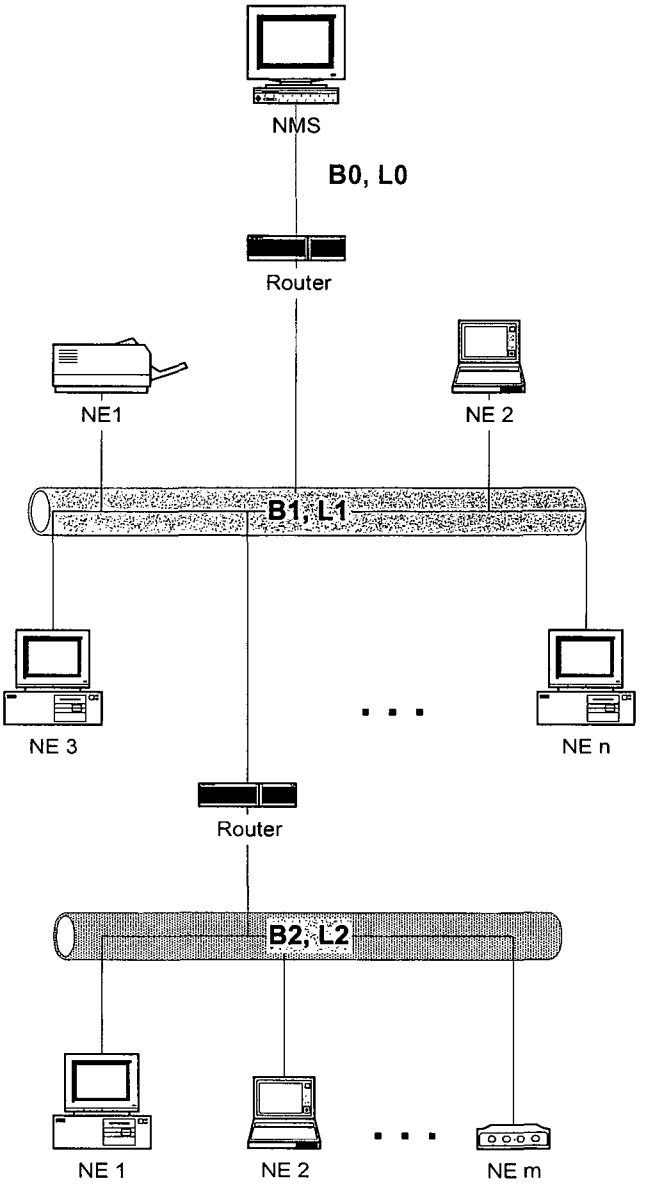


Figura 10.12 – Topologia do Caso C.

Os efeitos analisados foram testados em duas topologias: *Ethernet* e *Fast Ethernet* nos segmentos 1 e 2 respectivamente, e no modelo contrário, *Fast Ethernet* no segmento 1 e *Ethernet* no segmento 2.

Os resultados são apresentados de duas formas: variando-se simultaneamente o número de recursos dos dois segmentos e, fixando o número de recursos de uma rede, e variando a da outra para facilitar a análise dos gráficos.

- **Efeito da latência**

O experimento com a latência variante foi realizado variando-se a latência do enlace de gargalo, com os mesmos valores utilizados nos outros experimentos (tabela 10.1), de forma a favorecer uma comparação entre as topologias testadas. Os valores utilizados são de 10, 4 e 0.1 ms, para V1, V2 e V3 respectivamente.

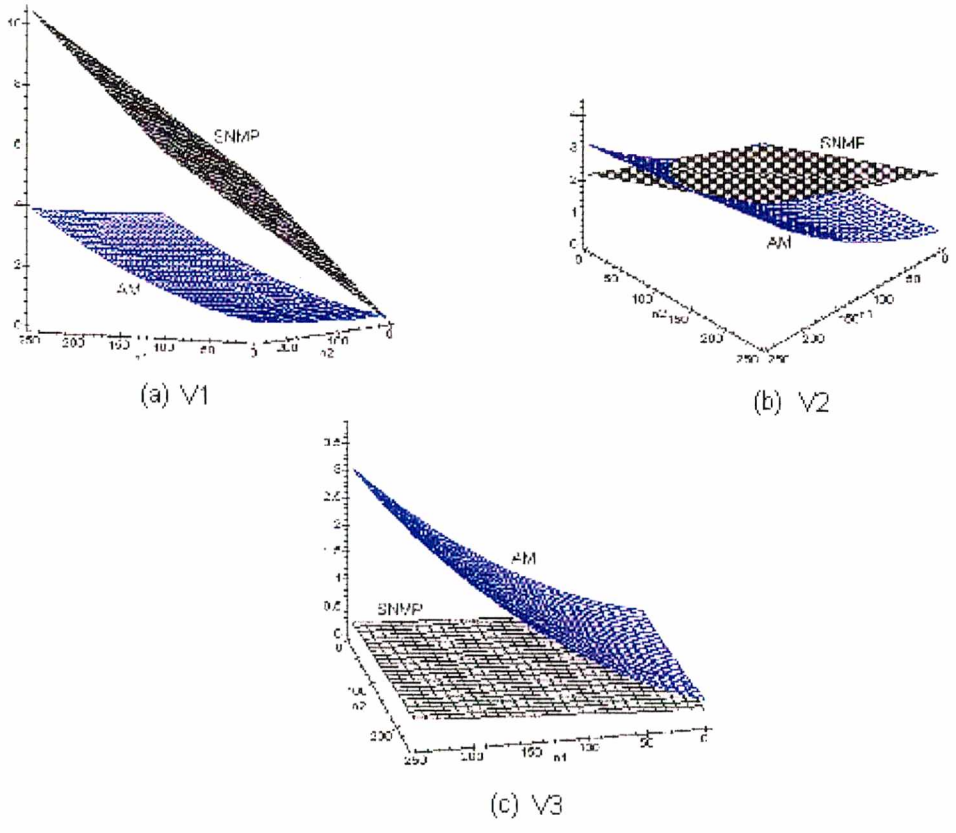


Figura 10.13 – Tempo de resposta para diferentes latência do enlace de gargalo. Topologia: Gargalo x Ethernet X Fast Ethernet.

Analisando a figura 10.13, da topologia Gargalo x *Ethernet* x *Fast Ethernet*, nota-se que, para uma latência muito alta (figura 10.13 (a)), o uso do agente móvel é mais indicado, independentemente do número de estações dos dois segmentos. Isto ocorre uma vez que o SNMP trafega informações no enlace de gargalo para cada consulta realizada. Para uma latência de valor médio (figura 10.13 (b)) e uma quantidade de estações do segmento *Ethernet* muito alto, o uso do SNMP é aconselhável. Para um

número de estações de até aproximadamente 200 nós, o agente móvel é mais eficiente. Na figura 10.13 (c), a latência do enlace de gargalo é baixa (0,1ms) implicando em uma ótima performance do SNMP, independente do número de estações gerenciadas. Com uma latência relativamente baixa no enlace de gargalo, o SNMP definitivamente é mais eficiente do que o agente móvel.

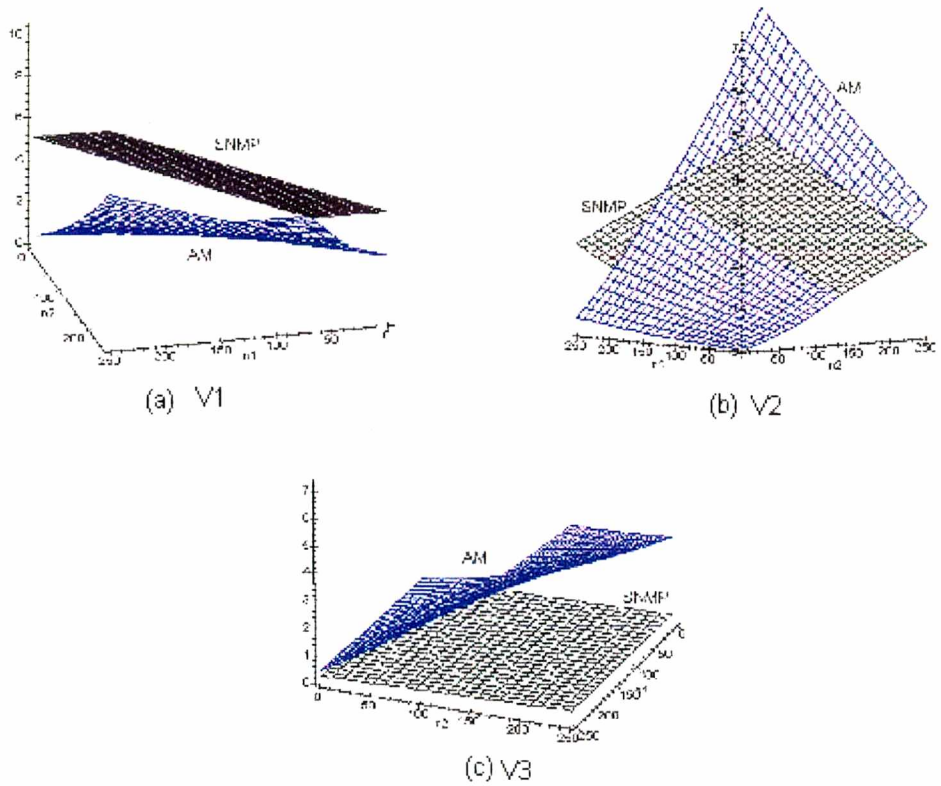
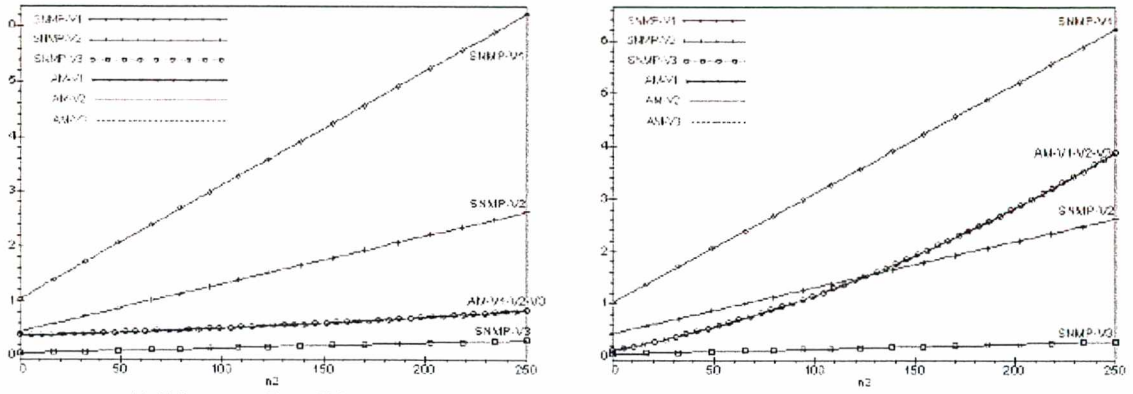


Figura 10.14 – Tempo de resposta para diferentes latência do enlace de gargalo. Topologia: Gargalo x Fast Ethernet X Ethernet.

A figura 10.14 ilustra as diferentes latências do enlace de gargalo, desta vez com a topologia Gargalo x Fast Ethernet x Ethernet. No caso de uma latência muito baixa, (figura 10.14 (a)), é visível que o uso do agente móvel é mais rápido do que o SNMP para qualquer número de estações gerenciadas. Já para uma latência de valor médio, (figura 10.14 (b)), analisando a tendência das duas funções, conclui-se que o gráfico do SNMP cresce, linearmente, com o aumento do número de estações gerenciadas,

enquanto que o comportamento do agente móvel é uma curva que sofre alta elevação à medida que o número de estações gerenciadas cresce, principalmente com as estações localizadas no barramento Ethernet, mais lento.

De forma a facilitar a compreensão deste efeito, outros testes foram realizados, fixando-se, desta vez, o número de elementos de um segmento e variando o do outro. Os resultados destes testes são apresentados nas figuras 10.15, 10.16 e 10.17, onde o número de recursos do primeiro segmento foi fixado em 50, 150 e 250 estações, respectivamente.



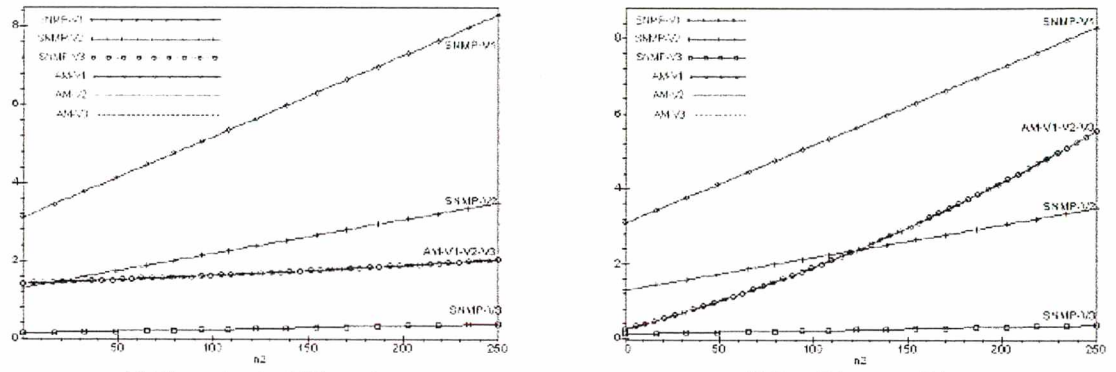
(a) Ethernet x Fast Ethernet (b) Fast Ethernet x Ethernet

Figura 10.15 –Tempo de resposta para diferentes latências do enlace de gargalo. Número de estações do primeiro segmento fixado em 50.

Analisando a figura 10.15, percebe-se que o comportamento do SNMP, independente da topologia, é inversamente proporcional à latência do enlace de gargalo. Para um enlace de gargalo com latência muito alta (V1), a performance do SNMP é muito inferior à performance do agente móvel. Por outro lado, para uma latência muito baixa (V3), o uso do SNMP resulta em um menor tempo de resposta, quando comparado ao uso do agente móvel. Nota-se, ainda, na figura 10.15 que, quando a topologia estudada é Gargalo x *Fast Ethernet* x *Ethernet*, figura 10.15 (b), o tempo de resposta do SNMP com uma latência média (v2) é menor do que o do agente móvel. Isso é explicado com a propriedade do agente móvel de armazenar os dados de cada elemento visitado, ou seja, ao chegar no segmento *Ethernet*, (figura 10.15 (b)), cuja banda é menor do que a

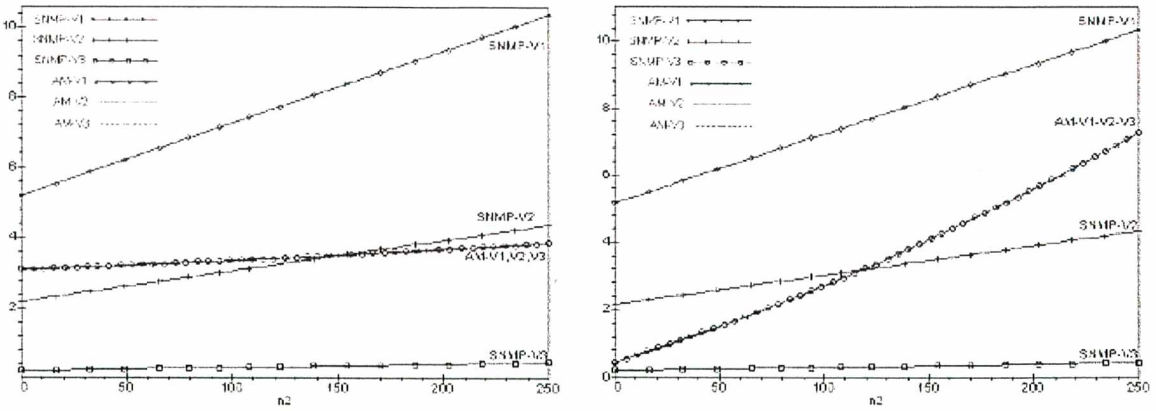
Fast Ethernet, o tamanho do agente móvel já estará bastante elevado, levando mais tempo para realizar a atividade de gerência.

As figuras 10.16 e 10.17 apresentam comportamentos semelhantes ao da figura 10.15 pelos mesmos motivos descritos acima. A diferença encontra-se no tempo de resposta, uma vez que o aumento do número de estações do primeiro segmento acarreta em um maior tempo de resposta.



(a) Ethernet x Fast Ethernet (b) Fast Ethernet x Ethernet

Figura 10.16 –Tempo de resposta para diferentes latências do enlace de gargalo. Número de estações do primeiro segmento fixado em 150.



(a) Ethernet x Fast Ethernet (b) Fast Ethernet x Ethernet

Figura 10.17 –Tempo de resposta para diferentes latências do enlace de gargalo. Número de estações do primeiro segmento fixado em 250.

- **Efeito da banda passante**

O experimento com a banda passante variante, seguindo os padrões dos outros estudos de caso, foi realizado variando-se a banda passante do enlace de gargalo, com os valores de 0.05, 2 e 5 Mpbs para V1, V2 e V3 respectivamente, com uma latência de 4ms. O tamanho da tarefa realizada é de 71 octetos para o pacote de pedido e 76 octetos para o pacote de resposta.

Nas figuras 10.18, o comportamento do SNMP e do agente móvel são ilustrados para os diferentes valores da banda passante do enlace de gargalo. Conclui-se que, para uma latência média fixa em 4ms, na topologia Gargalo x *Ethernet x Fast Ethernet*, o uso do agente móvel resulta em um menor tempo de resposta, independente da banda passante utilizada e do número de estações gerenciadas. Nota-se ainda que para um número de estações maior que 250, o uso do SNMP pode vir a ser mais viável, uma vez que sua função cresce linearmente, enquanto que o comportamento do agente móvel é uma curva determinada por uma função do 2^{o} grau.

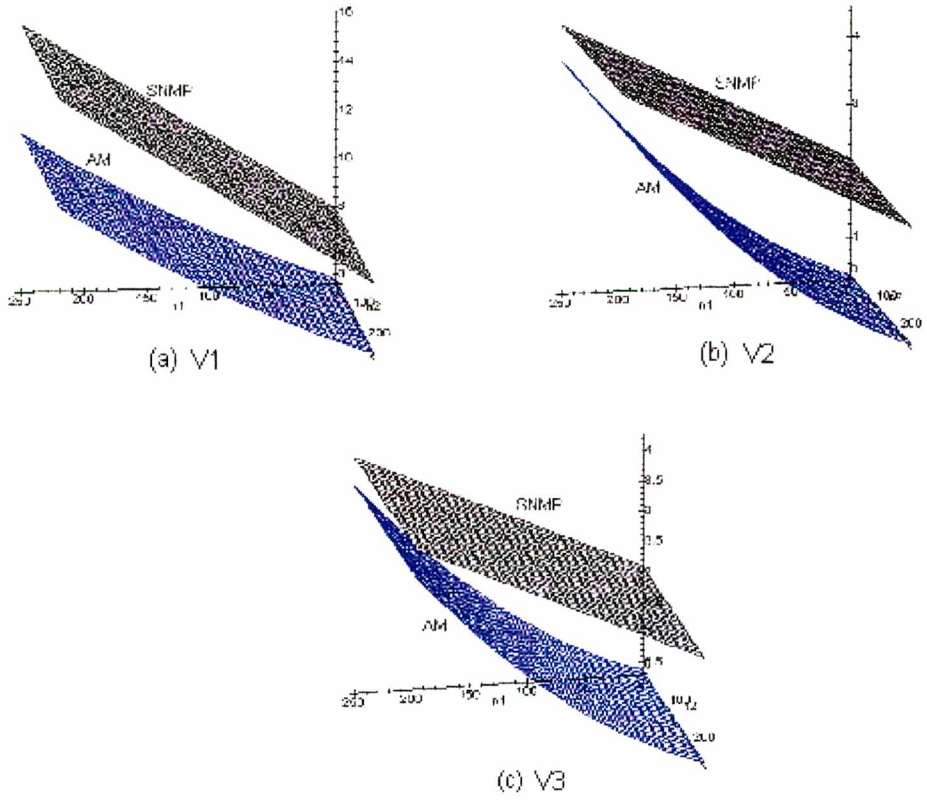


Figura 10.18 – Tempo de resposta para diferentes bandas passantes. Topologia: Gargalo x Ethernet X Fast Ethernet.

Na figura 10.19, o mesmo efeito é estudado, desta vez na topologia Gargalo x *Fast Ethernet* x *Ethernet*. Percebe-se que, quando a banda passante do enlace de gargalo é muito baixa (figura 10.19 (a)), o tempo de resposta do SNMP é um pouco mais elevado do que o tempo de resposta do agente móvel. Já para uma banda passante média ou alta, (figura 10.19 (b) e (c)), a performance do agente móvel é melhor quando o número de elementos gerenciados do barramento *Ethernet* é menor do que aproximadamente 150 estações. Para uma quantidade maior de estações gerenciadas, o uso do SNMP é aconselhável.

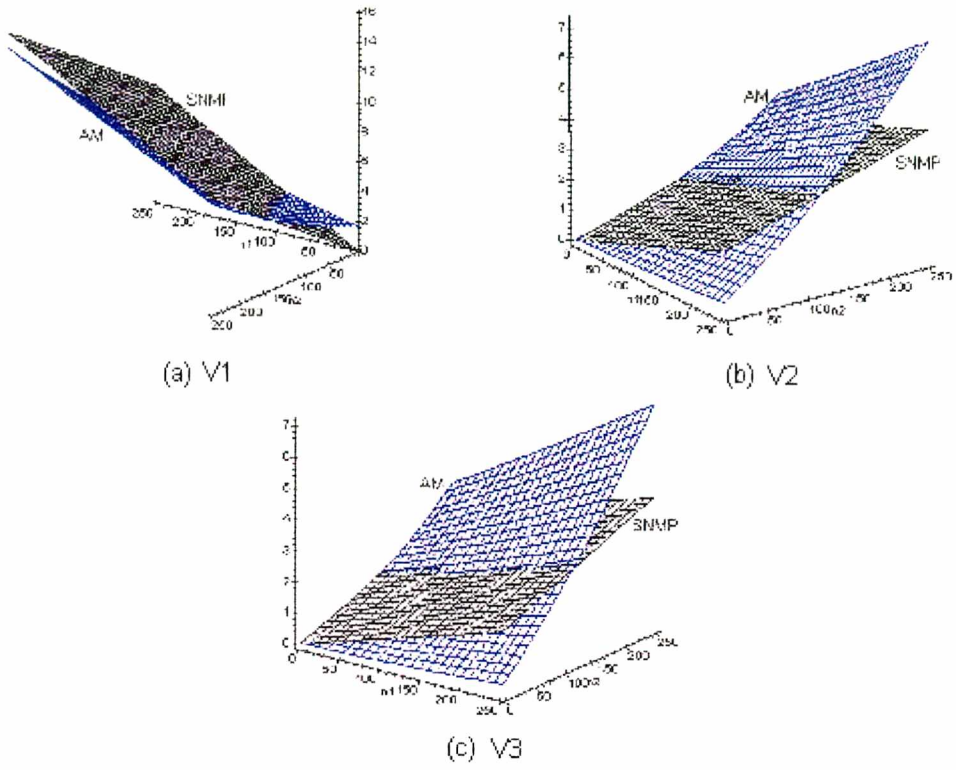
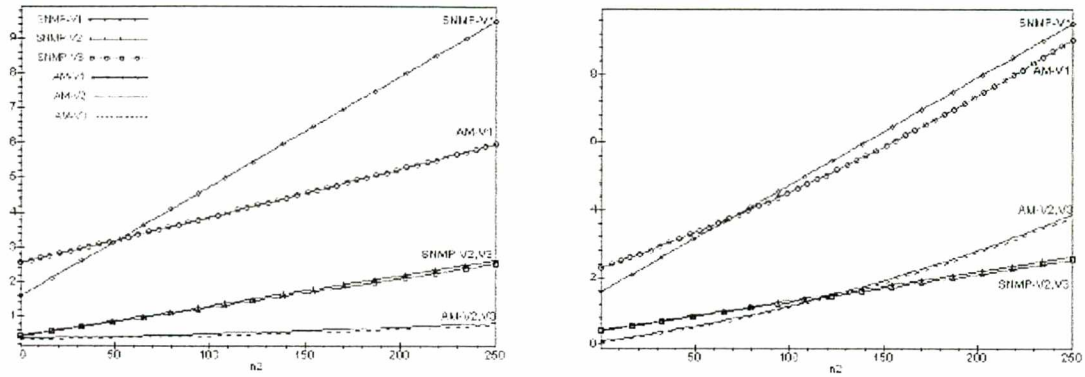


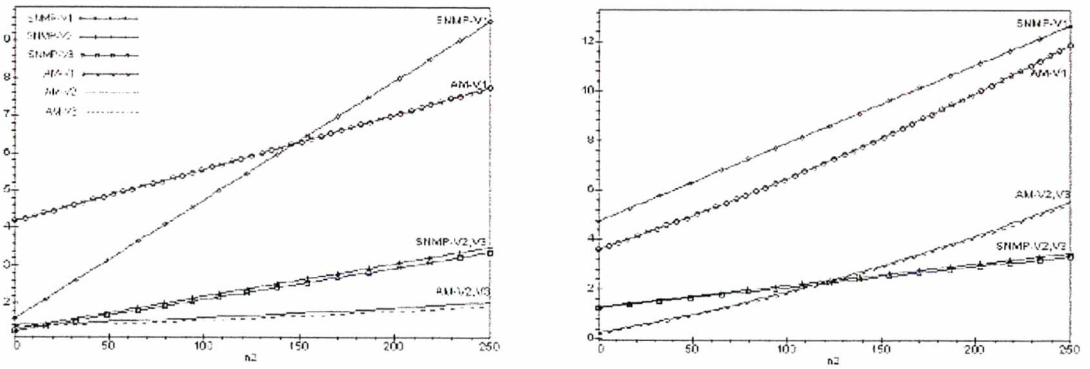
Figura 10.19 – Tempo de resposta para diferentes bandas passantes. Topologia: Gargalo x Fast Ethernet X Ethernet.

De forma a facilitar a compreensão deste efeito, outros testes foram realizados, fixando-se, desta vez, o número de elementos de um segmento e variando o do outro. Os resultados destes testes são apresentados nas figuras 10.20, 10.21 e 10.22, onde o número de recursos do primeiro segmento foi fixado em 50, 150 e 250 estações, respectivamente.



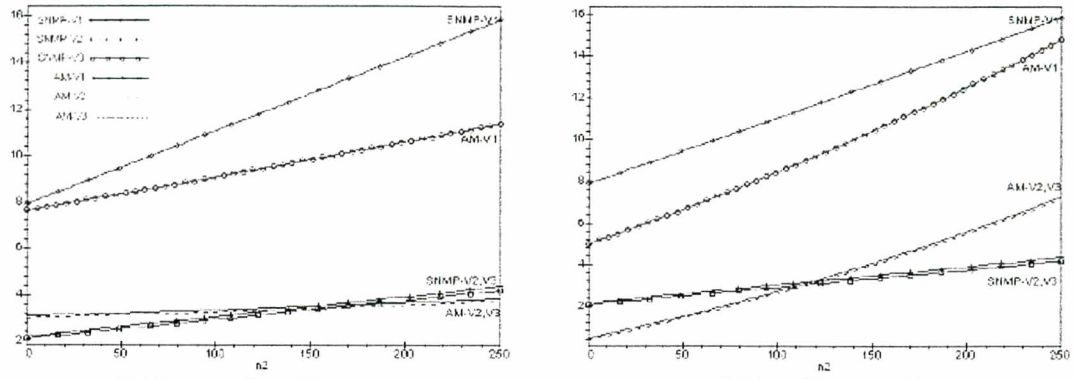
(a) Ethernet x Fast Ethernet (b) Fast Ethernet x Ethernet

Figura 10.20 –Tempo de resposta para diferentes banda passante do enlace de gargalo. Número de estações do primeiro segmento fixado em 50.



(a) Ethernet x Fast Ethernet (b) Fast Ethernet x Ethernet

Figura 10.21 –Tempo de resposta para diferentes banda passante do enlace de gargalo. Número de estações do primeiro segmento fixado em 150.



(a) Ethernet x Fast Ethernet (b) Fast Ethernet x Ethernet

Figura 10.22 –Tempo de resposta para diferentes banda passante do enlace de gargalo. Número de estações do primeiro segmento fixado em 250.

Observando as figuras 10.20, 10.21 e 10.22, conclui-se que, para uma banda passante muito baixa (V1=0.05Mpbs), a performance do SNMP é muito inferior ao do agente móvel na topologia Gargalo x *Ethernet* x *Fast Ethernet*. Na topologia inversa, a diferença entre o tempo de resposta dos dois paradigmas é mínimo. Para uma banda passante de valor médio ou alto (V2=2Mpbs, V3=5Mpbs) no enlace de gargalo, o comportamento do SNMP e do agente móvel não varia muito. Ainda observando estas três figuras, percebe-se que para a topologia Gargalo x *Ethernet* x *Fast Ethernet*, o agente móvel obteve melhor performance. Enquanto que na topologia inversa: Gargalo x *Fast Ethernet* x *Ethernet*, o uso do SNMP, à medida que o número de estações gerenciadas aumenta, é mais indicado.

- **Efeito tamanho inicial do agente móvel**

O efeito do tamanho inicial do agente móvel, assim como no Caso B, é analisado em duas topologias: Gargalo x *Ethernet* x *Fast Ethernet* e Gargalo x *Fast Ethernet* x *Ethernet*. A latência do enlace de gargalo é de 4ms e a banda passante do mesmo é fixado em 2Mpbs. Nos outros dois segmentos, a latência é fixa em 10μs e o tempo de resposta foi calculado para agentes móveis com tamanho iniciais de 1, 3, 5, 7 e 9k-

octetos. A tarefa executada é a mesma do experimento anterior: a consulta à variável ifInErrors.

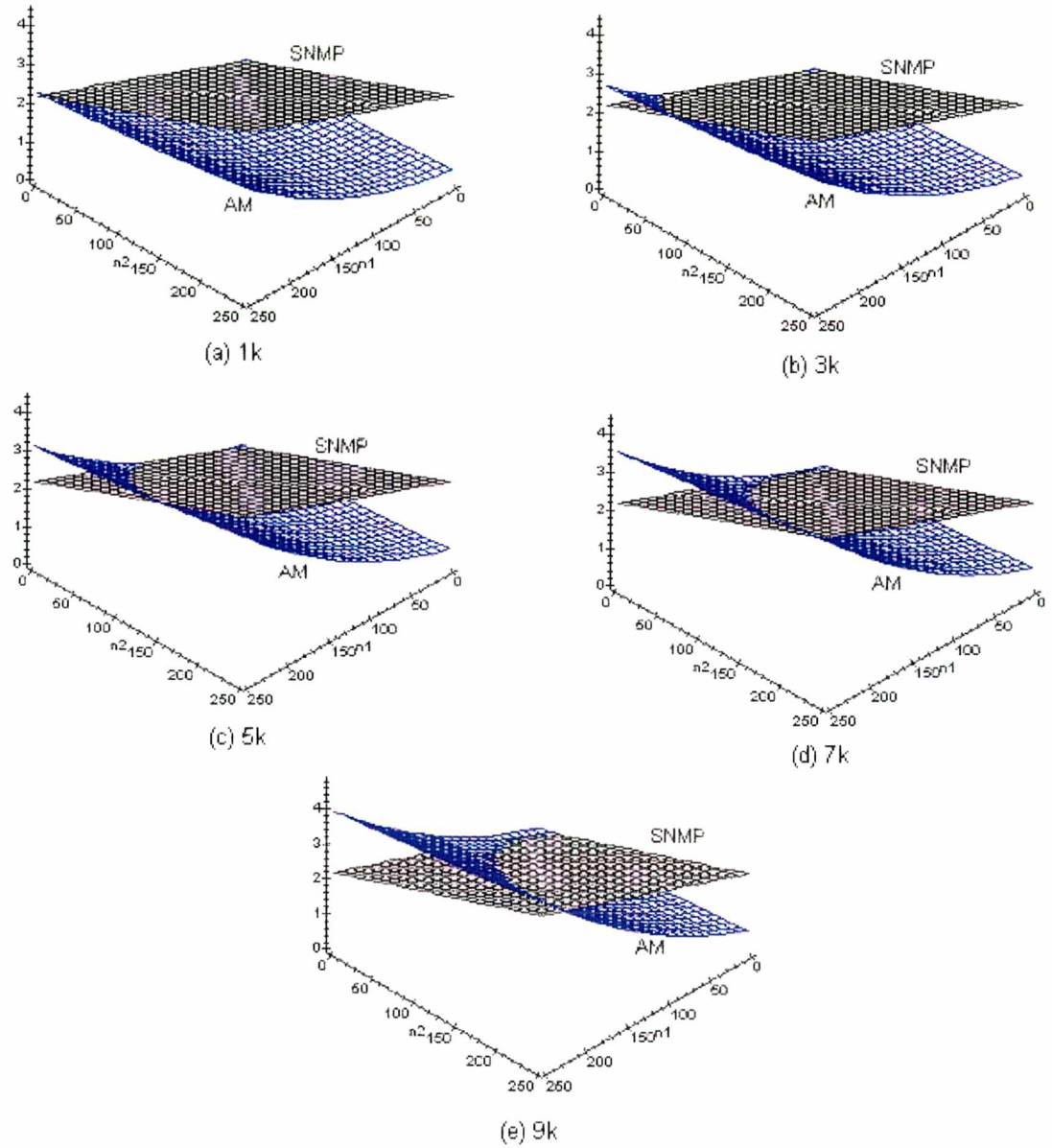


Figura 10.23 – Tempo de resposta para diferentes tamanhos iniciais do agente móvel na topologia Gargalo x Ethernet x Fast Ethernet.

De acordo com a figura 10.23, para agentes móveis com tamanho inicial de até 3k-octetos, a performance do agente móvel é melhor, praticamente, para qualquer número

de estações gerenciadas nos dois segmentos. Com o aumento do tamanho inicial do agente, para 5k ou mais octetos, nota-se que o desempenho do agente móvel diminui à medida que o número de elementos gerenciados no barramento *Ethernet* cresce. Isso deve-se ao fato de que o gerenciamento dos recursos localizados no barramento Fast Ethernet não influi muito no tempo de resposta, uma vez que a latência e banda passante deste segmento são altas. Quanto maior o tamanho inicial do agente móvel, pior a sua performance em relação ao número de estações do primeiro segmento.

Já na topologia inversa, Gargalo x Fast Ethernet x Ethernet, (figura 10.24), a performance do agente móvel é diretamente afetada com o aumento do número de estações do segundo barramento, Ethernet. Quanto maior o número de estações de ambos os segmentos, mais indicado o uso do SNMP nesta atividade de gerência. Para um número grande de recursos no barramento Fast Ethernet junto com um número pequeno de recursos no segundo barramento, Ethernet, melhor o uso de agente móvel.

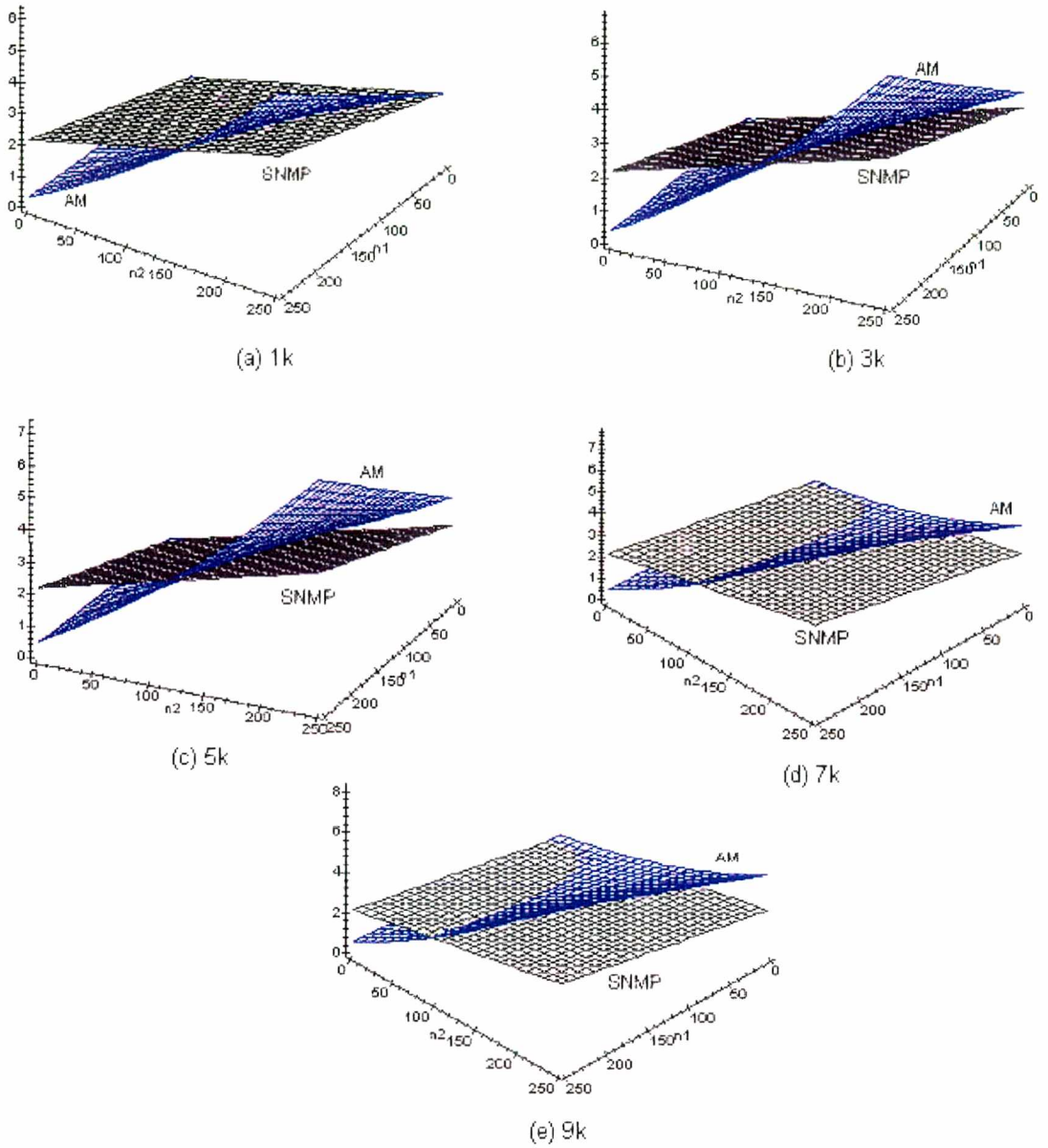


Figura 10.24 - Tempo de resposta para diferentes tamanhos iniciais do agente móvel na topologia Gargalo x Fast Ethernet x Ethernet.

- **Efeito da tarefa a ser realizada**

Por fim, o efeito das diferentes tarefas realizadas é estudado. Os valores para este experimento são os mesmos utilizados para este efeito nos outros estudos de caso.

A figura 10.25 ilustra o comportamento do SNMP e do agente móvel ao realizar diferentes tarefas na topologia Gargalo x *Ethernet* x *Fast Ethernet*.

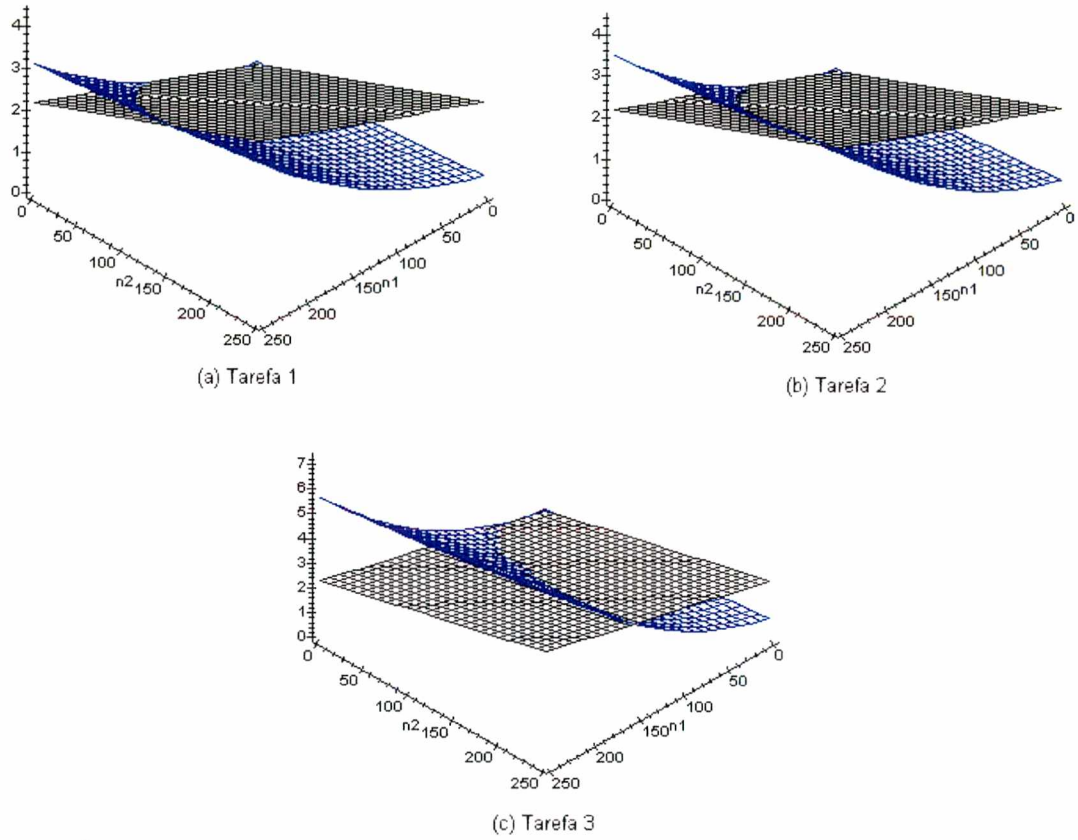


Figura 10.25 - Tempo de resposta para diferentes tarefas na topologia Gargalo x Ethernet x Fast Ethernet.

Analisando a figura 10.25, observa-se que a performance do agente móvel, é diretamente afetada pelo o aumento do número de estações do barramento Ethernet (segmento 1). Caso este número seja alto, o tempo de resposta do SNMP é menor do que o do agente móvel. O tempo de resposta varia muito pouco com os diferentes tamanhos de tarefa, a não ser na tarefa t3, onde o tempo de resposta pode chegar a aproximadamente 6 segundos.

Na topologia inversa, Gargalo x Fast Ethernet x Ethernet, a performance do agente móvel diminui com o aumento do número de estações no segundo segmento, barramento Ethernet (figura 10.26). O uso de agente móvel nesta topologia só é viável quando o número de recursos do segundo segmento for muito baixo. Caso contrário, é aconselhável a utilização do SNMP. É interessante notar que com uma tarefa maior (t3), o tempo de resposta do agente móvel chega a ser mais do que o dobro do SNMP para um número de estações muito alto nos dois barramentos.

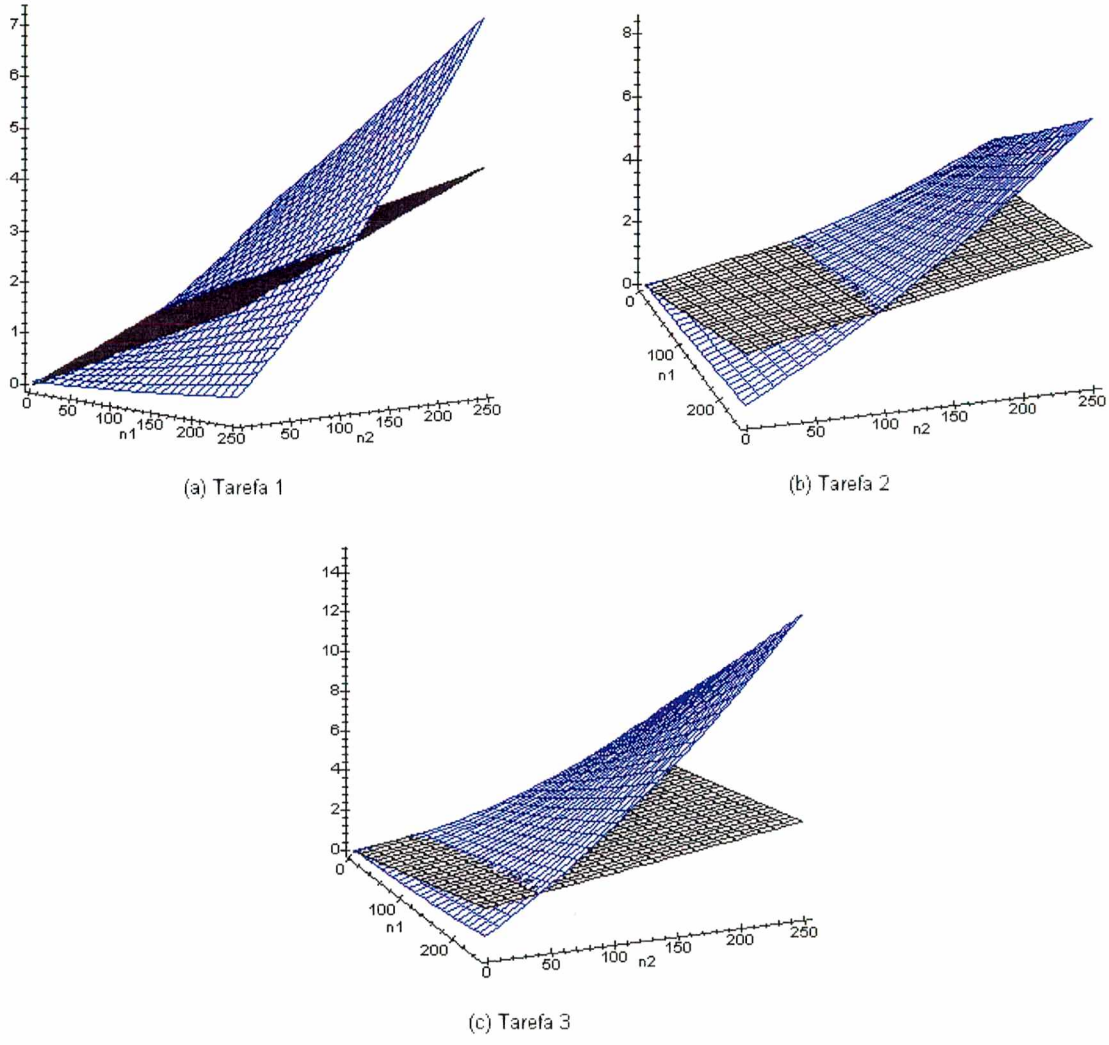


Figura 10.26 – Tempo de resposta para diferentes tarefas. Topologia Gargalo x Fast Ethernet x Ethernet.

- **Conclusão do estudo de caso C**

Diante da topologia apresentada neste estudo de caso, é importante levar em conta o número de estações conectadas ao barramento de menor banda passante, pois ele tem maior influência, na performance do agente móvel, nos efeitos estudados. Os valores da latência e da banda passante do enlace de gargalo também devem ser cuidadosamente analisados, uma vez que eles afetam, diretamente, o comportamento de ambos os paradigmas.

11 CONCLUSÃO

Este trabalho apresentou o paradigma de código móvel e o seu uso na atividade de gerência de redes. Devido à falta de padrões e a escassez de sistemas deste tipo, pouco se sabe sobre o verdadeiro desempenho de agentes móveis na gerência de redes. Os supostos benefícios e vantagens de agentes móveis apresentados na literatura, raramente foram testados e validados. Diante desta ausência de resultados práticos, este trabalho é de grande significância, uma vez que ele se propõe a realizar um estudo prático do desempenho de agentes móveis em topologias típicas de redes de comunicação de dados.

Este trabalho propôs um modelo matemático, através do qual três estudos de caso foram analisados, no intuito de comparar a performance do SNMP com o desempenho do agente móvel frente a uma série de efeitos como: variação da latência, banda passante, tamanho inicial do agente móvel e tarefas de gerenciamento. A métrica utilizada neste trabalho foi o tempo de resposta. Os resultados desta análise mostram que a topologia da rede gerenciada tem que ser cuidadosamente levada em consideração na hora da escolha do seu paradigma de gerenciamento.

Na primeira topologia estudada, a gerência é feita remotamente através de um enlace com alta latência e uma banda passante baixa. Neste caso, foi possível concluir que:

- O desempenho do SNMP é diretamente afetado pela variação da latência do enlace de gargalo, enquanto que o comportamento do agente móvel permanece o mesmo, independente da latência apresentada. Assim, para enlaces com latência muito alta, é mais aconselhável o uso de agente móvel, pois seu tempo de resposta é consideravelmente menor do que o do SNMP.
- A banda passante do enlace de gargalo só influiu no comportamento do SNMP e do agente móvel, quando o seu valor foi muito baixo (0.05Mpbs). Para enlaces com uma banda passante média ou alta, o desempenho dos dois paradigmas foi o mesmo, sendo o SNMP um pouco mais rápido que o agente móvel.

- A variação do tamanho inicial do agente móvel e do tamanho das tarefas de gerenciamento, praticamente não afetaram o desempenho do SNMP. Já o comportamento do agente móvel, sofreu significantes alterações com o aumento da quantidade dos bytes nas operações. Quanto maior o tamanho inicial do agente móvel e quanto maior o tamanho da resposta da consulta à MIB, maior o tempo de resposta apresentado pelo agente móvel. Este resultado é explicado através da propriedade do agente de percorrer toda a rede, armazenando as informações de cada operação sobre cada recurso visitado.

No segundo estudo de caso, o ambiente analisado consistiu na gerência local de uma rede, ou seja, a NMS foi conectada diretamente no barramento da rede a ser gerenciada. Com os experimentos realizados concluiu-se que:

- A variação da banda passante influenciou somente o desempenho do agente móvel. Uma vez que o SNMP não exige uma alta largura de banda, seu comportamento em uma rede *Ethernet* (Banda passante = 10Mbps) ou em uma rede *Fast Ethernet* (Banda passante = 100Mbps) resultou aproximadamente no mesmo tempo de resposta. O desempenho do agente móvel, por outro lado, foi altamente afetado com a variação na banda passante. O motivo deste resultado é encontrado na característica do agente móvel de armazenar as informações dos recursos visitados à medida em que ele percorre a rede. Ou seja, sua dificuldade em trafegar pela rede cresce proporcionalmente ao número de recursos visitados.
- Os efeitos da variação do tamanho inicial do agente móvel e do tamanho do pedido (*request*) e resposta (*reply*) das tarefas de gerência apresentaram resultados semelhantes ao estudo de caso anterior. Quanto maior o tamanho inicial do agente móvel ou quanto maior a resposta da atividade de gerência, pior o desempenho do agente móvel, independentemente da rede local ser Ethernet ou Fast Ethernet.

O terceiro estudo de caso apresentado consistiu na gerência remota de uma rede com mais de dois segmentos. Além das características do enlace de gargalo variarem com o efeito a ser estudado, a topologia da rede também foi alterada. Cada efeito foi estudado

em duas topologias de rede: Gargalo x *Ethernet* x *Fast Ethernet* e Gargalo x *Fast Ethernet* x *Ethernet*. Este último estudo de caso apresentou os seguintes resultados:

- Observou-se que a latência do enlace de gargalo afeta diretamente o comportamento do SNMP. Já o desempenho do agente móvel é praticamente o mesmo para qualquer um dos valores testados para a latência no enlace de gargalo. Novamente, conclui-se que, para enlaces com latência muito alta, o uso do agente móvel é mais aconselhável do que o do SNMP. Ainda neste experimento, foi possível concluir que para uma latência média (V2) no enlace de gargalo, o SNMP é mais aconselhável na topologia Gargalo x *Fast Ethernet x Ethernet*, enquanto que na topologia inversa, Gargalo x *Ethernet x Fast Ethernet*, o uso do agente móvel leva a um melhor tempo de resposta.
- A variação da banda passante, no enlace de gargalo nesta topologia, provocou resultados interessantes. Para enlaces de gargalo com uma banda passante muito pequena, o uso do agente móvel geralmente é mais aconselhável, independentemente da topologia da rede gerenciada. Quando o valor da banda passante do enlace de gargalo é médio ou alto, é necessário analisar dois parâmetros: o número de elementos no barramento com a menor banda passante, no caso deste modelo, o barramento *Ethernet*, e a topologia da rede gerenciada: *Ethernet x Fast Ethernet* ou o inverso. Em uma rede *Ethernet x Fast Ethernet*, o desempenho do agente móvel é melhor que o do SNMP. Na topologia inversa, o uso do agente móvel só é aconselhado quando o número de elementos na rede com menor banda passante (*Ethernet*) for pequeno ($n_2 \cong < 130$).
- Os resultados obtidos com o efeito do tamanho inicial do agente móvel foram semelhantes aos dos casos anteriores: quanto maior o tamanho inicial do agente móvel, maior o seu tempo de resposta e pior o seu desempenho. Mas, neste estudo de caso, observou-se que, na topologia *Ethernet* x *Fast Ethernet*, o desempenho do agente móvel é melhor, independentemente do seu tamanho inicial, quando o número de elementos gerenciados no barramento Ethernet não for muito alto ($n_1 <$ x). À medida em que o tamanho inicial do agente móvel aumenta, o valor de x decresce. De forma contrária, na *topologia Fast Ethernet x Ethernet*, o desempenho do agente móvel é, geralmente, pior do que o do SNMP, a não ser quando o número

de recursos gerenciados no barramento com baixa banda passante (*Ethernet*) for muito baixo.

- O resultado obtido com a variação do tamanho das tarefas de gerência, mostrou que o comportamento do SNMP não variou com as diferentes tarefas. Já o desempenho do agente móvel foi altamente afetado pela tarefa T3, cuja resposta possuía um tamanho significativamente maior que as outras duas tarefas. Neste caso, o tempo de resposta do agente móvel, comparado ao do SNMP, foi bem mais alto.

Analisando os efeitos dos três estudos de caso, foi possível concluir que:

- O desempenho do SNMP, na gerência de redes com enlace de alta latência, é muito baixo.
- A banda passante do enlace de gargalo não resulta em qualquer alteração significante para o SNMP, enquanto que, no agente móvel, esta alteração pode ser muito significante.
- O desempenho do agente móvel é inversamente proporcional ao seu tamanho inicial e ao tamanho da tarefa realizada, principalmente, em relação ao tamanho da resposta associada à tarefa, uma vez que este parâmetro será adicionado ao tamanho do agente móvel tantas vezes quanto o número de elementos visitados.

Os objetivos deste trabalho foram alcançados com sucesso. A proposta de uma nova taxonomia para a área de código móvel, a apresentação de um modelo matemático para a avaliação de desempenho do SNMP e dos agentes móveis na gerência de redes, e a análise dos resultados dos experimentos realizados, são significantes contribuições para a área de sistemas de código móvel.

11.1 Dificuldades encontradas

A principal dificuldade encontrada para o desenvolvimento deste trabalho foi a falta de padrões. Esta dificuldade, porém, serviu como principal motivação para a realização deste trabalho, uma vez que um dos seus objetivos é de contribuir para a compreensão da mobilidade de código.

11.2 Perspectivas futuras

Este trabalho apresentou diversos aspectos do paradigma de código móvel e, portanto, oferece uma série de perspectivas onde pesquisas podem e precisam ser realizadas para que o desenvolvimento de sistemas deste tipo seja impulsionado . Por ser um paradigma relativamente novo, muito se tem a fazer e a descobrir sobre ele. Os domínios de aplicação apresentados carecem de implementações para que esta tecnologia continue amadurecendo e impulsionando novas aplicações.

Pode-se adotar o uso de um simulador de redes, o NIST ou *Network Simulator*, para realizar os mesmos experimentos analisados neste trabalho através de uma dedução matemática. Outro ponto importante a ser realizado é o uso de um ambiente de desenvolvimento de agentes, para implementar um sistema de gerenciamento capaz de gerenciar as topologias apresentadas nos estudos de caso. Com o uso deste ambiente, pode-se validar os resultados obtidos neste trabalho podem ser submetidos a um processo de validação.

Este trabalho pode, ainda, ser refinado, considerando outros parâmetros (carga e perda de pacotes nos segmentos, tempo de processamento na CPU, entre outros), a fim de obter resultados mais precisos. Para isto, a dedução matemática proposta pode ser reutilizada, adicionando-se à variável latência (L_x), por exemplo, o valor do tempo médio de processamento na CPU do recurso gerenciado.

12 REFERÊNCIAS BIBLIOGRÁFICAS

[BALDI97] BALDI, M.; GAI, S.; PICCO, G. P. Exploiting code mobility in decentralized and flexible network management. In: Mobile agents: 1st international workshop MA '97. LNCS v. 1219. Germany : Springer-Verlag, apr. 1997. p. 13-26.

[2] BAUMANN, J.; HOHL, F.; RADOUNIKLIS, N. et al. Communication concepts for mobile agent systems. In: Mobile agents: 1st international workshop MA '97. LNCS v. 1219. Germany : Springer-Verlag, apr. 1997. p. 123-135.

[3] BHATTACHARJEE, S.; CALVERT, K.; ZEGURA, E. On active netwroking and congestion. Georgia, USA, 1996. Tech Report. GIT-CC-9602.

[4] BIC, L.; FUKUDA, M.; DILLENCOURT, M. Distributed computing using autonomous objects. IEEE Computer, Aug. 1996.

[5] CAI, T.; GLOOR, P.; NOG, S. DataFlow: a workflow management system on the Web using transportable agents. Hanover, NH, 1996. Technical Report TR96-283 : Department of Computer Science, Dartmouth College.

[6] CASE, J.; FEDOR, M.; SCHOFFSTALL, M. L. et al. Simple network management protocol. RFC 1157, May 1990.

[7] CASE, J.; McCLOGHRIE, K.; ROSE, M. et al. Structure of management information for version 2 of the simple network management protocol. RFC 1902, Jan. 1996.

[CHESS95] CHESS, D.; GROSOF, B.; HARRISON, C. et al. Itinerant agents for mobile computing. IEEE Personal Communications, v. 2, n. 5, p. 34-49, Oct. 1995.

[CUGOLA97] CUGOLA, G.; GHEZZI, C.; PICCO, G. P. et al. Analyzing mobile code languages. In: Vitek, J.; Tschudin, C. Mobile Object Systems: Towards the Programable Internet. LNCS v. 1222. Germany : Springer-Verlag, 1997. p. 93-111.

[FALCONE87] FALCONE, J. A programmable interface language for heterogeneous distributed systems. ACM Transactions on Computer Systems, v. 5, n. 4, p. 330-351, july 1987.

[FARMER96a] FARMER, W.; GUTTMAN, J.; SWARUP, V. Security for mobile agents: issues and requirements. In: NATIONAL INFORMATION SYSTEMS SECURITY CONFERENCE. Proceedings... p. 591-597, 1996.

[FARMER96b] FARMER, W.; GUTTMAN, J.; SWARUP, V. Security for mobile agents: authentication and state appraisal. In: FOURTH EUROPEAN SYMPOSIUM ON RESEARCH IN COMPUTER SECURITY. Procedings... Roma : Springer-Verlag, sep. 1996. p. 118-130.

[FORMAN94] FORMAN, G.; ZAHORJAN, J. The challenges of mobile computing. IEEE Computer, v. 27, n. 4, pg. 38-47, apr. 1994.

[GENESERETH94] GENESERETH, M.; KETCHPEL, S. Software agents. Communications of the ACM, v. 37, n. 7, jul. 1994.

[GHEZZI97] GHEZZI, C.; VIGNA, G. Mobile code paradigms and technologies: a case study. In: Mobile agents: 1st international workshop MA '97. LNCS, v. 1219. Germany : Springer-Verlag, apr. 1997. p. 39-49.

[GOLDSZMIDT95] GOLDSZMIDT, G.; YEMINI, Y. Distributed management by delegation. In: 15TH INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING. Proceedings... June 1995.

[GRAY95] GRAY, R. Agent Tcl: a transportable agent system. In: CIKM WORKSHOP ON INTELLIGENT INFORMATION AGENTS (1995 : Baltimore, Md). Proceedings... Dec, 1995.

[GRAY96] GRAY, R. Agent Tcl: a flexible and secure mobile-agent system. In: 1996 Tcl/Tk WORKSHOP. Proceedings... p. 9-23, july 1996.

[GRAY97] GRAY, R.; KOTZ, D., NOG, S. et al. Mobile agents for mobile computing. In: SECOND AIZU INTERNATIONAL SYMPOSIUM ON PARALLEL ALGORITHMS/ARCHITECTURES SYNTHESIS. Proceedings... Fukushima, mar, 1997.

[GRAY] GRAY, R.; KOTZ, D. CYBENKO, G, RUS, D. D'Agents: security in a multiple-language, mobile-agent system.

[HARRISON97] HARRISON, C.; CHESS, D.; KERSHENBAUM, A. Mobile agents: are they a good idea? In: VITEK, J.; TSCHUDIN, C. Mobile Object Systems: Towards the Programable Internet. LNCS v. 1222. Germany : Springer-Verlag, 1997. p. 25-47.

[HOOG91] HOOG, J. Island: aliasing protection in object-oriented languages. In: OOP-SLA '91. Proceedings... 1991.

[HOHL98] HOHL, F. Time limited blackbox security. In: Mobile agents and security, LNCS v. 1419. Springer-Verlag, 1998. p. 92-111.

[IMIELINSKY94] IMIELINSKY, T.; BADRINATH, B. Wireless computing: challenges in data management. Communications of the ACM, v. 37, n. 10, 1994. p. 18-28.

[JOHANSEN95a] JOHANSEN, D.; VAN RENESSE, R.; SCHNEIDER, F. B. Operating system support for mobile agents. In: FIFTH WORKSHOP ON HOT TOPICS IN OPERATING SYSTEMS. Proceedings... p. 42-45, may 1995.

[JOHANSEN95b] JOHANSEN, D.; VAN RENESSE, R.; SCHNEIDER, F. B. An introduction to the TACOMA distributed system – version 1.0. Norway, june. 1995. Tehcnical report 95-23. Department of Computer Science : University of Tromso and Cornell University.

[JONES94] JONES, K. Internet's SNMP and ISO's CMIP protocols for network management. International Journal of Network Management. p. 130-137. Sep. 1994.

[KINIRY97] KINIRY, J.; ZIMMERMAN, D. A hands-on look at Java mobile agents. IEEE Internet Computing, v. 1, n. 4, p. 21-30, 1997.

[KNABE95] KNABE, F. Language support for mobile agents. Pensylvania, Dec. 1995. PhD thesis : Carnegie Mellon School of Computer Science, Carnegie Mellon University.

[LANGE97] LANGE, D. Java Aglets application programming interface (J-AAPI). White paper, IBM Corporation, feb. 1997.

[LANGE96] LANGE, D.; CHANG, D. IBM Aglets Workbench – programming mobile agents in Java. White paper, IBM Corporation, sep. 1996.

[MAL] Mobile agent list. http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole/mal/preview/preview.html

[MILOJICIC99] MILOJICIC, D.; DOUGLIS, F.; WHEELER, R. Mobility: processes, computers and agents. 1 ed. ACM Press, feb. 1999.

[NECULA97] NECULA, G. Proof-carrying code. In: 24TH ACM SYMPOSIUM ON PRINCIPLES OF PROGRAMMING LANGUAGES. Proceedings... France, 1997.

[OMG95] OBJECT MANAGEMENT GROUP. CORBA: architecture and specification. Aug. 1995. (http://www.omg.org)

[PEINE97] PEINE, H.; STOLPMANN, T. The architecture of the Ara platform for mobile agents. In: Mobile agents: 1st international workshop MA '97. LNCS v. 1219. Germany : Springer-Verlag, apr. 1997. p. 50-61.

[ROTHERMEL97] ROTHERMEL, K.; POPESCU-ZELETIN, R. (eds.) Mobile Agents: 1st international workshop MA '97. LNCS v. 1219. Germany : Springer-Verlag, apr. 1997.

[RUBINSTEIN99] RUBINSTEIN, M. G. Análise do desempenho de agentes móveis no gerenciamento de redes. Programa de Engenharia Elétrica da COPPE/UFRJ. (Exame de qualificação de doutorado). Rio de Janeiro, Agosto. 1999.

[SCHNEIER96] SCHNEIER, B. Applied cryptography – protocols, algorithms and source code in C. 2 ed. John Wiley & Sons, Inc., 1996.

[STAMOS90a] STAMOS, J.; GIFFORD, D. Implementing remote evaluation. IEEE Transactions on Software Engineering, v. 16, n. 7, p. 710-722, jul. 1990.

[STAMOS90b] STAMOS, J.; GIFFORD, D. Remote Evaluation. ACM Transactions on Programming Languages and Systems, v. 12, n. 4, p. 537-565, oct. 1990.

[STRASSER96] STRASSER, M.; BAUMANN, J.; HOHL, F. Mole – A Java based mobile agent system. In: Special issues in object-oriented programming: Workshop reader of the 10th European conference on object-oriented programming ECOOP'96, p. 327-334, 1996.

[SUN94] SUN MICROSYSTEMS. The Java language: an overview. Technical report. Sun Microsystems, 1994.

[TENNENHOUSE97] TENNENHOUSE, D.; SMITH, J.; SINCOSKIE, W. et al. A survey of active network research. IEEE Communications, v. 35, n. 1, p. 80-86, jan. 1997.

[VIGNA98] VIGNA, G.(ed.) Mobile agents and security. LNCS v. 1419. Springer-Verlag, 1998.

[VIGNA97] VIGNA, G. Protecting mobile agents through tracing. In: THIRD WORKSHOP ON MOBILE OBJECT SYSTEMS. Proceedings... Finland, June 1997.

[WETHERAL97] WETHERALL, D.; GUTTAG, J.; TENNENHOUSE, D. ANTS: a toolkit for building a dynamically deploying network protocols. 1997. Technical report : MIT.

[WHITE94] WHITE, J. Telescript technology: the foundation for the electronic marketplace. Technical report, General Magic, Inc., 1994.

[WHITE96] WHITE, J. E. Telescript technology: mobile agents. In: BRADSHAW, J.(ed.) Software Agents. AAAI Press/MIT Press, 1996.

[WOOLDRIDGE94] WOOLDRIDGE, M.; JENNINGS, N. Intelligent agents: theory and practice. Oct. 1994.

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

[YEMINI96] YEMINI, Y., SILVA, S. Towards programmable networks. In: IFIP/IEEE INTERNATIONAL WORKSHOP ON DISTRIBUTED SYSTEMS: OPERATIONS AND MANAGEMENT. Proceedings... Italy, oct. 1996.

[YEMINI91] YEMINI, Y.; GOLDSZMIDT, G.; YEMINI, S. Network management by delegation. In: SECOND INTERNATIONAL SYMPOSIUM ON INTEGRATED NETWORK MANAGEMENT. Proceedings... Apr. 1991.

[ZIMMERMAN93] ZIMMERMAN, P. PGP user's guide. Mar. 1993.