AUTRAN DIAS DE ALMEIDA

Comparação entre métodos para roteamento de redes de dados usando redes neurais artificiais

Florianópolis - SC

2001

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

Autran Dias de Almeida

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. Jorge Muniz Barreto, Dr. Inf.

Florianópolis, Dezembro de 2001.

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

Autran Dias de Almeida

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 Programa de Pós-Graduação em Ciência da Computação.

_____
Prof. Raul Sidnei Wazlawick, Dr.
Coordenador do curso

Banca Examinadora

_____
Prof. Jorge Muniz Barreto, Dr.
Presidente da Banca e Orientador

_____
Prof. Mauro Roisenberg, Dr.

_____
Prof. Walter Celso de Lima, Dr.

PREFÁCIO:

É impossível para um homem aprender aquilo que ele acha que já sabe...

Anônimo

DEDICATÓRIA

A meus pais e a minha família, que sempre me deram o incentivo necessário, e o exemplo de vida, e principalmente pelo carinho irrestrito.

AGRADECIMENTOS

Á sinceridade e ao sentido do sorriso de todas as pessoas envolvidas neste processo: família, amigos e professores, através do qual é possível sentir a presença e a infinita bondade de Deus.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

RESUMO

Este documento tem como objetivo apresentar uma série de conceitos e propostas, para elaboração de uma gerência pró-ativa de uma rede de computadores, utilizando as tecnologias de redes neurais Artificiais, e a tecnologia de Agentes autônomos inteligentes.

As redes de computadores estão crescendo em importância, em complexidade a cada dia, e uma ação preventiva no gerenciamento destas redes passou a ser de suma importância para se conseguir bons resultados.

Neste documento serão apresentados os principais problemas que podem afetar o desempenho de uma rede de computadores, bem como algumas soluções baseadas em redes neurais para estes problemas. Também será apresentado o embasamento teórico necessário para utilização das tecnologias propostas.

ABSTRACT

This document objectives to present a series of concepts and proposals, to the elaboration of a Pro-Active management for a computers net, utilizing the technologies of artificial neural nets, and the technology of intelligent autionomous Agents.

The computers nets are rising daily in importance, in complexity. A preventive action for the management of those nets became an important way to obtain strong results.

In this document are presented the principal problems wich can affect the perdormance of a computer net, so of some solutions based in neural nets for this problems. The theoric imbasement, necessary to utilize the proposed technologies will also to be introducted.

CAPÍTULO 1

1.1 INTRODUÇÃO

Quando se trata de informática, é especialmente difícil tentar especular sobre o que pode ou não ser uma tendência para os próximos anos. Estas preposições ficam ainda mais difíceis quando se propõe a utilizar tecnologias que estão em plena ebulição, como é o caso da Internet, agentes e redes neurais, além de outros temas abordados neste documento.

Mas alguns padrões devem ser levados em conta, como por exemplo, que as redes de computadores chegaram para ficar, principalmente as redes baseadas no protocolo TCI/IP, que com o advento da Internet, tiveram um impulso extraordinário. Com é de se esperar, as redes TCP/IP, ainda têm muito que crescer, tanto em tamanho como em complexidade. Esta afirmação abre o caminho para o assunto abordado nesta dissertação, pois quanto maior o crescimento das redes, maior será o esforço necessário no seu gerenciamento.

Neste documento serão apresentados conceitos e propostas, visando aplicar uma gerência pró-ativa das redes de computadores, utilizando tecnologias que podem vir a ser a grande promessa para o futuro da informática, os agentes autônomos inteligentes, e as redes neurais artificiais.

Os agentes, com certeza ainda não ocuparam nem uma mínima parte, do papel que lhes está reservado no cenário da informática atual e futura, e por conseqüência ainda têm muito à oferecer a relação Homem – Máquina, podendo até causar uma incrível

revolução neste aspecto, pois traz uma abordagem completamente nova e interessante, que será de muito ajuda , principalmente com o já citado crescimento das redes de computadores , principalmente a Internet. Em um futuro próximo, os agentes serão largamente utilizados para as funções de pesquisa na Internet, execução de tarefas automatizadas, e quem sabe, poderão evoluir até se tornarem representantes dos seus usuários no mundo virtual.

Em se tratando de redes neurais, tem-se um campo de pesquisa aberto e cheio de desafios. A tecnologia tentando imitar a inteligência humana, de forma a unir a capacidade para cálculos dos computadores tradicionais, com o raciocínio e criatividade do pensamento humano. Com certeza, é uma tecnologia que num futuro próximo, será usada em larga escala em praticamente todas as áreas da informática.

Já a gerência de redes de computadores, deverá evoluir para formas pró-ativas, onde sistemas serão capazes de alertar sobre problemas nas redes de computadores, e até mesmo de evitá-los, dependendo do tipo do problema, e do grau de automatização desejado. Já não é mais interessante trabalhar apenas na correção destes erros, é preciso trabalhar para que estes erros não mais aconteçam. O grau de complexidade e de confiabilidade exigidos das redes de computadores atualmente já não permite este tipo de postura, e a gerência pró-ativa é uma das soluções.

Uma abordagem para a implementação deste tipo de gerência, com certeza necessita da aplicação do paradigma de agentes desenvolvidos com o auxilio das redes neurais artificiais. A utilização dos agentes auxilia a obter as informações necessárias, para implementar redes neurais. Estas, por sua vez, são capazes de reconhecer padrões de erros que poderiam levar à parada da rede, por exemplo. Os agentes também poderiam executar tarefas de coleta de informações, configuração automática, verificação de segurança, e, desta forma, implementar uma manutenção preventiva dos componentes da rede.As redes neurais, também poderiam tratar de problemas que exigem uma solução rápida, em detrimento de uma solução ótima, como é o caso do problema do roteamento nas redes de computadores.

1.2 OBJETIVOS

O objetivo central desta dissertação é apresentar os conceitos necessários para o entendimento do processo de gerência de redes de computadores, aplicando redes neurais, e agentes inteligentes. Também fazer um levantamento dos principais problemas no processo de gerência de computadores, e exemplificar a resolução de um destes problemas através das tecnologias já citadas. Além disso, serão apresentadas avaliações das propostas apresentadas.

Entre os objetivos específicos propostos estão:

- Apresentação da tecnologia de redes neurais Artificiais;
- Apresentação dos conceitos de Inteligência artificial distribuída e do paradigma de agentes;
- Apresentação do processo de gerência de redes de computadores, dando maior ênfase aos processo e ferramentas voltadas para redes TCP/IP.
- Determinação dos principais problemas encontrados no processo de gerência de computadores.
- Apresentação das alternativas para solução de um dos problemas apresentados, utilizando a tecnologia de redes neurais Artificiais;
- Avaliação das alternativas apresentadas..

1.3 LIMITAÇÕES DO ESTUDO

Uma das principais limitações no desenvolvimento deste trabalho foi a falta de um laboratório com uma rede onde pudessem ser implementas as propostas apresentas, para, desta forma, apresentar resultados reais dos métodos utilizados. Sendo assim, serão apresentadas somente avaliações teóricas, conseguidas através da bibliografia, e da análise dos métodos, além de uma simulação por software das soluções apresentadas.

Outra limitação diz respeito à quantidade de problemas que foram levantados no processo de gerência, e desta forma ficaremos restritos ao estudo de somente um dos problemas apresentados, o problema do roteamento nas redes de computadores.

Ainda uma outra limitação, foi quanto à proposição de novos métodos que poderiam ser utilizados na resolução dos problemas, ficando este documento restrito à apresentação a avaliação dos métodos encontrados na bibliografia. A proposição de um

novo método para resolução do problema de roteamento em redes de computadores foi propositadamente deixado para um estudo mais aprofundado.

1.4 ESTRUTURA DO TRABALHO

Este documento está dividido em seis capítulos, onde serão apresentadas as informações necessárias para o cumprimento dos objetivos propostos:

No capítulo 2 será apresentado um tutorial sobre a tecnologia de redes neurais artificiais. Serão apresentados os conceitos básicos, histórico, aplicações, topologias, e os principais métodos desenvolvidos até hoje. O capítulo será encerrado com algumas considerações sobre a construção de aplicações utilizando redes neurais artificiais. A leitura deste capítulo será desnecessária para quem já conhece o assunto, já que as redes neurais que serão realmente utilizadas na resolução dos problemas propostos serão apresentadas no capítulo 5.

No capítulo 3, será apresentada uma introdução à tecnologia de agentes, conceitos, sua utilização, classificação e técnicas de aprendizado. Será apresentado também um tipo de agente específico para Internet, utilizando tecnologia JAVA. O capítulo se encerra com considerações sobre o desenvolvimento de aplicações com agentes.

No capítulo 4, será detalhado o processo de gerência de redes de computadores, a sua divisão, atribuições de cada divisão, conceitos e uma abordagem especial sobre as ferramentas e protocolos destinados à gerência em redes de computadores baseadas em TCP/IP.

No capítulo 5, serão apresentados os principais problemas que podem afetar o desempenho de uma rede de computadores. Dentre estes problemas, foi escolhido o problema de roteamento, para proposição de soluções baseadas em redes neurais artificiais. Serão apresentadas várias propostas para a resolução do problema de roteamento, bem como a avaliação teória de cada uma delas.

No capítulo 6 , serão apresentados os dados de uma silumalção por softwares das soluções propostas, que servirão como subsídio para a avaliação das soluções propostas.

Capítulo 2

2.1Redes neurais Artificiais:

Redes neurais, ou sistemas conexionistas, são sistemas que definem uma nova arquitetura para máquinas capazes de processar informações mediante a propagação de estímulos.

Essa nova arquitetura proposta baseia-se na maneira com que o próprio homem processa informações. É um modelo inspirado no cérebro humano, cujo processamento é devido à grande interação entre as várias unidades de processamento denominado os neurônios, que formam uma intrincada rede onde essas células recebem e transmitem os estímulos que contém a informação ou parte dela.

As redes neurais artificiais tentam implementar fisicamente a maneira de funcionamento do cérebro e obter, como conseqüência, algumas características como capacidade de processar de forma paralela e eficiente à imensa quantidade de dados recebidos, capacidade de aprender com novas situações, capacidade de generalizar (respondendo corretamente a novas situações não vistas), a capacidade de reconhecer padrões rapidamente, dentre outras.

Estes sistemas estão sendo considerados como computadores de sexta geração que introduzem conceitos novos na relação homem /máquina, além de representar um marco na área de inteligência artificial, pois a dois estilos se completam (Figura 2.1).

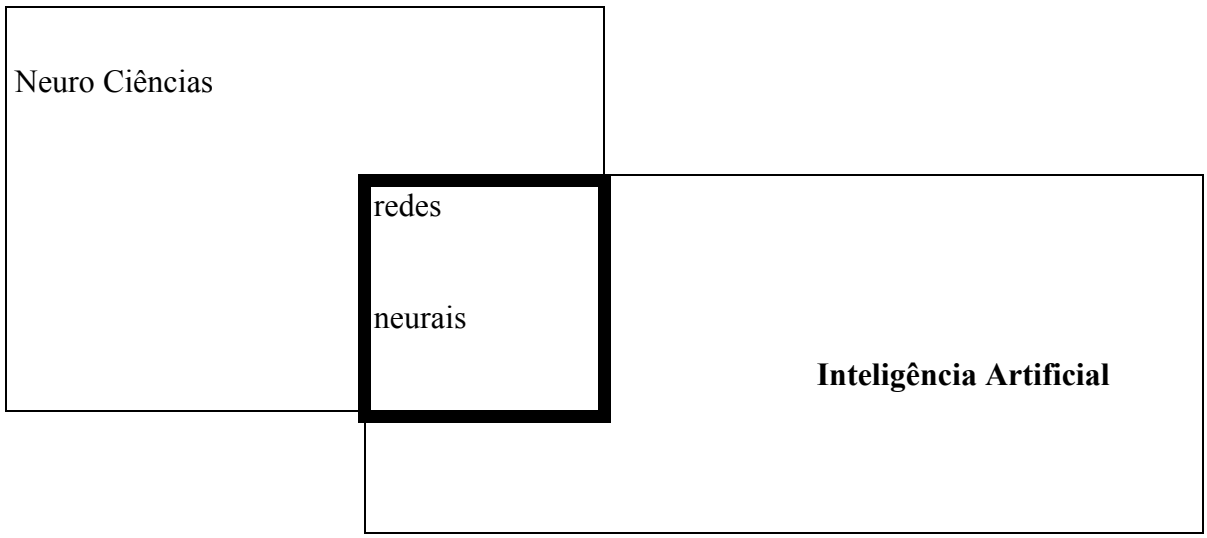

Figura 2.1: Contexto das redes neurais artificiais

2.2 Histórico:

A pesquisa no campo de sistemas neurais artificiais iniciou no começo dos anos 40, cujo interesse inicial estava restrito aos médicos e neurobiologistas teóricos.

A idéia de usar redes neurais é bastante antiga ao nível de inteligência artificial. No entanto, a sua história é marcada por uma fase de efervescência e de descrédito, devido a resultados de pesquisas e avanços tecnológicos.

A história começa em 1943 quando *McCulloch e Pitts* fizeram o primeiro modelo matemático de uma rede neural artificial. Esse modelo era capaz de computar, mas não de aprender.

Seu trabalho fazia uma analogia entre células vivas e o processo eletrônico, simulando o comportamento do neurônio natural, onde o neurônio possuía apenas uma saída, que era uma função de entrada (*threshold*) da soma do valor de suas diversas entradas. (Figura 2.2)

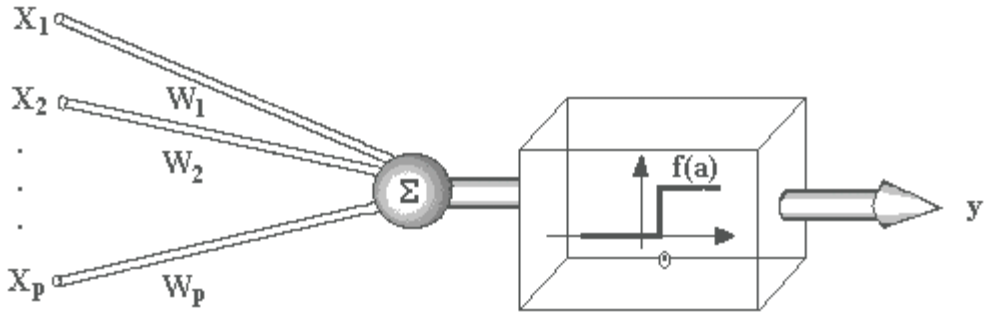

Figura 2.2: O neurônio Artificial de McCulloch e Pitts

Em 1948, *C. E. Shannom* desenvolve trabalhos sobre a "teoria da informação" e despontam interesse do cientista da informação e computação pelo modelo de redes neurais.

Em 1949, o psicólogo *Donald Hebb* demonstrou que a capacidade da aprendizagem em redes neurais vem da alteração da eficiência sináptica, isto é, a conexão somente é reforçada se tanto as células pré-sinápticas quanto as pós-sinápticas estiverem excitadas;

Em seu livro "- *Organization of Behavior*- ", *Hebb* introduz a capacidade de aprender. Ele descreve um sistema de aprendizado por correlação dos neurônios, resultando na elaboração da regra de aprendizagem de *Hebb*.

Os estudos *de K.S. Lashley* em 1950 estabeleceram que o cérebro tem uma representação do conhecimento distribuída. A idéia de *Lanshley* era de que o conhecimento não está localmente armazenado (sobre uma única unidade), mas, ao contrário, está armazenado de uma maneira distribuída. Não existem células especiais para memórias especiais; e sim muitas células que carregam uma parte da memória. Essa idéia é contrária à concepção do processamento dos computadores convencionais, onde a computação se concentra na unidade de processamento central (*UCP*).

Em 1951, *D.Edmond e M. Minaky* construíram a primeira rede neural artificial física. A máquina utilizou a regra de aprendizado de *Hebb* e foi capaz de armazenar 40 padrões de 40 dígitos binários.

Em 1956 no "*Darthmouth College*" nasceram os dois primeiros paradigmas da Inteligência Artificial, o simbólico e o conexionista. A Inteligência Artificial Simbólica tenta simular o comportamento inteligente humano desconsiderando os mecanismos responsáveis por tal. Já a Inteligência Artificial Conexionista acredita que se construindo um sistema que simule a estrutura do cérebro, este sistema apresentará inteligência, ou seja, será capaz de aprender, assimilar, errar e aprender com seus erros.

Em 1957, *Rosemblatt* cria um modelo de rede neural chamado *Perceptron*. Esse modelo era capaz de aprender padrões e de generalizar a partir dos padrões aprendidos.

Ele desenvolveu um mecanismo melhor para aprendizado do que *Hebb*, o procedimento da convergência do *perceptron*. O autor estudou seu modelo com análises matemáticas e simulações em computadores digitais. Entretanto, seus modelos não traziam desempenho satisfatório, tornando-se polêmicos.

Rosemblatt mostrou em seu livro (*Principles of Neurodynamics*) que no modelo dos "*Perceptrons*" os neurônios eram organizados em camada de entrada e saída, onde os pesos das conexões eram adaptados a fim de se atingir a eficiência sináptica(Figura 2.3)

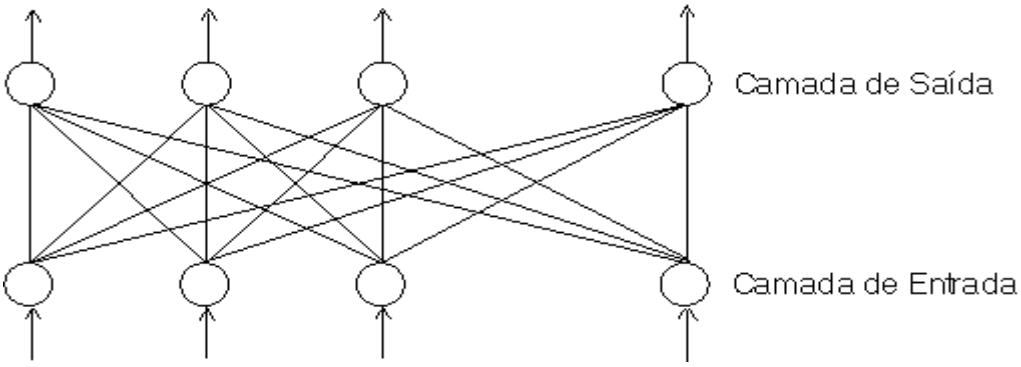

Figura 2.3: rede de Perceptrons proposta por Rosemblatt

O primeiro neuro-computador a obter sucesso (*Mark I Perceptron*) surgiu em 1957 e 1958, criado por *Frank Rosemblatt*, *Charles Wightman* e outros. Devido à profundidade de seus estudos, suas contribuições técnicas e de sua maneira moderna de pensar, muitos o vêem como o fundador da neuro computação na forma em que a temos hoje.

Após *Rosemblatt*, *Bernard Widrow* em 1960, com a ajuda de alguns estudantes, desenvolveu um novo tipo de elemento de processamento de redes neurais chamado de *Adaline*, equipado com uma poderosa lei de aprendizado que, ainda hoje permanece em uso. *Widrow* também fundou a primeira companhia de *hardware* de neuro computadores e componentes.

A rede *ADALINE* (*ADAptative LInear NEtwork*) e o *MADALINE* (*Many ADALINE*) propostas por *Widrow* e *Hoff* utilizavam saídas analógicas em uma arquitetura de três camadas(Figura 2.4).

Em 1962: *Bernard Widrow* desenvolveu um processador para redes neurais e fundou a primeira empresa de circuitos neurais digitais, a Memistor Corporation.

Em 1969, inicia-se um longo período na qual as redes neurais caíram em descrédito. É o chamado "período de inverno das redes neurais". Esse esfriamento ocorre com a publicação do livro "P*erceptron*", de *Minsky* e *Papert*. Neste livro, os autores provam, formalmente, que o perceptron era incapaz de aprender funções como, por exemplo, o Ou Exclusivo. Nesta obra, explicita se que durante o processo de aprendizagem a rede poderia moldar pontos de mínimos locais de energia que produziriam a aprendizagem não satisfatórias.

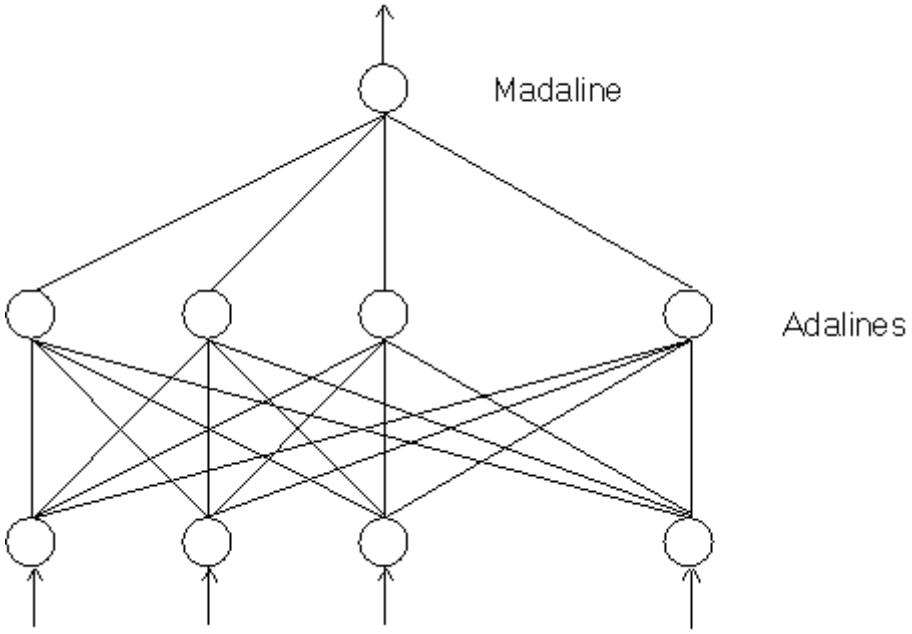

Figura 2.4: redes *ADALINE* e *MADALINE*

Apesar do resfriamento causado, a pesquisa continuou em pequena escala. Nesse mesmo ano, *S. Grosenberg* estudou mecanismos neurais inspirados ao mesmo tempo em percepção, memória, e mais tarde a visão. Inclusive, foi um dos primeiros que analisaram aspectos do modelo de aprendizagem por competição, influenciando muitos outros pesquisadores.

Ainda em 1969, *D. Willshaw* deu importantes contribuições para o entendimento da memória. Ele distribuiu análises matemáticas de modelos de memória e encontrou propriedades interessantes, associadas a vários esquemas modelados. Mais tarde, em colaboração a *Longguet – Higgins*, trabalhou como holofones. Um holofone é uma representação feita da memória do homem que é usada na análise do sistema de memória.

Em 1971, *Tuevo Kohonen* divulgou pesquisa na área de memória associativa auto organizadora, estudando como a mente organiza a informação que ela armazena. Neste mesmo ano, Amari trabalha com a teoria de redes neurais Booleanas, e Anderson analisa memórias associativas lineares (que representavam o conhecimento de forma totalmente distribuída).

Em 1974: *Werbos* lançou bases para o algoritmo de retropropagação (*backpropagation*).

Em 1976, *Bart Kosko* tentou unir redes neurais a pesquisa em lógica nebulosa do modelo chamado "mapas cognitivos difusos".

Rumelhart e *McClelland* criaram em 1977, um novo modelo inspirado no sistema de reconhecimento de voz *HEARPSAY*, da universidade de *Stanford*. Esse modelo foi chamado de processamento distribuído paralelo e levou à criação de vários algoritmos de aprendizagem. Tais como:

- Aprendizagem competitiva
- Máquina de *Boltzmann*
- Propagação Retroativa de erros.

Ainda neste ano, *Hetch-Nielsen* desenvolveu dois neuro computadores, o *Mark III* e o *Darph*, que financiou *o Mark IV*, visando a área de aplicação prática.

Em 1983, foi fundada sua própria empresa dedicada ao desenvolvimento e estudo das redes neurais, que desenvolveu um neuro computador chamado *ANZA*, que adaptava cartões para serem inseridos dentro de um computador *IBM AT*.

Nos anos 80, muitos dos pesquisadores foram bastante corajosos e passaram a publicar diversas propostas para a exploração de desenvolvimento de redes neurais bem como suas aplicações. Porém, talvez o fato mais importante deste período tenha ocorrido quando *Ira Skurnick*, um administrador de programas da *DARPA* (*Defense Advanced Research Projects Agency*), decidiu ouvir os argumentos da neuro computação e seus projetistas. Este ato não só abriu as portas para a neuro computação, como também deu a *DARPA* o status de uma das líderes mundiais em se tratando de "moda" tecnológica.

Outra "potência" que emergiu neste período foi *John Hopfield*, renomado físico de reputação mundial, que se interessou pela neuro computação, e escreveu artigos que percorreram o mundo todo, persuadindo centenas de cientistas, matemáticos, e tecnólogos altamente qualificados a se unirem a esta nova área emergente.

Em 1982, *Hopfield* enviou um artigo para a *National Academy of Science*, provando que as redes não mais poderiam encontrar mínimos de energia. Assim, essas

redes tinham habilidades computacionais coletivas fortes, que poderiam ser utilizadas em áreas como: física, a ciência da computação, psicologia cognitiva, e neuro ciências. É o início da chamada "primavera das redes neurais".

L. Cooper e C. Elbbaum fundaram em 1983, uma companhia, a *Nestor INC*, com interesses em aplicações comerciais, de projetos na área de sistemas de entrada para computador através da escrita manual e gráficos tridimensionais.

Apesar de um terço dos pesquisadores da área terem aderido à mesma pela influência de *Hopfield*, foi em 1986 que este campo de pesquisa "explodiu" com a publicação do livro "*Parallel Distributed Processing*" (Processamento Distribuído Paralelo) editado por David *Rumelhart e James McClelland*.

Em 1987 ocorreu em São Francisco a primeira conferência de redes neurais em tempos modernos, *a IEEE International Conference on neural Networks*, e também foi formada a *International neural Networks Society* (INNS). A partir destes acontecimentos ocorreram a fundação *do INNS journal* em 1989, seguido do *neural Computation* e do *IEEE Transactions on neural Networks* em 1990.

Desde 1987, muitas universidades anunciaram a formação de institutos de pesquisa e programas de educação em neuro computação.

2.3 Redes Neurais Hoje (Modelo Atual)

A simulação de redes neurais em computadores convencionais exige grande número de cálculos, o que não deixa de ser um grande problema. Como os modelos são grandes e muito complexos, o tempo de computação torna-se exponencialmente grande.

Um computador conceitualmente adaptado ao uso de redes neurais seria o neuro computador. Os neuro computadores possuem, portanto, um grande número de processadores que se comunicam para realizar o que uma rede neural naturalmente faz.

Os modelos neurais procuram aproximar o processamento dos computadores ao cérebro. As redes neurais possuem um grau de interconexão similar à estrutura do cérebro, e em um computador convencional moderno a informação é transferida em tempos específicos dentro de um relacionamento com um sinal para sincronização.Na Tabela 2.1, vemos um comparativo entre o cérebro humano e o computador:

Parâmetro	**Cérebro**	**Computador**
Material	Orgânico	Metal e plástico
Velocidade	Milisegundos	Nanosegundos
Tipo de Processamento	Paralelo	Seqüencial
Armazenamento	Adaptativo	Estático
Controle de Processos	Distribuído	Centralizado
Número de elementos processados	10 e 11 a 10 e 14	10 e 5 a 10 e 6
Ligações entre elementos processados	10.000	<10

Tabela 2.1 - Quadro comparativo entre o cérebro e o computador

O mesmo paralelo pode ser traçado comparando o computador com as redes neurais. Para tanto, a comparação não se dará com um computador específico encontrado no mercado, mas sim com o paradigma predominante nos computadores atuais. Este comparativo pode ser observado na Tabela 2.2

Computadores	Neurocomputadores
Executam programas	Aprendem
Executam operações lógicas	Executam operações não lógicas, transformações, comparações
Dependem do modelo ou do programador	Descobrem as relações ou regras dos dados e exemplos
Testam uma hipótese por vez	Testam todas as possibilidades em paralelo

Tabela 2.2 - Quadro comparativo entre computadores e neurocomputadores

2.4 O Neuro Computador

Dois caminhos vêm sendo analisados para tentar se construir neuro computadores: o eletro-ótico e o eletrônico.

- Computadores eletro-óticos são projetados para utilizar a luz na conexão entre as unidades de processamento. Devido ao fato de que as luzes são capazes de se sobrepor sem interferir uma sobre outra, essa abordagem é muito adequada para promover um grande número de conexões.
- Um outro método usado para implementar é eletronicamente. Tais neuro computadores têm as conexões físicas (por meio de filmes ou equivalentes) e usam transistores e chips em suas implementações.

Tamanho e custo não parecem ser fatores limitativos, tendo em vista o avanço tecnológico. Seguindo esse caminho, já se conseguiu fazer um neuro computador eletrônico na forma de placa para encaixar-se em um Slot de um IBM AT com capacidade para 300.000 conexões e 30.000 neurônios.

Outro progresso neste campo foi a construção de um sistema que permite a entrada manuscrita em computador via o bloco digitalizador (sem teclado, usando uma espécie de caneta ótica).

Outro modo de se ver este problema de implementação leva à diferenciação entre redes neurais digitais e analógicas. As redes neurais naturais parecem ser analógicas e este é o caminho mais tentado no mundo atualmente. Entretanto, construir sinapse dentro de chips, elementos que devemmudar para garantir aprendizagem, vem se constituindo um grande desafio. Surgiu, então, uma corrente que tenta implementar as redes neurais através de memórias (usada nos computadores convencionais). Alguns progressos nesta última abordagem vêm sendo obtidos.

Os neuro computadores não substituirão os computadores existentes. Os neuro computadores que vêm sendo projetados são membros subservientes do computador de *Von Neumam*. Programas que usam neuro computadores fazem uma sub-rotina chamar o neuro computador para fazer o seu trabalho especializado. Os neuro computadores farão seus próprios softwares que serão capazes de integrar-se com softwares existentes para criar uma máquina com capacidade de potenciais somados.

Em termos de softwares, programas que utilizam a filosofia de redes neurais já estão sendo comercializados, para aplicações diversas, e sendo bem aceitos mundialmente.

2.5 Aplicações:

Diversas áreas aplicam as técnicas de redes neurais, formando um grande conjunto que, a cada dia vem encontrando espaço maior e acrescentando novas áreas e novos elementos. O esquema da Figura 2.5 mostra algumas áreas de aplicação de redes neurais.

Uma área que sempre despertou grande interesse foi a de memória associativa, também conhecida como memória endereçável por conteúdo.Uma memória associativa processa todas as possíveis saídas para uma dada entrada, eventualmente encontrando a saída apropriada em um tempo constante. Outra tarefa que a rede neural é capaz de realizar é criar representações generalizadas da entrada apresentada. Estendendo essa idéia, ela é capaz de processar informações (responder corretamente) mesmo quando somente é dada uma parte da entrada.

Esta é uma característica útil em redes neurais, em que ela não precisa ter fornecido todas as entradas de informação para conseguir a saída adequada.
Similarmente, se uma informação parcialmente incorreta ou confusa é dada para a rede, ela fará uma "suposição melhor", processando uma saída de uma entrada dada, baseado em cima da representação generalizada interna.

Uma outra tarefa em que os modelos descritivos são bem-sucedidos é o problema de múltiplos simultâneos coagidos. O sistema é capaz de processar muitas entradas simultaneamente e produzir uma saída baseada naquelas entradas. Por causa da habilidade de produzir uma grande quantidade de informações simultaneamente, a robótica é uma aplicação que está bem favorecida nesta área.

Diversas aplicações de redes neurais artificiais são encontradas nas áreas de finanças, de recursos humanos ou de *marketing* (referência para exame). Ela vem propor um uso diferenciado do enorme volume de dados estocados nos computadores de empresas, que muitas vezes são pouco ou mal utilizados, transformando-os em informação útil à tomada de decisão. Mas isto não é feito através da proposição de telas sofisticadas de consulta e nem através dos sistemas passivos de análise de cenários

como os Sistemas de Apoio à Decisão tradicionais. Sistemas baseados em redes neurais propõem uma ajuda ativa à tomada de decisão.

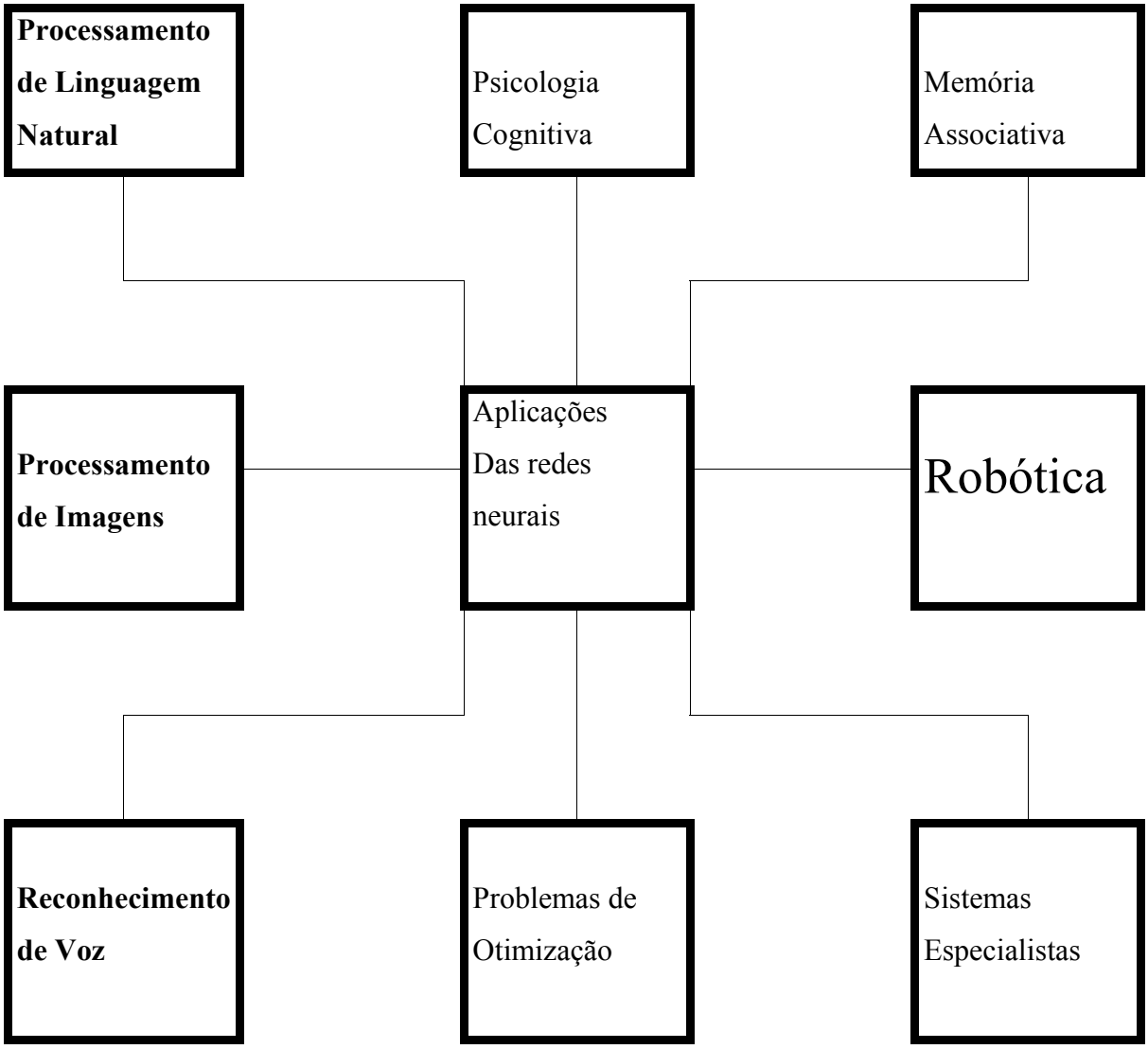

Figura 2.5: Áreas de aplicação das redes neurais.

As redes neurais têm sido utilizadas na área de administração para criação de bases de conhecimento. Uma das aplicações importantes é a avaliação de risco de crédito. As redes são, muitas vezes, comparadas neste tipo de aplicação com métodos estatísticos, em análise de dados. As redes neurais têm duas grandes vantagens em relação às técnicas estatísticas tradicionais:

- Em primeiro lugar, elas não trabalham somente com dados numéricos, mas também com dados qualitativos. Por exemplo, na avaliação de uma empresa, a rede pode levar em consideração até mesmo informações como o estilo de gestão efetuado pela empresa.
- Em segundo lugar, quanto maior o volume de dados, melhor será seu aprendizado, e melhor será a qualidade das previsões. Uma rede neural artificial consegue encontrar informações e relações entre os dados que não são possíveis de serem encontradas com técnicas estatísticas convencionais.

As aplicações de redes neurais são inúmeras. Muitos recebem sua primeira introdução lendo a respeito das técnicas no prognóstico de mercados financeiros. Grupos de investimento conhecidos utilizam redes neurais para analisar pelo menos uma parte do mercado financeiro e fazerem suas seleções.

O reconhecimento ótico de caracteres (OCR) é outro tipo de aplicação que já existe e está crescendo, e em breve estaremos em constante contato com esse tipo de aplicação. Outras aplicações bem sucedidas das técnicas de redes neurais artificiais são: análise de pesquisa de mercado, como acima citado, controle de processos industriais, aplicações climáticas, e identificação de fraude de cartão de crédito. Um banco americano, chamado *Mellon Ban, k* instalou um sistema de detecção de fraudes de cartão de crédito implementado com técnicas de redes neurais e os prejuízos evitados pelo novo sistema conseguiram cobrir os gastos de instalação em seis meses. Vários outros bancos começam a utilizar sistemas baseados em redes neurais para controlar fraudes de cartão de crédito. Estes sistemas têm a capacidade de reconhecer uso fraudulento com base nos padrões criados no passado com uma precisão melhor que em outros sistemas.

Uma experiência interessante feita por *T. Sojnowoki* e *C. Rosenberg* foi a chamada *NetTalk*, que fazia a conversão de textos escritos em leitura sonora. Esse modelo mapeou a entrada escrita em fonemas de saída. Os fonemas eram passados por um sintetizador chamado *DECTalk*, que produzia o som. O sistema aprendeu a ler o texto em voz alta sendo apresentado, sucessivamente, uma janela de sete letras do texto escrito e usando a letra central como saída que o fonema adequado poderia representar. Pelo mapeamento das sete letras do fonema adequado, a rede foi incorporando a aprendizagem de sensibilidade de contexto, dentro do processo de conversão. A letra

central das sete é a letra que representava o fonema e as três letras precedentes, bem como as três letras subseqüentes, eram o contexto para que a letra fosse representada (mapeava a influência que cada uma exercia sobre a outra). Depois que sete letras fossem apresentadas, era feito um deslocamento a fim de aprender o som da letra seguinte e o processo se repetia para todo o texto.

O modelo usou o algoritmo de propagação retroativa de erros de *Rumelhart*, *Hilton* e *Willians*. No estágio inicial, a saída emitia um som semelhante ao balbuciar de um bebê. Enquanto o treinamento evoluía, os sons emitidos se tornavam mais claros discerníveis como os de uma criança pequena, inclusive no caso de uma mesma letra possuir sons diferentes dependendo do contexto. Por exemplo, pronunciando o "A" corretamente em "*say*" e em "ran". O tempo de aprendizagem foi de apenas três meses, sendo significativamente bem menor do que o tempo de desenvolvimento dos programas comerciais que usam manipulação simbólica, e que geralmente demoram anos para elaborar as regras aprendidas naturalmente pela rede neural.

Uma aplicação em processamento de linguagem natural foi criada pelos cientistas cognitivistas, *D. Rumelhat e J. McClelland*, cuja rede neural aprendia o tempo passado de verbos da língua inglesa.

O objetivo deste estudo era de constatar se tal sistema adquiria as regras de formação do passado dos verbos do mesmo modo que as crianças. O modelo trabalhou com representações fonéticas, onde a entrada representava a raiz do verbo e as saídas indicavam o tempo passado a ser aprendido. A regra de aprendizagem usada foi a máquina de *Boltzmann*.

Uma criança passa através de três estágios quando adquire as regras para formação do passado dos verbos da língua inglesa:

- No primeiro estágio não há evidência de nenhuma regra sendo formada pela criança.
- O segundo estágio mostra conhecimentos explícitos de regras de lingüística onde ela pensa que todos os verbos são regulares, por exemplo, regularizando o verbo "*come*" em "*comed*".
- No estágio final ambas as formas de verbos regulares e irregulares existem; a criança aprendeu todas as regras e exceções.

Os testes foram conduzidos em vários estágios do treinamento da rede. A experiência de *Rumelhart* e *McClellnad* mostrou que o modelo utilizado passou pelos mesmos estágios conforme o treinamento evoluiu.

Outro resultado do treinamento foi a capacidade da rede neural em responder a verbos jamais vistos antes. Esse resultado mostrou que a rede foi capaz de abstrair o que tinha aprendido. Aplicando seus conhecimentos (capacidade de generalização) e sua habilidade de tratar entradas não totalmente definidas ou confusas.

Processamento de linguagem natural como uma aplicação de redes neurais tem sido estudada por *G. Cottril* e *S. Small* da *University of Rochester*. Eles construíram uma rede que tirava a ambigüidade da palavra mediante análise do contexto. Um exemplo está em como a uma rede pode entender a diferença do significado da palavra "*Throw*" transmite nas sentenças "*Bob Throw a Figth*" e "*Bob Throw a Ball*".

Outro trabalho nesta área que envolve redes neurais foi feito por *M. Fanty*, também da *University of Rochester*. Fanty desenvolveu um algoritmo que constrói uma rede neural para uma gramática livre de contexto, ou seja, a posição de uma palavra não influencia na determinação de seu significado, e usa esta mesma rede como um tempo interpretador desta gramática.

Um dos melhores resultados práticos da aplicação de redes neurais se deve à *Tuevo Kohonen*. Ele implementou uma máquina capaz de passar discursos pronunciados em finlandês e japonês para forma escrita em duas fases: o som é pré-processado, passando a uma rede neural anteriormente treinada e que obtém uma descrição fonética da seqüência dos sons obtidos. Esta descrição fonética é analisada por um software que produz a forma escrita final (ortograficamente correta). Para obter a descrição fonética, a rede neural é treinada a formar um mapeamento de sons articulados. O mapeamento preserva as regularidades intrínsecas aos fonemas na medida da proximidade entre eles.

O processamento de imagens é uma área que tem recebido bem as técnicas de redes neurais. Um exemplo é a rede de processamento de visão de *Mingolla* e *Grossborg*, que demonstrou que a segmentação do direcionador de imagens (que faz com que esta imagem seja deslocada, rotacionada ou alterada de tamanho), não influi no reconhecimento da imagem por parte da rede. Outro exemplo da aplicação de redes neurais no reconhecimento de imagem é o modelo construído por *N. Farhat* da *University of Pensilvania*, que se dedica a treinar redes neurais para discernir imagens

de radar de várias aeronaves. O resultado da aplicação mostra o reconhecimento de uma bomba com somente 20% da imagem fornecida.

A *Nestor INC.* desenvolveu uma aplicação de redes neurais que aceita escrita manual sobre um bloco digitalizador como entrada do computador. A rede aprende as idiossincrasias da escrita manual de qualquer pessoa, permitindo uma entrada mais direta no computador.

O interesse dessa companhia que está em uma rede que aceite escrita japonesa como entrada e converta-a para a forma do caractere do computador, eliminando a difícil tarefa de entrada no computador.

A pesquisa sobre a fala humana também está sendo explorada. As aplicações mais atuais em estudo de palavras tentam descobrir um método para o reconhecimento independente do locutor. Podemos citar como exemplo softwares como o Via Voice, da IBM.

J. Elman e *D. Fisher* usaram redes neurais na tentativa de descobrir as características escondidas na linguagem que permite aos homens distinguir as palavras. Outro projeto de descrição da linguagem e está em usar padrões temporariamente combinados para executar o reconhecimento independente do locutor.

Em outra aplicação, físicos da universidade de *Brown* têm usado uma rede simulada de conhecimentos médicos para relatar informações de casos, sintomas e tratamentos.

E assim existem várias pesquisas em torno de redes capazes de auto- organização, que classificam seu ambiente; tarefas de processamento paralelo com a iniciativa de defesa estratégica ou previsão do tempo; representação de sistemas de produção e muitas outras mais. No futuro, redes neurais estarão fazendo parte do nosso dia-a-dia, assim como os chips de transistores;

Uma outra aplicação prática de redes neurais está presente no cotidiano dos norte-americanos. Os Correios daquele país utilizam um sistema para reconhecer automaticamente códigos de endereçamentos postais. Do dia-a-dia para o entretenimento, a indústria cinematográfica de Hollywood tenta prever o sucesso de um determinado filme com a ajuda de redes neurais. Um ser humano pode detectar tendências e fazer projeções, mas este sistema pode lançar mão de um número infinitamente maior de variáveis o que torna seu prognóstico muito mais confiável.

O controle da qualidade industrial é outro campo de aplicação para redes neurais. Algumas grandes retíficas americanas utilizam o sistema na revisão preventiva de motores. Isto porque ele prevê quando pode acontecer um problema a partir da análise de seus ruídos, por exemplo.

Atualmente o Brasil se encontra entre os quinze maiores países do mundo em utilização de redes neurais Artificiais. Por isso destacaremos alguns projetos desenvolvidos e em desenvolvimento em nosso país.

Um projeto bastante interessante vem sendo desenvolvido por um grupo de pesquisadores de cinco instituições acadêmicas brasileiras para a Marinha do Brasil. O sistema é capaz de reconhecer, processar e analisar taticamente cartas náuticas eletrônicas. O coordenador do projeto explica que a função do sistema será substituir os humanos no desenvolvimento de algumas tarefas. Dentre elas, o reconhecimento de alvos e de imagens cartográficas. "A partir de informações fornecidas por radares será possível analisar uma série de fatores, como a segurança das regras de navegação, sem ocupar uma equipe", complementa. As Universidades envolvidas são: Universidade Federal de Pernambuco, Rio Grande do Sul e de Goiás, a UNICAMP e a USP de São Carlos.

Um outro trabalho interessante vem sendo desenvolvido na Universidade Federal de Santa Catarina, sob o Titulo de “Reconhecimento de Faces Humanas Através de Técnicas Conexionistas Aplicadas à Formas 3D” que usa redes neurais artificiais, no reconhecimento da Face Humana, aproveitando uma das grandes características das rede neurais, já que elas são mais tolerantes a pequenas variações no posicionamento das formas do que as abordagens tradicionais.

Existe também um sistema desenvolvido na UNICAMP que realiza através da tecnologia de redes neurais o prognóstico de casos reais de trauma craniencefálico, assim como recomenda ou não a realização de craniotomia.

A Petrobrás registrou, nos seus gastos anuais, uma economia de RS 15 milhões, como resultado da parceria firmada entre o Centro de Pesquisa da Petrobrás (CENPES) e o Laboratório de Engenharia de Algorítmos e redes neurais da PUC-RIO, através do Projeto TAO - Técnica Avançada em Otimização.O Projeto TAO se baseia no uso de computadores e redes neurais para o trabalho de regulagem das torres de refino,

tradicionalmente feito de maneira manual, sem a precisão de uma medição computacional.

No INPE, foi desenvolvido um classificador de imagens que utiliza técnicas de segmentação, e estão sendo conduzidos estudos em redes neurais em conjunto com o Centro Científico IBM/Rio.

Outro trabalho bastante interessante é desenvolvido pelo Professor Ricardo Mendes Júniorda Universidade Federal do Paraná. Este estudo aborda a utilização de redes neurais no processo de gerencimento de construção civil.

Apesar destas iniciativas, segundo o professor da USP São Carlos, André Ponce de Leon, o Brasil ainda não produziu nenhum resultado de ponta. "É uma área nova, mas com muitas possibilidades de crescimento. Até porque já estamos entre os 15 países que mais estudam estes sistemas", complementa.

2.6 O Modelo de Redes Neurais Artificiais

2.6.1 A Motivação:

A partir do momento em que as máquinas começaram a evoluir, um grande desejo do homem tem sido a criação de uma máquina que possa operar independentemente do controle humano. Uma máquina cuja independência seja desenvolvida de acordo com seu próprio aprendizado e que tenha a capacidade de interagir com ambientes incertos (desconhecidos por ela), uma máquina que possa ser chamada de autônoma, inteligente ou cognitiva.

O sucesso de uma máquina autônoma dependeria única e exclusivamente de sua capacidade de lidar com uma variedade de eventos inesperados no ambiente em que opera. Estas máquinas teriam maior capacidade de aprender tarefas de alto nível cognitivo que não são facilmente manipuladas por máquinas atuais, e continuariam a se adaptar e realizar tais tarefas gradativamente com maior eficiência, mesmo que em condições de ambiente imprevisíveis. Então, seriam muito úteis onde as ações humanas fossem perigosas, tediosas ou impossíveis; como em reatores nucleares, combate ao fogo, operações militares, exploração do espaço a distâncias em que uma nave espacial estaria fora do alcance do controle na terra, porém enviando informações.

2.6.2 A Inspiração:

As redes neurais, como já foi mencionado, inspiraram-se nos estilos de funcionamento do cérebro humano, que pode ser comparado a um poderoso computador, onde todo nosso conhecimento está armazenado na força das conexões e na sensibilidade dos neurônios que estão conectados a milhares de outros entre si.

O modelo do cérebro, entretanto, mostra a grande diferença em relação ao modelo de Von Neumann. A questão fundamental é a aprendizagem. Os computadores convencionais não conseguiram simular satisfatoriamente tarefas que os seres humanos realizam com naturalidade, como interpretar uma imagem, entender uma língua, lembrar um fato inexato, reconhecer um som. Quando conseguem, fazem- na com muita lentidão e dificuldade.

Umas das explicações mais aceitas na comunidade científica, é que o cérebro e os computadores operam em estilos diferentes. O modo de operação do cérebro é mais conveniente para as atividades citadas anteriormente e o modo de operação do computador é mais adequado para realizar operações matemáticas.

O computador pode simular qualquer estilo, mas à medida que esse estilo vai diferenciando do seu, ele vai ficando mais ineficiente.

Enquanto que o homem é capaz de fazer cálculos aritméticos assim como o computador, mas o tempo despendido para essa tarefa é significativamente maior.

O ciclo de tempo é o tempo tomado para processar uma simples parte da informação de entrada para a saída. O ciclo de tempo dos computadores mais avançados é de 1 ns .O nível médio do ciclo de tempo para um neurônio do cérebro é de 2 milisegundos.

Para fazer operações aritméticas um supercomputador equivale a um bilhão de pessoas. Para entender uma imagem uma pessoa equivale a milhares de supercomputadores.

A memorização dos dados pelo homem e o acesso a eles é lento, na máquina de *Von Neumann* a memorização é instantânea e o acesso rápido.

O desempenho em tarefas de reconhecimento de padrões é baixíssimo nos computadores e alto nos homens.

As redes neurais procuram imitar a arquitetura do cérebro humano de forma a beneficiar-se dos aspectos naturais a este último, como por exemplo, pequenas falhas

em nosso sistema nervoso (como a morte dos neurônios) não influem e nossa capacidade de processamento de dados, assim uma rede neural deve ter uma tolerância a falhas, o que não ocorre nos computadores convencionais.

A tendência é tentar aliar o bom desempenho das máquinas de *Von Neumann* (em termo de velocidade de processamento e precisão) com os aspectos naturais do cérebro humano.

Quando se descobrir os mecanismos que o homem usa nas suas atividades, essa distância diminuirá, pois será possível introduzi-los na estrutura de uma máquina.

2.6.3 Funcionamento:

A unidade básica de construção do sistema nervoso é o neurônio, que conduz a interligação da informação entre as várias partes do corpo. Consiste de uma célula chamada corpo, e dos axônios ou fibras nervosas, que o conectam às células de outros neurônios. As junções entre os neurônios ocorrem ambas no corpo da célula ou em extensões ramificadas do corpo da célula chamados dendritos. Tais junções recebem o nome de sinapses.

Axônios e dendritos podem ser tratados do mesmo modo que condutores isolados para transmissão de sinais elétricos para o neurônio.

Como pode ser percebido na Figura 2.6, o neurônio possui vários pontos de entrada de estímulos, que são os dendritos, e apenas um ponto de saída, o axônio. Deste modo, o neurônio não pode responder com dois finais diferentes, sua resposta é a única, embora ela possa atingir vários outros neurônios, já que o axônio pode se ramificar e conectar- se a dendritos de várias células nervosas.

De um modo simplificado, os estímulos chegam aos neurônios vindos do mundo exterior através de seus dendritos. Em seu corpo o neurônio é levado para um nível de ativação em função de características próprias e de estímulos recebidos. O axônio propaga o próprio nível de ativação, ou ainda, um sinal proveniente da aplicação de uma função de saída a este nível, ao seu mundo exterior.

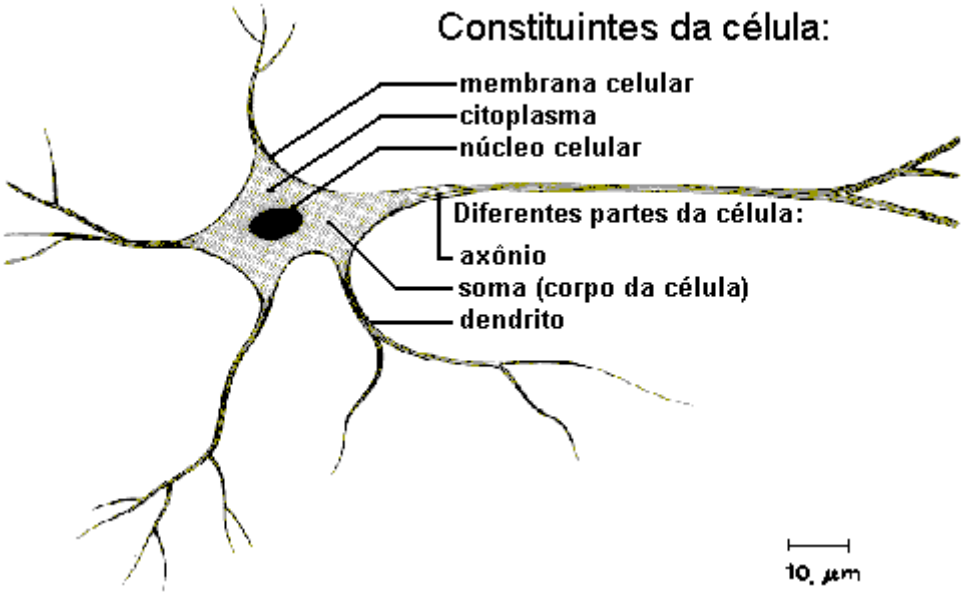

Figura 2.6: O neurônio

A função do neurônio dentro da rede neural é que determina a comunicação deste com seu mundo externo. Se o neurônio estiver no nível de entrada, ou seja, for uma unidade que recebe os estímulos externos do circuito neural, a comunicação é feita através de sua sensibilidade ao mundo exterior. Os neurônios do nível de entrada são denominados de sensores da rede neural. A nível interno a comunicação é feita através das sinapses, que interferem no sinal transmitido, multiplicando-o por um valor denominado de peso da sinapse ou força da conexão, alterando assim o sinal que é recebido por outro neurônio.

De forma geral, a operação de uma célula da rede se resume em (Figura 2.7, Equação 2.1):

- Sinais são apresentados à entrada;
- Cada sinal é multiplicado por um peso que indica sua influência na saída da unidade;
- É feita a soma ponderada dos sinais que produz um nível de atividade;

Se este nível excede um limite (*threshold*) a unidade produz uma saída;

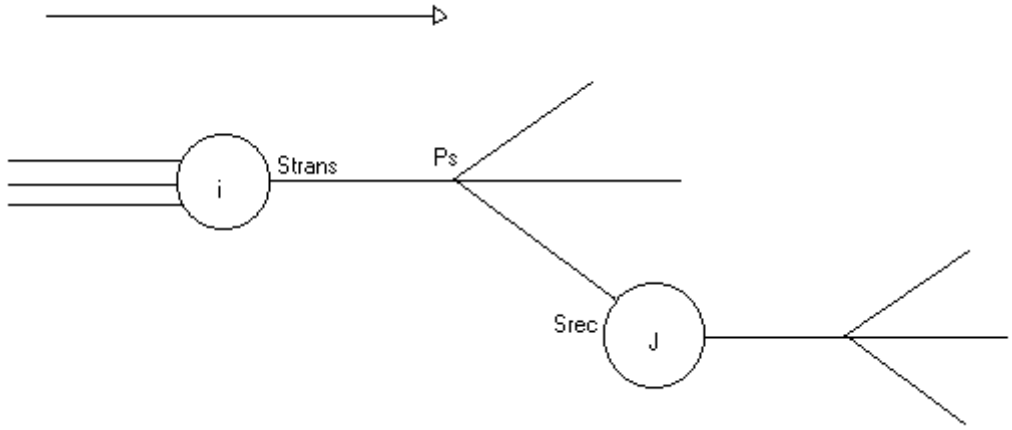

Figura 2.7: Representação de uma rede neural Artificial

Onde:

Strans é o sinal transmitido pelo neurônio i

Srec é o sinal transmitido pelo neurônio J

Ps é o peso da sinapse que interliga os neurônios i e J.

Então:

$$Srec = Ps * Strans \quad (2.1)$$

Para caracterizar o modelo geral de redes neurais, é preciso fazer sua diferenciação entre o estado de ativação, que é o resultado da combinação dos estímulos de entrada, e do estilo de processamento e sensibilidade próprios, e a resposta ao neurônio, que é o sinal transmitido através de seu axônio.Geralmente esta diferenciação não existe em grande parte dos modelos implementados. Todavia, para um caso geral, a saída é o resultado da aplicação de uma função de saída sobre o nível de ativação.

Para que um estado de ativação seja atingido, um neurônio combina todos os sinais recebidos. Normalmente se usa uma simples adição para obter o valor dessa combinação, que é a chamada de entrada líquida. O neurônio aplica a função da ativação sobre a entrada líquida obtendo o estado de ativação.

Se os neurônios possuem função de saída, esta é aplicada sobre o estado de ativação determinando o estímulo de saída que será enviado a outros neurônios através da sinapse ou para o próprio nível de saída da rede. Se a rede não possuir função de saída, o grau de ativação corresponde à própria saída da rede.Este processo pode ser mais bem visualizado na Figura 2.8.

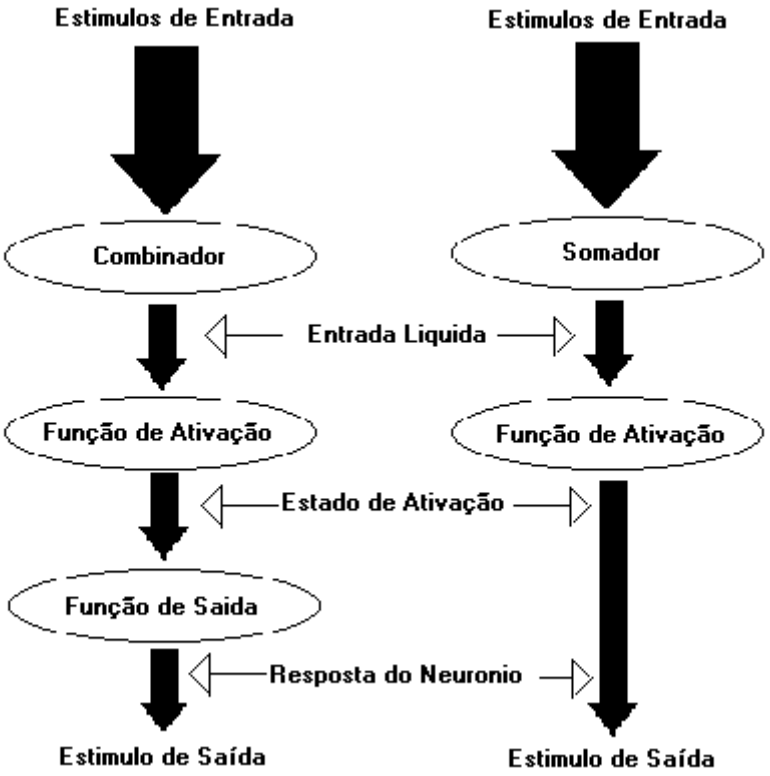

Figura 2.8:Representação da seqüência de eventos em um Neurônio Artificial

Os graus de ativação de um neurônio podem pertencer a um conjunto discreto ou variar continuamente. No caso discreto (Equação 2.2) os valores típicos são os valores da ativação bipolares {-1,+1}. Já no caso contínuo (Equação 2.3) é usual o intervalo [0,1], sendo que existe em casos em que o valor do grau de ativação pode ser qualquer valor real, ou seja, estar no intervalo entre infinito e infinito negativo[-∞ ,∞].

As funções ativação comumente usadas nos algoritmos de aprendizagem são:

- Função degrau Unitário (Discreta):

$$Ati(ent\_liq) = \begin{cases} 1 \text{ se } ent\_liq >= 0 \\ 0 \text{ se } ent\_liq <= 0 \end{cases} \quad (2.2)$$

- Função Sigmóide(Contínua):

$$Ati(ent\_liq) = 1 / 1+ e^{-entliq} \quad (2.3)$$

Onde:

Ati é a ativação do *I-ésimo* neurônio

Ent_liq é a entrada líquida

Os graus de ativação que atingem os neurônios não são determinados exclusivamente pela entrada líquida. Há um outro fator que determina diferentes ativações para a mesma entrada líquida chamado limiar de ativação (Threshold*)*, que define a sensibilidade de cada neurônio. Isto significa que uma entrada líquida acima deste limiar faz com que o neurônio fique excitado, e para valores abaixo deixa-o inibido.

A distinção entre um neurônio ativo e inativo em uma função de grau unitário é tal que leva o neurônio a um entre dois estados possíveis (0 ou 1). Quando a função de ativação é a função sigmóide, o estado de ativação do neurônio pode assumir qualquer valor entre 0 e 1.

Em comunicações digitais é importante obter respostas binárias da rede. Neste caso, fixa-se um limite que funcionará como um limiar de decisão, acima do qual o neurônio é considerado excitado e caso contrário, inibido. Este limiar é chamado de tolerância de ativação. Usualmente considera-se os seguintes valores:

- Neurônio ativo: 1
- Neurônio inibido: 0
- Tolerância de ativação: 0,5

As sinapses também recebem o mesmo tratamento, classificando-se em exitadoras ou inibidoras.

Em uma rede neural podemos agrupar os neurônios em níveis. O nível que recebe estímulos do meio ambiente é chamado de nível de entrada (os que contêm sensores). O nível que contém os "mostradores" da resposta da rede é chamado de nível de saída.

Os níveis intermediários que permitem representações internas de conceitos são chamados internos, escondidos ou intermediários. Os neurônios de um nível não se comunicam entre si (não existe sinapse entre eles). Os neurônios de um certo nível se comunicam apenas com todos os níveis anterior e posterior (no sentido da propagação de estímulos, ou seja, do nível de entrada para o nível de saída).

Para melhor entendermos esses conceitos, analisaremos a rede neural que resultou na aprendizagem da função ou exclusivo, representada na Figura 2.9, com os resultados demonstrados na Tabela 2.3.

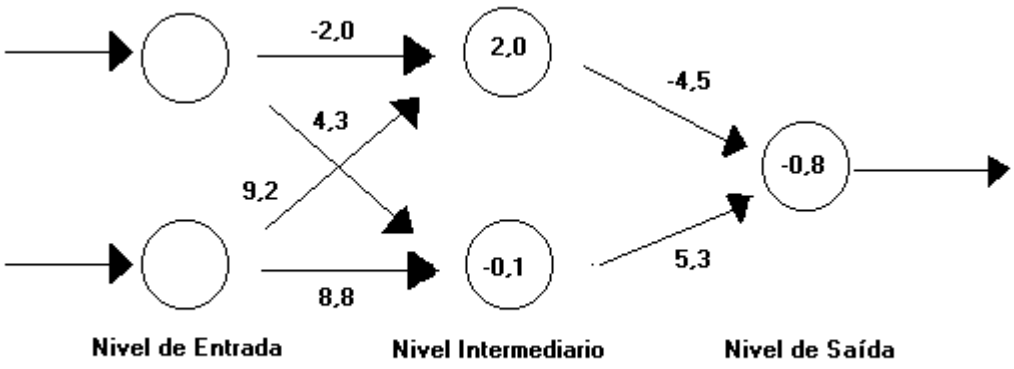

Figura 2.9: Aprendizagem da função Ou Exclusivo

Neurônio	Nível	Valor da Ativação
1	Entrada	1
2	Entrada	1
1	Intermediário	sigmóide [-2,0 * 1 + 9,2 * 1 –2,0] = A1
2	Intermediário	sigmóide [4,3 * 1 + 8,8 * 1 – (- 0,1)] = A2
1	Saída	sigmóide [- 4,5* A1 + 5,3 * a2 – (- 0,8)] = A3

Tabela 2.3: Resultados da aprendizagem da função Ou Exclusivo

Após esse exemplo, o estilo de computação deve ter ficado mais claro. Deve-se notar que os neurônios são capazes de combinar os estímulos de entrada e, através do processamento próprio, atingir uma certa ativação e propagar uma resposta, ao contrário dos computadores convencionais que realizam cálculos simples em UCP´s

2.7 Classificação de Redes Neurais Artificiais

Um dos objetivos da pesquisa sobre redes neurais na computação é desenvolver morfologias neurais matemáticas, não necessariamente baseadas na biologia, que podem realizar funções diversas. Na maior parte dos casos, modelos neurais são compostos de muitos elementos não lineares que operam em paralelo e que são classificados de acordo com padrões ligados à biologia.

Quando um processo é criado visando utilizar aspectos de redes neurais, este começa com o desenvolvimento de um neurônio artificial ou computacional baseado no entendimento de estruturas biológicas neurais, seguido do aprendizado de mecanismos voltados para um determinado conjunto de aplicações. Ou, em outras palavras, seguindo as três etapas:

- O desenvolvimento de modelos neurais motivados por neurônios biológicos;
- Modelos de estruturas e conexões sinápticas;
- O aprendizado das regras (um método de ajuste de pesos ou forças de conexões internodais)

Por causa de diferenças entre algumas ou às vezes todas as entidades envolvidas, diferentes estruturas de redes neurais tem sido desenvolvidas por pesquisadores. Do ponto de vista estrutural, a arquitetura de redes neurais pode ser classificada como estática, dinâmica ou *Fuzzy*, e de única camada ou múltiplas camadas. Além disso, diferenças computacionais surgem também quando se trata da maneira com que são feitas as conexões existentes entres os neurônios. Estas conexões podem ser estritamente em um sentido, em ambos os sentidos, lateralmente conectadas, topologicamente ordenadas ou híbridas.

2.8 Topologias de Redes Neurais Artificiais

2.8.1 Disposição dos Neurônios:

De acordo com *Rummelhart*, a rede neural deve possuir no mínimo duas camadas: a de entrada de dados e a da saída dos resultados. Como a rede apresenta desempenho muito limitado com somente duas camadas, a adição de uma camada intermediária faz-se necessária. Neste tipo de configuração, cada neurônio está ligado com todos os outros das camadas vizinhas, mas neurônios da mesma camada não se comunicam, além da comunicação ser unidirecional, apresentando assim um comportamento estático.

Já a rede neural de *Hopfield* apresenta comportamento dinâmico e fluxo de dados multidirecional devido à integração total dos neurônios, desaparecendo assim a idéia das camadas bem distintas. Com isso seu funcionamento é mais complexo, havendo certas complicações, seja na fase de aprendizado, quanto na fase de testes. Seu uso é direcionado a problemas de minimização e otimização, como por exemplo, de percurso de caminhões.

Há pesquisadores como *Hecht - Nielsen*, que afirmam que com apenas uma camada oculta já é possível calcular uma função arbitrária qualquer a partir de dados fornecidos. De acordo com *Hecht - Nielsen*, a camada oculta deve ter por volta de 2i+1 neurônios, onde i é o número de variáveis de entrada.

Outros, no caso de *Cybenko*, defendem o uso de duas camadas ocultas.

No caso de *Kudricky*, empiricamente observou-se que para cada 3 neurônios da primeira camada oculta era preciso um da segunda camada.

Já *Lippmann* afirma que a segunda camada oculta deve ter o dobro de neurônios da camada de saída. No caso de apenas uma camada oculta, ela deverá ter s(i+1) neurônios, onde *s* é o número de neurônios de saída e *i* o número de neurônios na entrada.Outros autores definem o número máximo como(Equação 2.4):

$$Omax = c / 10.(i+s) \qquad (1.4)$$

Onde:

s: número de neurônios de saída

i: número de neurônios na entrada

Em redes pequenas o número de neurônios da camada oculta pode ser a média geométrica entre o número de neurônios de entrada pelo número de neurônios de saída.

Independente de cada abordagem, quanto mais camadas de neurônios, melhor é o desempenho da rede neural, pois aumenta a capacidade de aprendizado, melhorando a precisão com que ela delimita regiões de decisão. Estas regiões de decisão são intervalos fixos onde a resposta pode estar. A camada de entrada possui um neurônio especial chamado de "*bias*" e serve para aumentar os graus de liberdade, permitindo uma melhor adaptação, por parte da rede neural, ao conhecimento à ela fornecido.

2.9 Tipos de Redes Neurais Artificiais:

2.9.1 Redes Diretas

As redes diretas são redes neurais cujo grafo não possui ciclos e podem ser representadas por camadas. Por exemplo, os neurônios que recebem sinais de excitação do meio externo estão na camada de entrada; os neurônios que estão na saída são chamados de camada de saída.

Neste modelo, os possíveis valores para ativação de um neurônio são todos números reais. A resposta do neurônio é seu próprio estado de ativação, a sua função de saída é a função identidade, que é obtida pela soma dos estímulos recebidos.

Para a aprendizagem, geralmente se usa a regra de *Hebb*, ou a Regra Delta, a serem descritas posteriormente, e que atuam como associadoras de padrões.

A modelagem matemática é simples. A construção da matriz de conectividade *m* (que armazena os pesos das sinapses), é suficiente para descrever o processo.

Suponhamos que temos uma rede de dois níveis com *i* neurônio de entrada (Matriz *E* com dimensão 1x *i*) e *j* neurônio de saída (matriz *s* com dimensão *i x j*).

A matriz de conectividade é, portanto, a matriz (*m i X i*), onde, *SNPij* é a sinapse que ligam neurônio transmissor i ao receptor *j*. Em termos matriciais temos(Tabela 2.4,2.5, 2.6):

$$S = E * M = \begin{bmatrix} s1 \\ s2 \\ \downarrow \\ sj \end{bmatrix} = \begin{bmatrix} e1 * snp11 & ... & ei * snpi1 \\ e1 * snp12 & \dots & ei * snpi2 \\ \downarrow & ... & \\ e1 * \ snp1j & ... & ei * snpij \end{bmatrix}$$

Tabela 2.4 Matriz de Conectividade de uma rede neural Direta.

Onde:

$$E = \begin{bmatrix} e1 \\ e2 \\ \downarrow \\ ej \end{bmatrix}$$

Tabela 2.5 Vetor de Entrada

E

$$M = \begin{bmatrix} snp11 & snpz1 & \dots & snpi1 \\ snp12 & snpz2 & \dots & snpi2 \\ \downarrow & \downarrow & ... & \downarrow \\ snp1j & snpzj & ... & snpij \end{bmatrix}$$

Tabela 2.6 Sinapses

Uma característica desse modelo é que se seus padrões de entrada forem ortogonais, sempre é possível conseguir associações perfeitas.

Apesar desta implementação não estar muito vinculada à inspiração do sistema nervoso, tem a vantagem de facilmente ser concretizada fisicamente, tanto por meio de hardware como por meio de *softwares*.

2.9.2 Redes com Ciclos (Recorrentes)

São redes em que o grafo de conectividade contém pelo menos um ciclo. São também chamadas redes com realimentação ou com *feedback*.

Ao introduzir-se a não-linearidade nas implementações, o desempenho melhora consideravelmente. Um método simples de introduzir a não-linearidade é adotar neurônios com limiares lineares.

Os níveis de ativação são valores binários 0 e 1, obtidos pela comparação da entrada líquida com o limiar de ativação, conforme a função de grau unitário. A função de saída é a função identidade.

Esse modelo insere a capacidade de abstração, possibilitando a aprendizagem de qualquer função, como por exemplo, o ou exclusivo.

Quando os padrões de entrada não são linearmente independentes faz-se necessário a introdução de níveis intermediários. Essa introdução permite que a rede aprenda qualquer função lógica. Todavia, não se conseguiu comprovar formalmente que algum caso existente possa convergir para aprendizagem correta de todos as situações.

2.9.3 Redes Simétricas

São redes cuja matriz (do grafo de conectividade) é simétrica, sendo um caso particular das redes com ciclos. Neste tipo de rede, todas a conexões são nos dois sentidos e são iguais. Um exemplo de utilização são as redes BAM(Bidirectional Associative Memories estudadas por Kosko, que serão apresentadas posteriormente.

2.10 Processo de Aprendizado de uma Rede Neural Artificial

A propriedade mais importante das redes neurais é a habilidade de aprender de seu ambiente e com isso melhorar seu desempenho. Isso é feito através de um processo iterativo de ajustes aplicado a seus pesos, o treinamento. O aprendizado ocorre quando a rede neural atinge uma solução generalizada para uma classe de problemas.

Denomina-se algoritmo de aprendizado a um conjunto de regras bem definidas para a solução de um problema de aprendizado. Existem muitos tipos de algoritmos de aprendizado específicos para determinados modelos de redes neurais, estes algoritmos diferem entre si principalmente pelo modo como os pesos são modificados.

A rede neural se baseia nos dados para extrair um modelo geral. Portanto, a fase de aprendizado deve ser rigorosa e verdadeira, a fim de se evitar modelos espúrios. Todo o conhecimento de uma rede neural está armazenado nas sinapses, ou seja, nos pesos atribuídos às conexões entre os neurônios. De 50 a 90% do total de dados devem ser separados para o treinamento da rede neural, dados estes escolhidos aleatoriamente a fim de que a rede "aprenda" as regras e não "decore" exemplos. O restante dos dados só é apresentado à rede neural na fase de testes a fim de que ela possa "deduzir" corretamente o inter-relacionamento entre os dados.

Outro fator importante é a maneira pela qual uma rede neural se relaciona com o ambiente. Nesse contexto existem os seguintes paradigmas de aprendizado:

- Por independência de quem aprende

As redes neurais Artificiais aprendem por memorização, contato, exemplos, por analogia, por exploração e também por descoberta.

- Por retroação do mundo

Diz respeito à ausência ou presença de realimentação explícita do mundo exterior, ou seja, que em certos intervalos de tempo um agente assinala acertos e erros.

    - Aprendizado Supervisionado: utiliza um agente externo que indica à rede um comportamento bom ou ruim de acordo com o padrão de entrada
    - Aprendizado Não Supervisionado (auto-organização): não utiliza um agente externo indicando a resposta desejada para os padrões de entrada. Utiliza, entretanto, exemplos de coisas semelhantes para que a rede responda de maneira semelhante.

- Por Finalidade do Aprendizado
    - Auto-associador: é apresentada à rede uma coleção de exemplos para que ela memorize. Quando se apresenta um dos elementos da coleção de exemplos, mas de modo errôneo, a rede deve mostrar o exemplo original, funcionando como um filtro.
    - Hetero-Associador: é uma variação do Auto-associador, mas que se memoriza um conjunto de pares. O sistema aprende a reproduzir o segundo elemento do par mesmo que o primeiro esteja pouco

modificado, funcionando, desta maneira, como um reconhecedor de padrões.

É necessário, também, que exista um Detector de Regularidades, que nada mais é que um reconhecedor de padrões em que o sistema deve se auto-organizar e criar padrões possíveis.

Podemos denominar ainda ciclo como sendo uma apresentação de todos os *N* pares (entrada e saída) do conjunto de treinamento no processo de aprendizado. A correção dos pesos num ciclo pode ser executado de dois modos:

- Modo Padrão: A correção dos pesos acontece a cada apresentação à rede de um exemplo do conjunto de treinamento. Cada correção de pesos baseia-se somente no erro do exemplo apresentado naquela iteração. Assim, em cada ciclo ocorrem *N* correções.
- Modo *Batch*: Apenas uma correção é feita por ciclo. Todos os exemplos do conjunto de treinamento são apresentados à rede, seu erro médio é calculado e a partir deste erro fazem-se as correções dos pesos.

2.11 Algoritmos de Aprendizado

As regras ou algoritmos de aprendizagem vêm sendo desenvolvidos para se obter soluções ótimas (mínimos globais de energia), ou soluções razoáveis (mínimos locais de energia), em um tempo de execução relativamente curto.

O modo como a rede aprende é determinado pelo algoritmo usado para mudar suas conexões. A seguir serão apresentados os algoritmos mais conhecidos atualmente e que mostraram alguns resultados com a sua utilização.

2.11.1 Modelos de *Hopfield*

No modelo de Hopfield o momento da ativação do neurônio é aleatório e as conexões são possíveis entre quaisquer dos neurônios, permitindo até realimentações, o que leva uma proximidade maior do sistema nervoso humano.

Hopfield considera que todas as conexões são simétricas, isto é, o peso das sinapses que atuam na transmissão do estímulo do neurônio *i* para o *j* é idêntico ao peso das sinapses que atua na transmissão *j* para *i* (Equação 2.5).

$$Wij = Wji \tag{2.5}$$

Cada possível estado da rede, segundo *Hopfield*, tem associado uma grandeza Energia, cuja superfície é definida pela quantidade desta, e forma um relevo, onde o estado atual da rede caminha, via evolução do processamento, para um mínimo local. Neste estado, existe uma barreira energética que torna o estado estável(Equação 2.6).

$$€ = -\sum Wij \tag{2.6}$$

Onde:

i e *j* são neurônios ativos;

Wij é o peso da sinapse entre os neurônios *i* e *j*

€ é a energia do estado

Uma interpretação da utilização da energia no cálculo do desempenho do sistema pode ser dada por:

- Se existir uma forte sinapse inibidora entre dois neurônios ativos, a energia é positiva e não contribui para o processo de encontrar o mínimo de energia. Este estado é então negligenciado no processo, já que a ativação de uma unidade força a desativação daquela que esteja a ela ligada por uma sinapse inibidora, tornando esse estado instável;
- Se existir uma forte sinapse excitadora entre dois neurônios, o estado contribui com uma energia negativa no processo de convergência da rede.

Esse modelo de rede pode ser empregado para memórias associativas, determinando-se algumas unidades para neurônios de entrada e outras para neurônios de saída. A rede deve ser treinada para formar um relevo que torna os casos que se desejem ensinar em estados estáveis. Estimulando-se os neurônios de entrada com um certo padrão e impedindo os neurônios de entrada de se modificarem, o que é chamado de "*Clamping*". Os outros neurônios deverão atingir a estabilidade, mediante ativação aleatória, exibido padrão desejável nos neurônios de saída.

O modelo de *Hopfield* é utilizado principalmente na solução de problemas de otimização e de roteamento. A solução desse tipo de problema leva, com freqüência, à necessidade de encontrar o mínimo global de uma superfície e um espaço em N dimensional.

O problema mais típico é o problema do caixeiro viajante, de difícil solução computacional. Neste problema, deseja-se determinar a ordem que um caixeiro viajante deve seguir para visitar um grupo de cidades, percorrendo um menor percurso total.

Para resolver este problema como *N* cidades, *Hopfield* propunha usar uma matriz quadrada de *NxN* neurônios ,de modo que as linhas designassem cidades e as colunas definissem a ordem das cidades na lista de visita do caixeiro.

Para que haja uma possível solução, deve-se apenas ter *N* neurônios ativos e não pode haver dois neurônios ativos numa mesma linha ou coluna. Isto é possível montando- se uma função de energia (e que será minimizada) onde estados desse tipo são privilegiados (tem baixa energia).

A função de energia é completada inserindo-se o privilégio para percursos menores apoiada na distância que uma cidade possui em relação a outra, ou seja, existe um componente que resguarda os pontos de estabilidade para que sejam estados estáveis e outra que implementa a função de análise de desempenho (no caso, o percurso menor).

Este problema pode ser resolvido fisicamente através de um circuito eletrônico, em que a função de energia desejada é comparada com a função de energia natural do circuito, ou através da simulação do circuito em um computador convencional, mediante o uso de suas equações de movimentos resultantes da aplicação das regras de *Kirchoff* em determinados pontos.

A rede neural, para resolver este problema, deve ser de tal modo que:

- Unidades de uma mesma fila (Linha ou coluna) forçam–se à inibição de tal maneira que ativação de uma leve e à inibição das demais;
- Unidades que não estejam em uma mesma fila devem excitar-se, tanto mais quanto mais próximas estiverem das cidades a que se referem.

2.11.2 Regra de Aprendizado de *Hebb*

Descrito aqui, temos uma sugestão simples de uma teoria que responde à pergunta: “Como nós aprendemos?”.

A base desta teoria data do ano de 1949 do livro "*Organization of Behavior*", escrito por *Hebb*. A idéia central estava na seguinte afirmação:

"Quando um axônio de uma célula A está próxima o suficiente de excitar uma célula B e repetidamente ou persistentemente toma parte em ativá-la, algum processo crescente ou mudança metabólica se apossa de uma ou ambas as células de forma que a eficiência de A, assim como a de uma das células B excitadas, são aumentadas".

Assim como o modelo de *McCulloch-Pitts*, esta lei de aprendizagem não explica tudo sobre este tema, porém, de uma forma ou de outra, ela está presente em muitos modelos de redes neurais que conhecemos hoje. Apesar de ser uma regra muito simples, destinada apenas à associações de padrões.

A idéia desta regra foi conceituada da seguinte forma: "Ajuste o valor da sinapse de acordo com o produto de suas ativações simultâneas".

Isto quer dizer que:

- Se duas unidades estiverem ativas no mesmo estado de ativação, incrementa- se a sinapse;
- Se duas unidades estiverem em estados diferentes (uma ativa e outra inativa) decrementa-se o valor da sinapse.

Essa formulação pode ser descrita do seguinte modo(Equação2.7) :

$$\Delta Wij = \eta.Aj.Ti \qquad (2.7)$$

Onde:

Wij é a variação da sinapse *Wij* que conecta a unidade de entrada i a saída j

Aj é a ativação da unidade de entrada *j*

Ti é a ativação desejada para a unidade de saída *i*

η é o fator que determinará a magnitude da variação (taxa de aprendizagem

2.11.3 *Perceptron*

Este algoritmo representa um grande avanço em relação à regra anterior, devido principalmente ao teorema da convergência dos *perceptrons* . Esse teorema garante que se um conjunto de padrões for "aprendível", o processo de aprendizagem o encontrará.

O teorema determina que:

- Caso a unidade *j* responda 0 quando deveria responder 1, aumenta-se os pesos de todos as unidades ativas ligadas à sinapse por uma quantidade η (taxa de aprendizagem livre);
- Caso a unidade *j* responda 1 quando deveria responder 0, decrementa-se os pesos de todas as sinapses que ligam as unidades ativas por η

2.11.3.1 *Perceptron* Multi Camadadas (MLP)

A forma de arranjar perceptrons em camadas é denominado *Multilayer Perceptron*(MLP). O *multilayer perceptron* foi concebido para resolver problemas mais complexos, os quais não poderiam ser resolvidos pelo modelo de neurônio básico. Um único perceptron ou uma combinação das saídas de alguns perceptrons poderia realizar uma operação XOR, porém, seria incapaz de aprendê-la. Para isto são necessárias mais conexões, as quais só existem em uma rede de perceptrons dispostos em camadas. Os neurônios internos são de suma importância na rede neural, pois se provou que sem estes se torna impossível a resolução de problemas linearmente não separáveis.
Usualmente as camadas são classificadas em três grupos:

- Camada de Entrada: onde os padrões são apresentados à rede;
- Camadas Intermediárias ou Ocultas: onde é feita a maior parte do processamento, através das conexões ponderadas; podem ser consideradas como extratoras de características;

- Camada de Saída: onde o resultado final é concluído e apresentado.

2.11.4 Regra Delta (Regra *Widrow-Holff*)

Tradicionalmente, usa-se esta regra em redes de dois níveis com unidade de limiar linear.

Se os padrões de entrada forem ortogonais, a regra determina associações corretas, senão, o algoritmo alcança uma solução intermediária do estilo da linearização por mínimos quadrados. O nome "Regra Delta" vem da utilização de deltas, diferenças, erros.

A regra prevê que o peso da sinapse que interliga neurônio de entrada *i* para outro de saída *j* é alterado por um valor proporcional à diferença entre o valor esperado na saída e o valor real da ativação, onde a constante de proporcionalidade é a taxa de aprendizagem livre η, conforme a Equação 2.8 .

$$\Delta Wij = \eta \, . \, (tpj - \alpha pj) \, . \, \alpha pi \qquad (2.8)$$

Onde:

tpj é a saída esperada para o padrão p na unidade j

αpj é a saída real para o padrão p na unidade j

αpi é a saída real para o padrão p na unidade i

η é a unidade de aprendizagem livre do sistema.

Matematicamente, algumas observações podem ser feitas:

- As sinapses de todas as entradas que estiverem inativas e as sinapses das unidades de saída que estiverem respondendo corretamente não serão modificadas;
- Quanto maior o erro em uma unidade de saída, maior será a variação que tentará impor em suas sinapses para que este erro diminua;
- Se a unidade de saída responder mais que o desejado, a variação será no sentido de diminuir os pesos das sinapses, e assim diminuir a entrada líquida desta unidade;

- Se a unidade de saída responder menos que o desejado, a variação será no sentido de aumentar os pesos das sinapses e aumentar, deste modo, a entrada líquida desta unidade.

2.11.4.1 Regra Delta Generalizada

A Regra Delta generalizada é também denominada b*ackpropagation* ou Propagação Retroativa de Erros.

A regra Delta Generalizada, como diz o próprio nome, é uma generalização da Regra Delta para redes neurais com unidades escondidas. Esta regra criou um modo de ajustar as sinapses que conduzem as unidades escondidas (ou níveis intermediários).

O algoritmo explicita a propagação de um padrão através da rede neural, nível a nível, da entrada para a saída, calculando a resposta de cada neurônio e calculando os deltas nas unidades intermediárias (nível a nível no sentido oposto).

As alterações correspondentes aos pesos das sinapses são feitas após o cálculo de todos os deltas.

Esta é talvez a mais popular regra de aprendizado. A maioria dos programas para tratar redes neurais artificiais dispõe de uma implementação de *backpropagation* ou na forma original (usando gradiente) ou em uma forma modificada para melhorar a performance da regra. Além de ser a primeira regra inventada para efetuar treinamento supervisionado de redes diretas com mais de duas camadas e conseqüentemente não limitada a resolver problemas linearmente separáveis, tem ainda a vantagem de que, se a função de ativação for uma função analítica derivável, a derivada pode ser calculada explicitamente, evitando todos os problemas inerentes à derivação numérica.

Como na *backpropagation* a correção a cada passo é proporcional à derivada, os pesos mudam mais para as unidades que têm uma ativação de valor intermediário, o que contribui para a convergência do algoritmo.

Convém ainda notar que a retropropagação pode ser usada (teoricamente) para neurônios onde as entradas são combinadas usando multiplicação, para redes com realimentação ou recorrentes. Finalmente cabe salientar que nem toda rede direta pode ser treinada pela retropropagação, pois devido ao cálculo da derivada é necessário que a função de ativação seja derivável. Por outro lado, é possível treinar redes com ciclos por

retropropagação. Portanto, é impreciso chamar redes diretas de redes de retropropagação como se faz freqüentemente na literatura.

2.11.5 Adaline(Adaptative Liner Network)

O Adaline é um padrão de classificação adaptativo que utiliza a regra delta para a sua operação.

O Adaline consiste em duas partes:

- *Adaptative Linear Combiner* (ALC): é o local aonde as entradas são multiplicadas pelos pesos a elas associados e somados, gerando um valor analógico (entrada Liquida)
- Função de saída: É a função de saída discreta, que resulta em +1 se a saída do *ALC* for positiva ou igual a 0 , e –1 se for negativa.

O Adaline é adaptativo no sentido que existe um procedimento para modificação dos pesos bem definido, que permite ao algoritmo fornecer os valores de saída corretos para as entradas que foram dadas. E ele é linear porque a saída é uma simples função linear dos valores de entrada.

Madaline

Uma das primeiras redes neurais multicamadas com elementos adaptativos treináveis foi a Madaline.Seu nome vem de “Many Adalines” e ela surgiu a necessidade de uma estrutura com vários níveis , já que o Adaline (Como qualquer outro método que utilize uma função discreta) não pode executar a função ou Exclusivo(XOR).Mas com uma Madaline composta por três Adalines em duas camadas, este problema está resolvido.

Os sinais de entrada da Madaline fluem em um só sentido. O estado de um neurônio só influencia os elementos das camadas seguintes(Feed Forward).

2.11.6 BAM (Bidirectional Associative Memory)

O conceito de memória associativa é quase intuitivo: aparenta ser uma das funções primárias do cérebro. Facilmente associamos o nome de uma pessoa com o seu rosto, ou cheiro de uma comida com o seu aspecto em si.

O modelo BAM consiste em duas camadas de neurônios que são conectados bidirecionalmente, ou seja, o fluxo de informações passa da camada A para B e vice versa.

Assim como em outros modelos de redes neurais, nba BAM há pesos associados com as conexões entre os neurônios.

Neste tipo de rede, não há necessariamente um nível de entrada e um nível de saída, podendo ser qualquer um dos dois. Fornecendo uma entrada na primeira camada, os sinais vão se propagar para a segunda camada e novamente para a primeira, já com novos sinais. Isto se repetirá até que não ocorra qualquer alteração nos sinais.

Seu treinamento é feito de maneira simples, construindo uma matriz de pesos quadrada e simétrica. A desvantagem é que o número de padrões e valores são limitados.

2.11.7 Modelos Probabilísticos

Apesar de encontrar-se boas soluções quando atinge um mínimo local de energia, em um problema de otimização, essas não apresentam a melhor solução.

Os modelos probabilísticos surgiram com a finalidade de se encontrar uma solução ótima. Tais modelos não garantem o encontro da melhor solução, mas controlam as probabilidades de atingir cada mínimo, a fim de tornar o mínimo em absoluto mais freqüente.

Existem vários modelos e algoritmos probabilísticos, como a Teoria da Harmonia, de *Paul Smolensky* e a máquina de *Boltzmann* de *G. E. Hilton* e *T. j. Sejnowsky*, aprofundaremo-nos um pouco mais neste último exemplo por ser mais usual.

A máquina de *Boltzmann* é uma combinação de idéias divididas de diversos campos, incluindo: otimização combinatorial, teoria da informação, modelagem de

redes neurais, visão de máquina, computação paralela e modelamento de memória associativa.

Essa modelagem assegura que a probabilidade relativa de se atingir 2 mínimos diferentes no final do processamento, depende da diferença da energia entre eles, sem considerar a seqüência de atuação dos neurônios.

O modelo da máquina de *Boltzmann* usa a função de distribuição *Boltzmann* (ou função da probabilidade termodinâmica) para ajustar a energia das conexões.

Definindo o valor de um estado por esta função, a probabilidade de uma mudança de estado é muito pequena quando a rede atinge uma solução ótima, mas alta quando a rede está longe de uma solução. Esta alteração dos estados é realizada pela aplicação de uma técnica chamada Algoritmo *Metrópolis*.

A troca introduz uma variável *T* que funciona como temperatura, de modo que para cada temperatura a probabilidade global de 2 possíveis padrões de entradas *A* e *B* alcance um balanço ou equilíbrio térmico definido pela Equação 2.14:

$$PROBa / PROBb = e^{-(Ea - Eb) / t} \qquad (2.14)$$

Onde:

E é a soma da energia para cada estado de dois possíveis padrões permutados *A* e *B* e *T* é a variável atuando como temperatura.

Essa distribuição de probabilidade muda o estado como uma função da temperatura, onde em altas temperaturas a rede efetua uma pesquisa grosseira sobre a matriz de estados (que rapidamente converge globalmente), e então a temperatura refrigera, que é incrementalmente feita por um mínimo local próximo de uma região de mínimo global.

Para aplicar esse modelo de rede, o padrão é manipulado pelo cálculo da mudança de uma quantidade de energia que resultaria em um afastamento rápido de cada estado dentro de um valor alternativo e indiferente de seu estado atual, mudando o estado para 1 com probabilidade(Equação 2.15):

$$P1 = 1 / 1 + e^{-\Delta E/T} \qquad (2.15)$$

Senão muda para valor de estado 0

O processo é repetido muitas vezes diminuindo progressivamente a temperatura até que ela se torne muito baixa e o estado se fixa dentro de um padrão do mínimo global. Essa técnica tem se mostrado muito útil em partição gráfica, e muito utilizado para projetar placas de circuitos em larga escala.No lado negativo, entretanto, ela precisa de muitos ensaios, e é mais lenta que o modelo *Hopfield*.

Uma extensão interessante da máquina de *Boltzmann* é a imitação da aprendizagem autônoma.

Um problema difícil de recente pesquisa em redes neurais foi chamado problema de atribuição de crédito. Para formar com computações não triviais, uma rede deve conter estados que não são diretamente coagidos para a entrada. Entretanto quando a rede se constitui corretamente, ela está impedida de determinar qual dos muitos valores da conexão da matriz de estados causou o problema (não possui lógica para alterar suas análises).

Uma modificação da máquina de *Boltzmann* pode solucionar este problema. O método leva em consideração o simples relacionamento entre o valor do peso na matriz de estados e a probabilidade relativa de pares de estados para encontrar o equilíbrio térmico. Se as probabilidades de pares de estados forem calculadas quando alguns estados são capturados pelo ambiente e alguns estados intermediários não são (*Pcapturados*) e compararmos a probabilidade dos mesmos estados quando não capturados (*Pnãocapturado*), o peso da matriz de estados entre os dois estados é modificado assim que mudar(Equação 2.16):

$$Wij = c.(Pcapturado - Pnãocapturado) \quad (2.16)$$

Onde:

c é uma porcentagem pequena.

Usando este método, a máquina de *Boltzmann* pode aprender a detectar irregularidades nos padrões influenciados pelo ambiente e gradualmente ajustar seus pesos internos até que ela possa recriar a mesma regularidade durante a reconstituição.

Este método é uma promessa para sistemas que podem aprender a respeito de novos ambientes sem intervenção ou interpretação humana.

2.11.7.1 Máquina de *Boltzmann*

A computação das máquinas de *Boltzmann* pode ser realizada da seguinte maneira:

Fixam-se as unidades de entrada com o padrão de interesse e deixam-se as unidades restantes atuarem, probabilisticamente, ativando-as e desativando-as até que a rede atinja o equilíbrio térmico.

Neste ponto, as unidades de saída ainda estão mudando de estados, mas as probabilidades dos padrões de saída tornaram-se invariantes com a continuação do processo.

Se reduzirmos a temperatura continuamente, forçando equilíbrio térmico em temperatura baixa, teremos padrões de saída muito prováveis. Quando a temperatura for nula, atingimos um estado provável, pois a ativação de cada neurônio será regida por uma função de grau, que dará a probabilidade 0 ou 1 do neurônio estar ativo, definindo, então o seu estado.

De forma simplificada, podemos aplicar o algoritmo de aprendizagem seguindo os seguintes passos:

1) Fixar T, η, Tol

Onde:

T é a Temperatura

η é a Taxa de Aprendizagem Livre

Tol é o tempo de tolerância para a rede aprender

2) Inicializar a matriz de conectividade W com valores aleatórios pequenos, de dimensão i X j onde i é o comprimento do padrão de entrada e j o de saída.

3) Obter os pares dos vetores de entrada e de saída.

4) Treinar a rede até atingir a convergência, dentro da tolerância especificada(Equações 2.17,2.18,2.19,2.20).

$$NetPj = \sum Wij * Inpi \quad (2.17)$$

$$Netpj = f\ limiar\ (Netpj) \quad (2.18)$$

$$\delta pj = \eta * (Outpj - Netpj) * Inpi \quad (2.19)$$

$$Wij = Wij + \delta pj \quad (2.20)$$

Onde:

Netpj é a saída real do padrão *p*

Inpi é o padrão de entrada

Outpj é o padrão de saída desejado

f limiar é a função de ativação

Wij é o peso da sinapse que interliga a unidade i do nível de entrada à unidade *j* do nível de saída.

A máquina de *Boltzmann* possui duas características importantes:

- Pode conter níveis intermediários (e o treinamento para eles repete-se da mesma forma)
- Possui a capacidade de Abstrair dados.

2.11.8 Aprendizagem por Competição

No aprendizado competitivo, usado nas redes popularizadas por *Kohonen*, neurônios são inibidos por outros neurônios de modo a que a competição entre eles leva a apenas um acabar excitado. Assim, enquanto uma rede neural baseada em um aprendizado *Hebbian*, vários neurônios de saída podem estar simultaneamente ativos, no caso do aprendizado competitivo, somente um neurônio de saída fica ativo de cada vez.

No aprendizado competitivo,as entradas possuindo alguma semelhança, tendem a excitar o mesmo neurônio na saída. Assim é que este paradigma de aprendizado pode servir para sugerir classificações.

A aprendizagem por competição usa redes estruturadas, com as unidades divididas em níveis, e que podem estar ativas ou inativas.

Os neurônios recebem estímulos de todos os neurônios do nível anterior e envia estímulos a todos os neurônios do nível posterior.

As unidades de um mesmo nível se ligam por sinapses inibidoras e as de níveis diferentes por sinapses excitadoras. Em cada nível, somente uma unidade de cada grupo pode estar ativa no tempo, inibindo as outras.

As unidades ativas de cada nível formam o padrão de entrada do próximo nível.

Os grupos são responsáveis pela detecção das características, onde cada unidade representa um estado dessa característica.

Uma vez iniciada a competição, a unidade vencedora adquire o direito de atingir o máximo valor de ativação(1) e levar todas as outras de seu grupo para o nível mínimo(0).

A unidade, dentro de cada grupo, que venceu a competição, tem o direito de alterar o valor de sua sinapse.

A variação do peso da sinapse excitadora entre as unidades de níveis vizinhos é dada pela fórmula na equação 2.21 e 2.22:

$\Delta Wij = 0$ se a unidade j perder no estímulo k ou (2.21)

$g * Cik / Nk - g * Wij$ se a unidade j ganhar no estímulo k (2.22)

Onde:

Cik é igual a 1 se a unidade i estiver ativa no padrão de entrada no nível N, e caso contrário é 0.

Nk é igual ao número de unidades ativas no padrão de entrada, ou seja, $Nk = \sum Cik$

g é um parâmetro que atua com taxa de aprendizagem livre do sistema.

Podemos observar alguns resultados interessantes deste algoritmo:

- Cada grupo de elementos classificam os estímulos em estados de mesmo número de elementos. A distribuição de estados é de tal forma uniforme, que uma unidade não fica muito mais ativada do que a outra;
- Se os estímulos de entrada possuem características bem definidas, o sistema encontrará esses grupos;
- Se os estímulos forem muito estruturados, as classificações serão bastante estáveis.

2.11.8.1 Mapas de *Kohonen*

O mapa de *Kohonen* é um algoritmo capaz de realizar auto organização, a rede aprende sem conhecer, mesmo na fase de treinamento, as respostas aos estímulos a que se dispõe.

Para aprender sem supervisão externa, uma rede deve ter a capacidade de se auto organizar de modo a refletir, dentro de si, as irregularidades existentes nos estímulos neurais.

Esse algoritmo é uma descoberta de *Tuevo Kohonen*, da universidade tecnológica de *Helsinque* na Finlândia.

As redes utilizadas por *Kohonen* precisam apenas de um nível. Os neurônios são altamente conectados ao ambiente externo. Não existe um esquema de estímulos propagados de nível a nível. Apesar desse mecanismo parecer simples, exige as seguintes condições:

- Os pesos das sinapses para cada neurônio devem ser inicializados de modo a produzir vetores (pensando que cada peso fosse uma coordenada) aleatoriamente direcionados dentro do círculo unitário;
- Deve-se utilizar entradas normalizadas para um valor constante, o usualmente 1.

A rede neural proposta por *Kohonen* funciona da seguinte forma:

Quando a rede recebe um estímulo do mundo exterior, todas as unidades competem para ver qual unidade responde com mais força. Se todas as unidades tiverem sido inicializadas da mesma forma, todas responderão da mesma forma.

A unidade que respondeu mais forte ganha direito de se tornar mais especializada em responder o mesmo estímulo e estende esse privilégio já sua vizinhança. Deste modo, essa região terá mais facilidade em responder a estímulos da mesma natureza em casos futuros.

À medida que o tratamento evolui, a vizinhança vai se tornando menor de modo a dar maior especialidade a cada unidade. O percentual de transformação dos pesos das sinapses também vai decaindo até atingir zero (quando a fase de treinamento está terminada), como pode ser observado na Tabela 2.10.

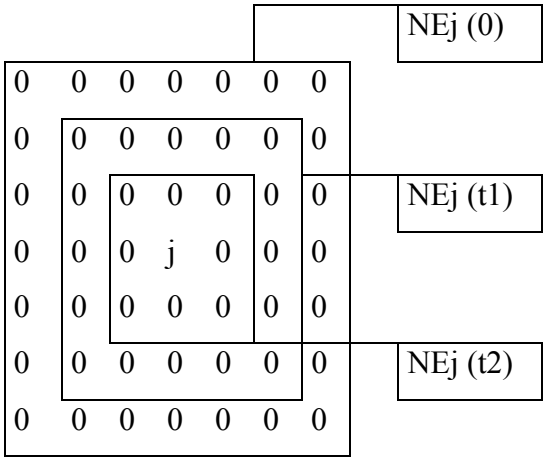

Tabela 2.7: Definição da vizinhança *Nej(t) (0< t1 < t2)*

A normalização dos estímulos de entrada é necessária para não fazer com que a energia de um estímulo afete na qualidade de sua absorção pela rede

De modo geral poderíamos expressar a formação dos mapas de *Kohonen* da seguinte forma:

1. Inicialização dos pesos que ligam as unidades da rede às unidades de entrada (captadores).
2. Apresentar estímulos.
3. Computar a distância (a diferença do estímulo para todos os neurônios).
4. Selecionar o neurônio com menor distância.
5. Atualizar pesos do módulo vencedor e seus vizinhos.
6. Voltar ao passo 2.

Este algoritmo está sendo muito usado para aplicações relacionadas ao reconhecimento da fala.

2.11.8.2 ART(Adaptive Resonance Theory).

A Teoria da Adaptação Ressonante (Adaptive Resonance Theory) resultou num classificador projetado por Carpenter e Grossberg que possui entrada binária, conexão retro-alimentada e aprendizado não-supervisionado.

A idéia de armazenamento dos padrões tem semelhanças com o modelo de Kohonen, uma vez que há a distinção de zonas, a comparação entre as distâncias vetoriais das somas ponderadas, e a escolha de um neurônio vencedor.

No modelo ART existe a retro-propagação dos sinais, o que permite-lhe a seleção dos neurônios que melhor representariam um dado padrão de entrada. A interconexão entre os dois nodos de saída permite a inibição lateral entre eles, visando a definição do valor máximo de saída.

O aprendizado do modelo ART é bem distinto dos outros modelos, apenas com algumas semelhanças com o de Hopfield e o de Kohonen. Assemelha-se ao de Kohonen em sua forma de distribuição dos pesos na rede através da seleção de um neurônio vencedor, com a diferença de que no ART apenas o vencedor é atualizado, não havendo a atualização da vizinhança, que caracteriza o aprendizado competitivo. Quanto à semelhança ao Hopfield, nota-se que o padrão a aprender confunde-se com o a reconhecer, uma vez que há iterações para ambos.

O modelo ART possui a facilidade de não ter necessidade de um aprendizado prévio, podendo adaptar-se de acordo com a necessidade. Por outro lado, este modelo é muito sensível a padrões distorcidos, não podendo fazer uma boa diferenciação entre padrões semelhantes, limitando, assim, em muito suas aplicações.

Utiliza-se o ART para reconhecimento de imagens, mais comumente para reconhecimento de caracteres.

2.12.9 Aprendizado Reforçado

O Aprendizado Reforçado ou "*Reinforcement Learning*" consiste no aprendizado através do método da tentativa e erro, de modo a otimizar um índice de performance chamado sinal de reforço.

Este paradigma de aprendizado tem profunda motivação biológica, em que comportamentos que provocam satisfação têm como conseqüência um reforço das conexões que os produziram, e aqueles que provocam insatisfação tem uma modificação do valor das correspondentes conexões.

2.11.10 Aprendizado Aleatório

O aprendizado é dito aleatório quando os passos no sentido de obter o comportamento aprendido se baseiam em valores tomados aleatoriamente, que são testados para verificar sua adequabilidade. Assim, em essência o aprendizado aleatório segue os seguintes passos:

- Selecione os valores das conexões sinápticas de modo aleatório;
- Verifique o valor da performance da rede;
- Provoque uma variação aleatória nas conexões sinápticas e verifique o novo valor da performance da rede. Se melhorar, retenha este novo valor de conexões. Caso contrário escolha um critério para escolher nova variação;
- Verifique se um critério de parada especificado inicialmente foi satisfeito e neste caso, páre o aprendizado.

De uma certa forma, o aprendizado aleatório coincide com o aprendizado com a natureza.

2.11.11 Aprendizado Evolutivo

Aprendizado evolutivo é o paradigma de aprendizado que, tirando inspiração da evolução biológica, é capaz de modificar a topologia e os valores das conexões sinápticas de modo a fazer uma rede se tornar apta a resolver um problema. Este assunto pode servir tanto como algoritmo de aprendizado como para determinar a topologia da rede a ser usada para resolver determinado problema.

2.12 Desenvolvimento de Aplicações utilizando Redes Neurais Artificiais

Os seguintes passos são necessários para o desenvolvimento de aplicações utilizando redes neurais artificiais:

2.12.1 Coleta de dados e Separação em Conjuntos

Os dois primeiros passos do processo de desenvolvimento de redes neurais artificiais são a coleta de dados relativos ao problema e a sua separação em um conjunto de treinamento e um conjunto de testes. Esta tarefa requer uma análise cuidadosa sobre o problema para minimizar ambigüidades e erros nos dados. Além disso, os dados coletados devem ser significativos e cobrir amplamente o domínio do problema; não devem cobrir apenas as operações normais ou rotineiras, mas também as exceções, e as condições nos limites do domínio do problema.Normalmente, os dados coletados são separados em duas categorias:

- Dados de treinamento, que serão utilizados para o treinamento da rede.
- Dados de teste, que serão utilizados para verificar sua performance sob condições reais de utilização.

Além dessa divisão, pode-se usar também uma subdivisão do conjunto de treinamento, criando um conjunto de validação, utilizado para verificar a eficiência da rede quanto a sua capacidade de generalização durante o treinamento, e podendo ser empregado como critério de parada do treinamento.

Depois de determinados estes conjuntos, eles são geralmente colocados em ordem aleatória para prevenção de tendências associadas à ordem de apresentação dos dados.

Além disso, pode ser necessário pré-processar estes dados, através de normalizações, escalonamentos e conversões de formato para torná-los mais apropriados à sua utilização na rede.

2.12.2. Configuração da rede

O terceiro passo é a definição da configuração da rede, que pode ser dividido em três etapas:

- Seleção do paradigma neural apropriado à aplicação.
- Determinação da topologia da rede a ser utilizada - o número de camadas, o número de unidades em cada camada, etc.
- Determinação de parâmetros do algoritmo de treinamento e funções de ativação. Este passo tem um grande impacto na performance do sistema resultante.

Existem metodologias, "dicas" e "truques" na condução destas tarefas. Normalmente estas escolhas são feitas de forma empírica. A definição da configuração de redes neurais é ainda considerada uma arte, que requer grande experiência dos projetistas.

2.12.3 Treinamento

O quarto passo é o treinamento da rede. Nesta fase, seguindo o algoritmo de treinamento escolhido, serão ajustados os pesos das conexões. É importante considerar, nesta fase, alguns aspectos tais como a inicialização da rede, o modo de treinamento e o tempo de treinamento.

Uma boa escolha dos valores iniciais dos pesos da rede pode diminuir o tempo necessário para o treinamento. Normalmente, os valores iniciais dos pesos da rede são números aleatórios uniformemente distribuídos, em um intervalo definido. A escolha errada destes pesos pode levar a uma saturação prematura. Quanto ao modo de treinamento, na prática é mais utilizado o modo padrão devido ao menor armazenamento de dados, além de ser menos suscetível ao problema de mínimos locais, devido à pesquisa de natureza estocástica que realiza. Por outro lado, no modo batch se tem uma melhor estimativa do vetor gradiente, o que torna o treinamento mais estável.

A eficiência relativa dos dois modos de treinamento depende do problema que está sendo tratado.

Quanto ao tempo de treinamento, vários fatores podem influenciar a sua duração, porém sempre será necessário utilizar algum critério de parada. O critério de parada do algoritmo backpropagation não é bem definido, e geralmente é utilizado um número máximo de ciclos, mas devem ser consideradas a taxa de erro médio por ciclo, e a capacidade de generalização da rede. Pode ocorrer que em um determinado instante do treinamento a generalização comece a degenerar, causando o problema de over-training, ou seja, a rede se especializa no conjunto de dados do treinamento e perde a capacidade de generalização.

O treinamento deve ser interrompido quando a rede apresentar uma boa capacidade de generalização e quando a taxa de erro for suficientemente pequena, ou seja, menor que um erro admissível. Assim, deve-se encontrar um ponto ótimo de parada com erro mínimo e capacidade de generalização máxima.

2.12.4 Teste

O quinto passo é o teste da rede. Durante esta fase o conjunto de teste é utilizado para determinar a performance da rede com dados que não foram previamente utilizados. A performance da rede, medida nesta fase, é uma boa indicação de sua performance real.

Devem ser considerados ainda outros testes como análise do comportamento da rede utilizando entradas especiais e análise dos pesos atuais da rede, pois se existirem valores muito pequenos, as conexões associadas podem ser consideradas insignificantes e assim serem eliminadas (*Prunning*). De modo inverso, valores substantivamente maiores que os outros poderiam indicar que houve over-training da rede.

2.12.5 Integração

Finalmente, com a rede treinada e avaliada, ela pode ser integrada em um sistema do ambiente operacional da aplicação. Para maior eficiência da solução, este sistema deverá conter facilidades de utilização como interface conveniente e facilidades de aquisição de dados através de planilhas eletrônicas, interfaces com unidades de

processamento de sinais, ou arquivos padronizados. Uma boa documentação do sistema e o treinamento de usuários são necessários para o sucesso do mesmo.

Além disso, o sistema deve periodicamente monitorar sua performance e fazer a manutenção da rede quando for necessário ou indicar aos projetistas a necessidade de treinar novamente a rede. Outras melhorias poderão ainda ser sugeridas quando os usuários forem se tornando mais familiares com o sistema. Estas sugestões poderão ser muito úteis em novas versões ou em novos produtos.

Capítulo 3

3.1 Inteligência Artificial Distribuída (IAD):

A Inteligência Artificial Distribuída pode ser vista como sendo um sub-campo da Inteligência Artificial voltada para a construção de agentes inteligentes, que são as entidades dentro de um sistema inteligente distribuído. Essas entidades são autônomas e podem cooperar entre si através de algum mecanismo de natureza inteligente.

A Inteligência Artificial Distribuída propõe estratégias para a solução de problemas que se caracterizam por serem solucionados através da interoperabilidade de sistemas ou entidades, geralmente heterogêneos, remotamente distribuídos numa rede de computadores.

3.1.1 Por que utilizar IAD?

Existem muitas razões para que seja empregado o conceito de Inteligência Artificial Distribuída. Entre elas podemos ressaltar as que seguem:

- Mais poder de computação ou hardware mais barato: Esta é uma das principais razões para se aplicar as funcionalidades distribuídas em um sistema: o alto custo do hardware necessário para resolver problema de forma centralizada, que nem sempre é rápido o suficiente para tal tarefa.
- Maior segurança e tolerância a falhas: Se não se pode confiar a resolução de um determinado problema a um único processo, nada mais justo que encaminhar este problema a um grupo de *experts*. Tendo vários processos especialistas, quando um problema surge, o problema pode ser colocado em evidência, sendo analisado por vários sistemas, paralelamente. Como conseqüência disso, temos várias opiniões, e não apenas uma, sobre qual atitude deve ser tomada.
- Aproveitamento da Tecnologia existente: Utilizando-se da IAD, pode-se resolver problemas que não seriam possíveis resolver com a tecnologia já desenvolvida. Com a IAD a base de conhecimentos pode ser subdividida em áreas, assim como o problema. A IAD estimula decomposição do problema total de gerenciamento, tornando a solução o somatório de módulos (sistemas).

3.1.2 Benefícios da IAD:

Existem alguns benefícios para a utilização da IAD nas soluções de problemas distribuídos considerando os seguintes critérios: (PARAISO, 1997) (APUD BOND E GESSER, 1988)

- Adaptabilidade: Sistemas de IAD são mais apropriados para lidar com problemas distribuídos em termos espaciais, lógicos, temporais ou semânticos;
- Custo: Um grande número de pequenas unidades computacionais pode ser mais interessante, quando os custos em comunicação não são relevantes;
- Desenvolvimento e Gerenciamento: A inerente modularidade do sistema em IAD permite o desenvolvimento das partes do mesmo de forma separada e paralela;
- Eficiência e Velocidade: A concorrência e a distribuição dos processos em diferentes máquinas pode aumentar a velocidade de processamento haja vista o paralelismo na execução das tarefas.
- História: A integração de recursos distribuídos e até mesmo heterogêneos tais como hardware e software de diferentes plataformas;
- Isolamento/Autonomia: O controle de processos locais pode ser encarado como uma maneira de proteção ou de aumento da segurança do sistema;
- Naturalidade: Alguns problemas são naturalmente melhor resolvidos através de uma configuração distribuída, como é o caso da gerência de redes de computadores;
- Limitações de Recursos: Os agentes computacionais individuais ligados a recursos escassos podem, através da cooperação, superar problemas complexos;
- Especialização: Pode-se especializar cada agente de acordo com seu domínio de conhecimento e aplicação.

Observando cada critério acima e a respectiva vantagem do uso da IAD, é notório que conforme ocorra a evolução das redes de computadores em termos de confiabilidade, robustez e usabilidade, a busca de soluções para os problemas distribuídos se torne cada vez mais comum. É nesse contexto que se sugere mecanismos para resolução adequada desses problemas.

3.1.3 Divisão e Classificação da IAD:

A IAD pode ser dividida em duas áreas maiores: Resolução Distribuída de Problemas(DPS) e Sistemas Multi-agentes(MAS).

Na resolução distribuída de problemas, os agentes cooperam uns com os outros, dividindo e compartilhando conhecimento sobre o problema e sobre o processo de obter uma solução. Nesta abordagem, os agentes são projetados especificamente para resolver aquele problema ou classe de problemas. Sob um ponto de vista externo, um sistema DPS é visto como uma unidade. O processo de coordenação das ações dos agentes é definido em tempo do projeto (OLIVEIRA, 1996).

Em Sistemas multi-agentes, o projetista não volta sua atenção para um problema específico, mas para um domínio específico. Nesta abordagem, a idéia consiste em coordenar o comportamento inteligente de um conjunto de agentes autônomos, cuja existência pode ser anterior ao surgimento de um problema em particular. Os agentes devem raciocinar a respeito das ações e sobre o processo de coordenação em si. As suas arquiteturas são mais flexíveis e a organização do sistema está sujeita a mudanças visando adaptar-se às variações no ambiente e/ou no problema a ser resolvido (OLIVEIRA, 1996).

3.2 Agentes

Agente é uma entidade cognitiva, ativa e autônoma, ou seja, que possui um sistema interno de tomada de decisões, agindo sobre o mundo e sobre os outros agentes que o rodeiam, e, por fim, que é capaz de funcionar sem necessitar de algo ou de alguém para o guiar (tem mecanismos próprios de percepção do exterior).

Uma Analogia pode ser feita com agentes no mundo real levando a caracterizar um agente como uma entidade ativa, que possui conhecimento específico sobre um determinado domínio. De posse de bases de conhecimento e de mecanismos de raciocínio, os agentes autonômos devem ser capazes de reconhecer situações em que eles poderiam atuar, de forma transparente ao usuário.

Em (LUCK,1997) é proposta uma hierarquia de quarta ordem, que envolve as seguintes entidades:

- Entidade: a idéia básica é de que todos os componentes do mundo são entidades. Estas seriam um mecanismo de abstração muito útil, que permite diferenciá-las das demais entidades do ambiente.
- Objeto: é uma entidade que possui atributos e métodos.
- Agentes: é uma instância de um objeto que possui um propósito ou um conjunto de propósitos.
- Agentes Autônomos: Podem ser conceituados como sistemas dotados da capacidade de interação independente e efetiva com o ambiente onde se encontra, regida pelo objetivo de concluir alguma tarefa.

Andrew Wood (WOOD, 1994) sugere algumas definições informais para o desenvolvimento tais como: agente humano, agente inteligente, ferramenta de software ou aplicação, comunicação ou interação, observação e interface. A seguir serão apresentadas tais definições:

- Agente Humano: qualquer pessoa que interage direta ou indiretamente com um sistema de computador (geralmente chamada de usuário ou usuário final);
- Agente Inteligente: uma entidade que se comporta como um agente.
- Ferramenta de *Software* ou Aplicação: qualquer software que não seja um agente. Inclui os softwares normalmente utilizados no processamento de informações, como editores de texto, planilhas e agendas eletrônicas, programas de desenho e editoração, etc.
- Comunicação ou Interação: passagem de controle ou informações que podem ser transmitidas entre as entidades.
- Observação: quando uma entidade está obtendo informações sobre o estado de outra entidade.
- Interface: o ponto pelo qual duas entidades comunicam-se ou interagem. Com relação a interação homem-máquina, esta é tradicionalmente conhecida como a interface do usuário

3.3 Caracterizando um Agente

O conceito agente pode englobar muitos outros conceitos, funções e exemplos. Desta forma, como o consenso em termos do que pode ser definido como Agente é incerto, o leque de opções para se definir uma determinada instância da realidade como Agente é de amplitude quase infinita. É também uma completa utopia buscar possibilidades em definir parâmetros, características, funções e restrições totalmente racionais ao que representa o conceito Agente, pois a mente humana é intrinsecamente criativa e inovadora. Representando esta gama de possibilidades, tem-se a lista abaixo:

- Interface de cliente usuário (Exemplo: *mail agent*)
- Agentes físicos (Exemplo: robôs)
- Agentes confiáveis (Exemplo: RV (*Virtual Reality*) e gráficos)
- Agentes inteligentes de software

3.3.1 O que é um Agente Inteligente de *Software*?

Um agente de software pode ser definido como um processo autônomo e direcionado a metas. Está situado em seu ambiente estando ciente, reagindo e retornando a seu ambiente. Pode cooperar com outros agentes (de *software* ou humanos) para executar suas tarefas. De um agente de *software* são esperadas três características:

- Adaptação: Os agentes se adaptam aos seus ambientes e usuários, e aprendem com suas experiências;
- Cooperação: Agentes usam linguagens e protocolo padrão para cooperar e colaborar para alcançar metas comuns;
- Autonomia: Agentes agem autonomamente para seguir sua agenda.

A mobilidade, embora não seja rigorosamente necessária, constitui-se em uma característica também importante para um agente. O assunto agentes móveis será abordado posteriormente.

Entre os diversos agentes de *software*, podemos citar, por exemplo:

- Agentes de interface com o usuário ou agentes *HCI inteligentes*
- Agente de modelação e de adaptação de usuários
- Assistentes pessoais
- Agente de recuperação pessoal
- Tecnologia de software móvel
- Agente de cooperação de software
- Agente de pesquisa de recuperações
- Agentes de mercado
- Mediadores e facilitadores

3.4 Tipos de Agentes

Os agentes inteligentes podem ser classificados em várias categorias, de acordo com suas características:

- Conselheiro: oferece ajuda e treinamento. Ensina os passos iniciais para usar um determinado sistema. Fornece suporte contínuo, observando todas as ações do usuário, as quais ele pode interceptar e pedir confirmação. Pode ser consultado para mostrar como executar uma atividade particular, ou então, sugerir métodos alternativos e mais rápidos para executá-la.
- Guia: ajuda a navegação em bancos de dados e hipermídia. Classifica, recupera e filtra grandes quantidades de informações, apresentando somente os dados relevantes e importantes aos usuários, no formato personalizado. Fornece caminhos apropriados para o usuário navegar pelo banco de dados, e auxilia-o caso se sinta "perdido".
- Empregado: executa as atividades tediosas ou repetitivas. Atividades são executadas imediatamente, e algum tipo de feedback pode ser fornecido tanto pelo usuário como pelo próprio agente.
- Representante: trabalha na ausência do usuário. De certa forma, seria parecido ao agente Empregado citado anteriormente, exceto pelo fato de que as atividades não precisam ser imediatamente executadas ou então, são executadas somente após eventos específicos. Por exemplo, pode fazer *Backups* de arquivos de

madrugada ou fazer pedidos de compras, caso algum produto atinja o limite mínimo no estoque.

- Comunicador: trabalha com outros usuários e seus agentes, para assim, conseguir executar as atividades às quais foi designado. Pode, por exemplo, organizar reuniões de recursos e pessoas. Ou então, pode reunir um grupo de agentes para que assim possam executar uma atividade mais complexa.

3.5 Propriedades dos Agentes

Os agentes devem apresentar as seguintes características:

- Aprendizagem: acumulação de conhecimentos;
- Autonomia: utilizado como sinônimo de independência, ou seja, a capacidade do agente operar separadamente e decidir o que fazer enquanto opera separadamente;
- Comunicabilidade: comunica-se com outros agentes, usuários, objetos e seu ambiente;
- Confiabilidade: demonstra veracidade e benevolência nas informações e ações realizadas em nome do usuário;
- Cooperatividade: utilizado como sinônimo de colaboratividade, ou seja, a capacidade dos agentes trabalharem juntos para concluírem tarefas mutuamente benéficas e complexas;
- Degradação gradual: capacidade de o agente executar parte de uma tarefa quando existe incompatibilidade na comunicação ou domínio;
- Discurso: é necessário para a execução das tarefas mais simples, uma vez que o usuário precisa estar seguro de que o agente cumprirá sua agenda e suas tarefas de forma esperada;
- Flexibilidade: escolha dinâmica das ações e da seqüência de execuções das mesmas, em resposta a um estado do ambiente;
- Inteligência: conjunto de recursos, atributos e características que habilitam o agente a decidir que ações executar, ou seja, capacidade de tratar ambigüidades;
- Mobilidade: capacidade de um agente mover-se de uma máquina para outra;

- Persistência: utilizado como sinônimo de continuidade temporal, ou seja, manter um estado interno conciso através do tempo;
- Personalização: Aprende sobre o usuário e adapta suas ações de acordo com ele;
- Planejamento: habilidade de sintetizar e escolher entre diferentes opções de ações desejadas para atingir os objetivos;
- Pró-atividade: exibe oportunismo e comportamento direcionado a objetivos;
- Reatividade: utilizado como sinônimo de sentir e agir, ou seja, percebe o ambiente e responde às modificações que ocorrem nele.
- Representabilidade: representar o usuário em suas ações;
- Responsabilidade: descrição intencional da natureza de uma tarefa delegada.

3.6 Abordagens para a Construção de Agentes

A idéia de empregar agentes inteligentes de software na realização de certas atividades foi introduzida por pesquisadores visionários que perceberam grandes oportunidades nesta promissora área. Uma grande quantidade de trabalhos já foram realizados, principalmente na modelagem e construção de agentes. No entanto as técnicas disponíveis atualmente não permitem a construção de agentes capazes de produzir interações de alto nível, parecidas com as humanas.

Dois problemas principais podem ser identificados na relação entre agentes inteligentes e usuários:

- Competência - Como fazer um agente adquirir conhecimento suficiente para decidir quando agir e qual o tipo de ação ele deve realizar?
- Confiança - Como se pode conseguir com que o usuário sinta-se confortável, ao delegar atividades a um agente?

Estes problemas devem ser tratados como aspectos eminentes na construção de agentes. Atualmente consideram-se duas abordagens básicas para a construção de agentes:

A primeira abordagem consiste em tornar o próprio programa do usuário em um agente inteligente. Um exemplo pode ser o sistema desenvolvido por *Malone* e *Lai*, constituído por "agentes semi-autônomos", os quais utilizam uma coleção de regras

programadas pelo usuário para processar as informações relacionadas a uma atividade particular. O grande problema dessa abordagem é que ela não trata do critério de "competência", de um modo satisfatório. A abordagem requer muita perspicácia, compreensão e esforço por parte do usuário. É ele quem tem de reconhecer a oportunidade para empregar um agente, tomar a iniciativa de criá-lo, fornecer ao agente um conhecimento explícito, e manter constantemente atualizadas todas as regras do agente, na medida em que seus hábitos de trabalho ou interesses possam mudar. Nesta abordagem praticamente não existe o problema da confiança no agente, uma vez que o usuário (normalmente) confia em suas próprias habilidades de programação.

A segunda abordagem ou "abordagem baseada no conhecimento", consiste em dotar um agente com um extensivo conhecimento específico a um determinado domínio, sobre a aplicação e sobre o usuário (chamado de modelo do domínio e modelo do usuário, respectivamente). Em tempo de execução, o agente usa seu conhecimento para reconhecer os planos do usuário e encontrar oportunidades para contribuir com ele. Porém, ambos os critérios de competência e confiança, constituem-se como problemas nesta abordagem. A competência pode ser atingida, porém com altos custos, pois a construção de agentes baseados em conhecimento necessita de uma grande quantidade de trabalho e esforço por parte do engenheiro do conhecimento, para conseguir extrair conhecimento suficiente sobre a aplicação e o domínio. Por outro lado, pouco deste conhecimento ou mesmo da arquitetura de controle do agente podem ser usados na construção de agentes específicos a outras aplicações. O segundo problema é que o conhecimento do agente é fixo, ou seja, não pode ser customizado de acordo com os hábitos e preferências de usuários individuais. Outro problema é o de confiança. Provavelmente, não seria interresante fornecer ao usuário um agente que é muito sofisticado, qualificado e autônomo desde o início do seu uso. Isto pode acarretar no usuário um sentimento de perda de controle e de compreensão da atividade a ser realizada. Uma vez que o agente foi programado por alguma outra pessoa, o usuário pode não compreender as limitações do agente e o modo como ele trabalha, entre outras coisas.

3.7 Desenvolvimento de Agentes

A conduta de um agente autônomo é determinada pela interação de três fatores:

- O ambiente em que o agente opera,
- As tarefas que ele executa,
- E o desenvolvimento do agente propriamente dito.

O primeiro passo para desenvolver um agente é uma análise cuidadosa do ambiente e das tarefas necessárias para determinar os requisitos a respeito de seu comportamento. A arquitetura de um agente é desenvolvida através de um processo interativo em que um projeto é implementado e o comportamento do agente resultante evolui empiricamente até se ajustar ao ambiente.

A interação entre a tarefa, o organismo e o ambiente de um agente determina sua atitude e comportamento.

Para suportar o desenvolvimento de um agente, tem-se um conjunto de ferramentas para avaliação empírica destes. Estas ferramentas têm sido vistas como microscópios para o estudo da conduta de agentes e desta forma espera-se que, assim como a descoberta do microscópio abriu novos horizontes aos biólogos, estas ferramentas irão ampliar o entendimento de fatores que governam efetivamente os agentes autônomos.

3.8 Formas de aprendizagem

Uma abordagem alternativa, a "abordagem de aprendizado", é especificada por *Pattie Maes* (MAES,1994) A hipótese por ela testada é que, sob certas condições, um agente pode "autoprogramar-se", isto é, o próprio agente pode adquirir o conhecimento de que necessita para ajudar o usuário. Ao agente é fornecido um conhecimento mínimo, e ele aprende o "comportamento" do usuário e de outros agentes. As condições particulares que tem de ser satisfeitas para que esta aprendizagem seja possível são: o uso da aplicação têm que envolver uma substancial quantidade de comportamento repetitivo, e este comportamento repetitivo deve ser potencialmente diferente para usuários diferentes.

Se o comportamento demonstrado por diferentes usuários é o mesmo, uma abordagem baseada no conhecimento poderia fornecer os resultados esperados de forma mais rápida do que a abordagem de aprendizado. Porém, se a primeira condição não é satisfeita, o agente não será capaz de aprender nada, pois não há nenhuma regularidade para aprender nas ações do usuário.

Esta abordagem resolve o problema de confiança. Se o agente desenvolve gradualmente suas habilidades, ao usuário é dado tempo suficiente para construir um modelo de como o agente toma decisões, o que é um dos pré-requisitos para um relacionamento de confiança. Com relação ao critério de competência, este também é satisfeito, por três razões principais:

- Requer menos trabalho do usuário final e do desenvolvedor da aplicação;
- O agente pode adaptar-se mais facilmente ao usuário, com o passar do tempo, tornando-se customizado às preferências e hábitos individuais;
- A abordagem facilita a transferência de informações, hábitos e conhecimentos entre diferentes usuários da comunidade.

Assim, segundo *Pattie Maes(MAES,1994)*, o agente irá adquirir conhecimento basicamente de quatro maneiras possíveis:

- "Olhando sobre os ombros do usuário", ou seja, observando a forma como o usuário executa suas ações. Assim, consegue encontrar regularidades e padrões em suas ações, podendo então, oferecer formas de automatizá-las.
- Usando *feedback* direto e indireto. Um *feedback* indireto acontece quando o usuário não aceita uma sugestão do agente e executa uma ação diferente. Já um *feedback* direto ocorre quando o usuário da resposta explícita as ações do agente, como "Não faça isso novamente" ou "Gostei do artigo sugerido".
- O agente aprende através de exemplos explícitos fornecidos pelo usuário. O usuário pode treinar o agente dando exemplos hipotéticos de eventos e situações e dizendo o que o agente deve fazer em tais casos. O agente então registra as ações, traça relacionamentos entre objetos, e muda sua base de exemplos para incorporar o novo exemplo sendo especificado.
- O agente adquire maiores conhecimentos pedindo conselhos a agentes que ajudam outros usuários, os quais realizam atividades semelhantes. Se um agente

não sabe que ação tomar em uma certa situação, pode apresentá-la a outros agentes e pedir por sugestões. Adicionalmente, o agente pode aprender por experiência, porque outros agentes são boas fontes de sugestões. Dessa forma, passa a confiar somente nos agentes que, no passado, forneceram recomendações que agradaram ao seu usuário.

3.9 Comunicação entre Agentes

Como há uma grande variedade de software no mercado, criados por diversas pessoas, dos mais diferentes países, credos e cultura, obviamente há uma enorme diferença entre a forma e estrutura de seus programas. Como os agentes se propõem a estabelecer a comunicação entre os aplicativos do usuário, sejam quais forem, há uma necessidade de criar-se um padrão de comunicação. Logicamente, é necessário que primeiro se crie um padrão entre os agentes para , depois de testado, passe a ser utilizado também nas aplicações comuns.

Criar um padrão não é uma tarefa simples. Existe a necessidade de ter-se um grupo de desenvolvimento. Este tem que levar em consideração a heterogeneidade dos programas, escritos em diferentes situações, por pessoas distintas, em diferentes linguagens, com interfaces também diferentes.

O grupo *DARPA Knowledge Sharing Effort*, usa uma abordagem declarativa, baseada na idéia de que a comunicação pode ser modelada com a troca de afirmações declarativas.Esta Abordagem é compacta e suficientemente expressiva para trocar uma vasta quantidade de diferentes tipos de informações. Este grupo definiu uma Linguagem chamada *ACL(Agent Communication Language)*, que satisfaz estas necessidades.

A *ACL* é constituída por três partes - um vocabulário, uma linguagem interna chamada *KIF* (*Knowledge Interchange Format*) e uma outra linguagem chamada *KQML* (*Knowledge Query and* Manipulation *Language*). Uma mensagem *ACL* é uma expressão *KQML* na qual os argumentos são termos ou sentenças em KIF, formadas por palavras encontradas no vocabulário da *ACL*.

O vocabulário da *ACL* é armazenado em um dicionário com as palavras apropriadas às áreas comuns das aplicações. Cada palavra tem notações formais (em

KIF) e descrição em inglês. É um dicionário aberto, permitindo o acréscimo de novas palavras em áreas já existentes ou então de novas áreas que possam surgir.

3.10 Agentes Móveis

Os agentes móveis tiveram sua origem a partir de uma investigação crítica sobre como a comunicação entre computadores tem sido realizada desde a década de 70. O princípio central da comunicação em redes de computadores é realizado através do *RPC* (*Remote Procedure Call*). Em linhas gerais, o *RPC* é a capacidade de um computador chamar um procedimento em um outro computador. As informações transportadas através da rede são constituídas por dados que servem como argumentos em caso de uma solicitação ou como resultados em caso de resposta. O procedimento é interno ao computador que o executa.

Uma outra alternativa ao *RPC* seria a *RP* (*Remote Procedure*). Através dela, um computador não apenas chama um procedimento em outro computador, mas também fornece o procedimento a ser executado. Se esta abordagem for aprimorada, temos a possibilidade de que o procedimento seja iniciado no computador remetente, e prossiga sua execução no computador destinatário. Este procedimento, que apresenta uma mobilidade através da rede, caracteriza um agente móvel.A *RP* apresenta duas grandes vantagens sobre o *RPC*:

- Tática: o desempenho da *RP* é superior ao *RPC*, pois as invés de comandos, um agente é enviado através da rede, reduzindo assim o tráfego de mensagens.
- Estratégica: a *RP* conduz a padronização, uma vez que o agente adiciona sua funcionalidade apenas ao cliente, sem a necessidade de uma disponibilização deste serviço no servidor.

Um agente móvel é composto por:

- Código: o programa que define o agente;
- Atributos: descrevem as informações do agente, tais como sua origem e proprietário, requerimentos de recursos, chaves de autenticação, tarefas já executadas, etc.
- Estado: variáveis internas ao agente, que lhe permite continuar suas atividades após mover-se para outro computador.

3.11 Sistema Multi-Agentes

Uma vez que existe a linguagem de comunicação (ou protocolo) e a tecnologia para se construir agentes está disponível, uma pergunta surge: Como os agentes devem ser organizados para colaborarem entre si? Duas abordagens diferentes têm sido exploradas:

- Comunicação direta - os próprios agentes cuidam da coordenação;
- Coordenação auxiliada - há programas especiais para organizar a coordenação.

Embora ambas sejam interessantes, uma abordagem que permita a união das duas seria mais prática. Utiliza-se então a abordagem chamada de sistema federado. É um sistema em que os agentes não se comunicam diretamente entre si, e sim através de um supervisor, que se encarrega de comunicar-se entre os diversos supervisores e da comunicação entre seus agentes.

O supervisor precisará suportar alguns requisitos básicos, permitindo que os agentes possam:

- Pedir informações de outros agentes;
- Observar as atividades de outros agentes;
- Interceptar e mudar os pedidos destinados para outros agentes;
- Estabelecer comunicação com agentes sob o controle de outros supervisores.

3.12 *CORBA* e Agentes Móveis

CORBA (*Common Object Request Broker Architecture*) é um padrão aberto utilizado na construção de sistemas distribuídos orientados a objeto. Os principais elementos da arquitetura são os objetos distribuídos e o *ORB* (*Object Request Broker*). Os primeiros são os componentes básicos do software e o *ORB* é o mecanismo pelo qual esses componentes trocam mensagens.

Algumas características da arquitetura, tais como um mecanismo robusto de troca de mensagens (*ORB*) e uma interface comum entre os objetos, fazem do *CORBA* uma excelente plataforma para a implementação de sistemas multi-agentes. A *OMG* (*Object Management Group*), grupo responsável pelo *CORBA*, recentemente especificou como o mesmo deve ser usado na construção de agentes móveis através da especificação *CORBA Mobile Agentes*.

3.13 Características que tornam uma aplicação apropriada para Agentes

Segundo *Andrew Wood (WOOD,1994)* as seguintes caracteristicas são necessárias para aplicar o paradigma de agentes:

- Adaptação: Tarefa que requer um certo grau de adaptabilidade; o agente necessita desenvolver habilidades para executá-la aprendendo melhores ou novos meios, o que também inclui métodos para evitar falhas e adaptar-se às necessidades, desejos e objetivos pessoais do usuário.
- Pesquisa: A tarefa não é completamente definida; o agente deve considerar uma grande quantidade de possíveis soluções, escolhendo uma das mais adequadas de acordo com sua experiência.
- Demonstração: A tarefa envolve aprendizado e treinamento. Isto inclui ensinar os usuários a usarem ferramentas de software de maneira eficaz, e também, de certa forma, fornecer explicações sobre o que o agente está fazendo.
- Ajuda: A tarefa requer um certo grau de cooperação entre o usuário e o agente. O agente poderia fazer críticas construtivas sobre a forma do usuário trabalhar, ou fornecer "dicas" sobre como utilizar de maneira mais eficiente os recursos do sistema.
- Autonomia: A tarefa requer atenção constante ou regular, mas pouca ou nenhuma entrada ou interação. Dessa forma, delegar esta tarefa para um agente seria muito útil e benéfico. Um exemplo seria o monitoramento de sistemas simples, onde uma mudança no comportamento do sistema poderia gerar a execução automática de alguma tarefa ou ação por parte do agente.
- Assincronismo: A tarefa tem um intervalo significativo entre seu início e término. Este intervalo poderia ser devido ao tempo de processamento de grandes quantidades de informação ou mesmo a falta de informações vitais em um determinado momento.

3.14 Vantagens na Utilização de agentes inteligentes

Entre as vantagens do uso dos agentes inteligentes destaca-se:

- Soluções integradas
- Recuperação de informação distribuída
- Eficiência, indexação flexível e recuperação
- Interfaces e buscas
- Roteamento e filtro
- Recuperação efetiva
  Recuperação em Multimídia
- Extração de informação
- Feedback relevante

3.15 Agentes Inteligentes para Internet

A Internet é o meio de comunicação em voga e os Agentes Inteligentes estão substituindo as aplicações convencionais, modificando o trabalho de processar um grande volume de informações.

No futuro, os Agentes Inteligentes poderão ser o melhor meio de se pesquisar na Internet, porque, independente de quão organizada ela esteja, nunca acompanhará seu crescimento em informação, ou seja, ela ainda não tem mecanismos para controlar e acompanhar seu próprio crescimento, sendo necessária a adoção dos agentes inteligentes, facilitando ao usuário poder tirar proveito das informações existentes.

O fato dos Agentes Inteligentes serem até então de natureza estacionária, limita seu escopo de atuação a uma máquina isolada da rede. Se acrescentarmos a capacidade de mobilidade do agente, então, enriquecemos o processo ampliando o escopo de atuação do mesmo para toda a rede.

Isso tornaria o agente independente para buscar informações, cooperar, decidir e adaptar-se ao contexto em que se encontra.

Visando dar aos agentes essa capacidade adicional, surgiu uma linha de pesquisa advinda da Internet, o estudo dos *Aglets* (agentes móveis para Internet), que é o passo inicial para que o agente adquira essa importante capacidade de comunicação no meio em que se encontra.

Os Aglets são objetos escritos na linguagem Java, uma combinação de agente e *Applets*. São um tipo de Software autônomo, uma vez que podem parar sua própria execução , montar um itinerário e continuar sua execução em um local remoto , podem perfazer ações como a de navegar pela rede sem a necessidade de intervenção externa . Estritamente falando, um *Aglet* não é um Agente Inteligente Autônomo, porque lhe falta conhecimento e capacidade de decisão.

Para suportar as diversas arquiteturas de Agentes Inteligentes Autônomos sem alterar sua capacidade de interferir no meio em que se encontra e ainda possibilitar que eles atuem em ambientes diversos da mesma forma, são necessárias as Bancadas de Trabalho. Ou seja, é necessário um mecanismo para que agentes com arquiteturas heterogêneas e em ambientes diversos possam atuar fornecendo sempre resultados confiáveis.

Bancadas de Trabalho são ferramentas para construção e simulação de Agentes Inteligentes heterogêneos, fornecendo uma arquitetura de suporte na qual o usuário descreve seus próprios agentes e ambientes onde estes evoluem. A bancada de trabalho *Aglet Workbench* foi implementada pela *IBM(International Buisines Machines)* com o fim específico de construir-se agentes móveis (*Aglets*).(Maiores informações veja em AGLET,2001, e no item 3.15.1 deste documento)

3.15.1 *Aglets*

Visando explorar os recursos da Internet, com a finalidade de desenvolver uma tecnologia que se adapte às constantes mudanças deste meio e que ainda seja capaz de se locomover autonomamente pelos diversos itinerários conhecidos ou não, os laboratórios da *IBM* desenvolveram o projeto dos *Aglets*. Trata-se de uma nova tecnologia, a fusão da autonomia de um agente de software com o dinamismo dos *Applets*.(*Applets* são mini-aplicativos escritos na linguagem Java, capazes de serem executados em páginas da WWW. Foram grandes inovações providas pelo Java, uma vez que seu código é executado pela máquina hospedeira, e não por um servidor.).

O *Aglet* representa um novo passo na busca da evolução de aplicações executáveis na Internet, inserindo códigos de programa que podem ser transportados levando consigo as informações de estado (contexto).

Um *Aglet* é um objeto escrito em Java, uma vez que essa linguagem é própria para Internet. Ele pode ser transportado de um *Host* ((hospedeiro, trata-se de uma *URL-Uniform Resource* Locator) , um endereço dentro da Internet onde se localiza uma máquina remota) ao outro , podendo executar-se , parar sua execução , salvar contexto e despachar-se para outro *host* levando além do código , todos os dados com que trabalhava .

O *Aglet* possui autonomia, pois, uma vez iniciado pode decidir por si só (sem a intervenção do usuário), onde irá e o que irá fazer. É capaz de receber dados e solicitações do mundo externo, porém, decide o modo pelo qual irá usá-los e como atendê-las.

Aglets podem ser associados a Inteligência Artificial, porém não necessariamente. Se não portarem inteligência, são considerados apenas agentes móveis.

3.15.2 Conceitos Fundamentais

Como qualquer outro programa escrito em Java, os *Aglets* são construídos através de classes e métodos. As classes são implementadas no Java *Aglet Application Programming Interface* (*J-AAPI*), um padrão desenvolvido pela *IBM* para construção de *Aglets*. Seus principais métodos definem o ciclo de vida pelo qual o *Aglet* irá passar, desde sua criação até sua retirada do contexto em que está. A inclusão de classes e métodos diferentes por parte do usuário fará com que o *Aglet* possua uma identidade, conferindo a ele a capacidade de executar uma tarefa específica com maior ou menor grau de inteligência e autonomia. Isso significa que o *Aglet* possui métodos básicos que serão sempre utilizados. Porém, cabe ao programador ocupar-se de desenvolver métodos específicos para a tarefa que deseja realizar. Por outro lado, deve programar de modo a extrair o máximo das capacidades de mobilidade e autonomia desta nova tecnologia com a finalidade de implementar um agente inteligente.

Todas as atividades realizadas por um *Aglet* seguem seu ciclo de vida básico, que é caracterizado pela chamada dos métodos definidos no *J-AAPI*. Os principais métodos deste ciclo são:

- Criação: criação e clonagem;
- Destruição: retirada;
- Mobilidade: despachar e retornar;

- Armazenagem: desativação e ativação;
- Comunicação: troca de mensagens;

A vida de um *Aglet* começa em sua criação, onde seus métodos serão invocados e ele será inserido em um contexto.Na sua criação, o *Aglet* recebe um identificador único e imutável que será seu mecanismo de referência no meio em que está.Um novo *Aglet* pode ser criado a partir da solicitação do usuário ou a partir da clonagem de um *Aglet* já existente. Uma vez criado, o *Aglet* é capaz de se locomover no ambiente, vagando de um *host* ao outro de modo a executar com eficiência suas tarefas. Na capacidade de locomoção encontra-se o método de despacho, quando o *Aglet* interrompe sua execução no *host* em que está e desloca-se para um *host* destino, onde entrará em execução novamente do ponto em que parou e o método de retração, quando retorna ao *host* original. No decorrer do processo de execução de suas tarefas, o *Aglet* pode "parar para dormir". Assim é denominada a capacidade do mesmo de parar sua execução e armazenar-se em qualquer outro repositório (diminuição do fluxo das redes), sendo pré-determinado o período no qual ele estará desativado. Esta capacidade é oferecida pelos métodos de armazenagem do *Aglet*.

Capítulo 4

4.1 Gerência de redes:

4.2 Introdução

À medida que a tecnologia de redes evolui, é necessário ter um gerenciamento eficiente para garantir a integridade da rede , e desta forma manter o sue desempenho de forma a satisfaszer sueus usuários.

O modelo de referência *ISO/OSI(International Standards Oraganization/Open System Interconnection)* subdividiu a gerência de redes em cinco grandes áreas funcionais(modelo funcional): Gerência de Falhas, de Configuração, de Desempenho, de Segurança e de Contabilização (KLERER, 1988). Essas funções têm sido comumente aplicadas para desenvolver uma gerência reativa de redes, ou seja, estão sendo utilizadas na detecção de problemas na rede e na busca de uma solução quando esses problemas ocorrem.

Com o crescimento e evolução das redes de computadores, a gerência reativa torna-se muito limitada. Acredita-se que associando a inteligência artificial aos sistemas de gerência de redes é possível desenvolver um gerenciamento pró-ativo nesses ambientes.

4.3 Gerência Pró-ativa

A gerência pró-ativa de redes é a capacidade de prever certos problemas impedindo-os de provocar a degradação dos serviços oferecidos por estes recursos.

Assim como existe a possibilidade de se usar programas de simulação, também sistemas especialistas podem ser utilizados com o mesmo objetivo de gerenciar uma rede de forma pró-ativa. Esses sistemas poderiam apoiar os administradores de redes, emitindo sugestões de solução, evitando assim a ocorrência de maiores problemas na rede. Entretanto, para se determinar uma efetiva gerência pró-ativa de redes, é necessário que sistemas desse tipo façam parte das futuras plataformas de gerenciamento(JANDER,1993).

4.4 O modelo funcional proposto pela *ISO/OSI*

4.4.1 Gerência de Configuração

A gerência de configuração consiste no mapeamento dos recursos que compõem o ambiente de rede, tais como *hubs*, *switches*, roteadores, estações de trabalho, servidores, circuitos e impressoras conectadas à rede.

O objetivo da gerência de configuração é permitir uma visão abrangente dos recursos instalados e disponíveis nos ambientes, de forma a possibilitar, por intermédio de inventários e mapeamento da rede, a identificação de alterações e o planejamento de capacidade para novas implementações de software ou hardware.

4.4.1.1 Funções da Gerência de Configuração:

Entre as funções da gerência de configuração destacam-se:

- Mapeamento dos recursos de rede, tais como *hubs*, *switches*, roteadores, estações de trabalho, servidores, circuitos e impressoras conectadas à rede;
- Execução de inventário de hardware e software em tempo de *login*;
- Emissão de alertas sobre a alteração de configuração na topologia ou nos objetos inventariados;
- Configuração do programa de inventário permitindo a inclusão de novos software ou hardware;
- Manter histórico da topologia de forma a possibilitar comparativos ou reconfiguração;
- Identificar a adição ou remoção de novos objetos na topologia;
- Definir periodicidade em que os inventários serão realizados de acordo com as especificações das redes gerenciadas;
- Implementar políticas de configuração de *desktops* visando a diminuição de custos de administração;
- Emissão de relatórios configuráveis do inventário dos objetos gerenciados e da topologia de rede;
- Distribuição de software para estações e servidores de acordo com o estabelecimento prévio de requisitos de configuração;

- Recebimento de alertas relacionados a problemas de desvios ou não conformidade no processo de gerência e controle da configuração;
- Habilitar controle remoto de estação e servidores para investigação e diagnóstico de falha.

4.4.2 Gerência de Falhas

O objetivo da gerência de falhas é permitir a identificação de falhas que possam provocar ou provocam a indisponibilidade de recursos ou serviços, com a indicação de alertas aos gerentes de rede, permitindo uma atuação pró-ativa ou corretiva dos eventos, configurados previamente de acordo com o seu grau de importância (informativo, menor, maior, crítico).

4.4.2.1 Funções da Gerência de Falhas:

Entre as funções da gerência de falhas destacam-se:

- Definição de *thresholds* que indiquem a iminência de falhas nos objetos gerênciados (*hubs*, *switches*, roteadores, circuitos, estações de trabalho, servidores, impressoras e aplicações), conforme métricas a serem definidas;
- Configuração de alertas em caso de limites alcançados, de acordo com os *thresholds* definidos;
- Emissão de alertas no painel de gerência indicando erro ou indisponibilidade de recursos ou serviços, correlacionando-os ao grau de importância do evento;
- Configuração de mecanismos que permitam o acionamento de responsáveis, em caso de falhas ou anormalidade, através de *e-mail*, *bip*, *pager*, etc.;
- Habilitação de *logs* contendo histórico de ocorrências de falhas e ações implementadas para a solução do problema, gerando base de conhecimento que permita agilizar e apoiar a solução de novas ocorrências.

4.4.3 Gerência de Desempenho

O objetivo da gerência de desempenho é permitir o acompanhamento da disponibilidade dos recursos de rede e serviços, bem como o seu nível de utilização, confrontando-os com os aspectos de configuração dos componentes de rede e aplicação, de forma a estabelecer os níveis de serviços adequados aos ambientes operacionais definidos.

4.4.3.1 Funções da Gerência de Desempenho:

Entre as funções da gerência de desempenho destacan-se:

- Acompanhamento de tráfego de pacotes em equipamentos de infra-estrutura de rede (*hubs*, *switches*, roteadores e circuitos) visando estabelecer níveis adequados de desempenho, conforme especificação de cada equipamento;
- Habilitação de coleta de dados referentes ao número de pacotes trafegados em relação à taxa de colisão e broadcast na rede, visando estabelecer níveis adequados de performance no meio de transmissão, de acordo com as métricas a serem definidas;
- Coleta de informações sobre a taxa de utilização por protocolos na rede, visando isolar ou identificar tráfego desnecessário de protocolos ou serviços de rede (*broadcast*, colisões);
- Coleta de informações referentes ao tráfego de pacotes por segmento de rede, estabelecendo o nível de utilização e performance adequadas de acordo com os serviços disponíveis na rede, implementando comparativos com os demais segmentos visando diagnosticar problemas de desempenho;
- Habilitação de ferramentas que permitam a simulação de aumento de carga de serviços na rede, visando estabelecer níveis de capacidade versus desempenho, permitindo uma avaliação de adição de novos serviços ou recursos de rede;
- Habilitação do gerenciamento de componentes de servidores (agentes do software de gerência a ser utilizado ou através da integração dos já existentes) que indiquem nível de utilização de *UCP*, memória, adaptador de rede, disco, visando estabelecer configuração adequada do hardware em relação ao uso de aplicações ou serviços;

- Monitoração e disponibilização de informações sobre o tempo de reposta de aplicações ou serviços, considerando-se o tempo de resposta dos componentes de rede, tais como *LAN*, *UCP*, banco de dados, link entre roteadores, de forma a mensurar o tempo de resposta total por aplicação;
- Monitoração do número simultâneo de usuários por aplicação, visando estabelecer níveis adequados de capacidade e desempenho;
- Habilitação de *thresholds* que disparem alarmes aos gerentes de rede indicando nível de desempenho inadequado, como base em taxa de utilização de componentes acima de patamares previamente definidos, como uso de UCP, memória, erros e colisões no meio de transmissão.

4.4.4 Gerência de Contabilização

A gerência de contabilização consiste no acompanhamento dos custos de utilização de recursos que compõem o ambiente operacional, determinando os níveis de utilização de cada objeto ou serviço gerenciado.

O objetivo do processo de gerência de contabilização é acompanhar o nível de utilização de recursos ou serviços, subsidiando a implantação de políticas de uso de recursos de infra-estrutura.

4.4.4.1 Funções da Gerência de Contabilização

Entre as funções da Gerência de Contabilização destacam-se:

- Implantação de tecnologia para segmentação do acesso aos recursos de rede, permitindo a contabilização do volume de uso de recursos de infra-estrutura e serviços de rede, por cliente;
- Acompanhamento e apresentação de percentuais de utilização de aplicações;
- Acompanhamento e apresentação de freqüência de usuários simultâneos nas aplicações;
- Acompanhamento e apresentação de consumo de disco, CPU, memória e banda de *links* de comunicação por aplicação/cliente;
- Acompanhamento e apresentação do número de transações por aplicação;

- Habilitação de alertas que indiquem alta taxa de utilização de rede e aplicação, visando estabelecer qualidade na utilização dos recursos;
- Habilitação de *logs* de registros.

4.4.5 Gerência de Segurança

A gerência de segurança consiste na integração dos modelos de gerência aos requisitos de segurança implementados nos ambientes operacionais, possibilitando o acompanhamento e acionamento de alertas sobre eventos correlacionados à ruptura dos esquemas de segurança definidos.

O objetivo da gerência de segurança é manter o controle de acesso dos usuários. Para isso devem ser implementados alertas, *logs*, auditorias que auxiliem a manter a segurança dos recursos disponíveis.

4.4.5.1 Funções da Gerência de Segurança:

Entre as funções da gerência de segurança destacam-se:

- Integração com os sistemas de autenticação para os ambientes operacionais definidos, por intermédio de agentes a serem instalados, visando estabelecer o envio de alertas relacionados aos eventos de tentativa de acesso indevido;
- Definição e emissão de alertas sobre o bloqueio de contas nos sistemas disponíveis, visando estabelecer o uso de *logs* de acompanhamento destes eventos;
- Integração com as ferramentas de auditoria implementadas nos sistemas de rede ou aplicação, visando habilitar alarmes de eventos previamente definidos, proporcionando a execução de medidas preventivas de erradicação de ações destrutivas intencionais ou acidentais;
- Integração com as soluções de antivírus hoje disponíveis, indicando falhas na distribuição de assinaturas e alertas de incidências de vírus;
- Integração com as ferramentas de backup hoje disponível com a habilitação de alarmes indicando término normal ou anormal das rotinas (*abend*), com registro em *logs* para consultas, visando auditar os eventos.

- Habilitação de *MIBS(Management Information Base)* nos equipamentos de infra-estrutura de rede, possibilitando a interação com os softwares de gerenciamento proprietários, visando a integração de alertas em nível de mau funcionamento, provocado por excessivos erros, colisões, *jabbers*, etc.

4.5 Gerência de redes Sob *TCPIP*

Existe um grande numero de de redes de computadores baseadas nos protocolosTCP/IP (*Transmission Control Protocol*/*Internet Protocol)*, como é o caso da Internet. Isto deve-se ao fato de serem protocolos de fácil implementação e manutenção. Além disso permitem a interligação de redes locais através de redes de longa distância, com um desempenho considerável.

O gerênciamento das redes TCP/IP é realizado através do protocolo SNMP (*Simple Network Management Protocol)*, que permite que uma ou mais máquinas na rede sejam designadas como gerentes da rede. Estas máquinas recebem informações de todas as outras máquinas da rede, e através do processamento destas informações pode gerenciar toda a rede e detectar facilmente problemas ocorridos.

As informações coletadas pela máquina gerente estão armazenadas nas próprias máquinas da rede, em uma base de dados conhecida como MIB (*Management Information Base)*. Nesta base de dados estão gravadas todas as informações necessárias para o gerenciamento deste dispositivo, através de variáveis que são requeridas pela estação gerente.

Entretanto, em uma interligação de diversas redes locais, pode ser que uma rede local esteja funcionando perfeitamente, mas sem conexão com as outras redes, e, conseqüentemente, sem conexão com a máquina gerente. O ideal é implementar em alguma máquina, dentro desta rede local, um protocolo para gerenciamento que permita um trabalho off-line, isto é, que a rede local possa ser gerenciada, ou pelo menos tenha suas informações de gerenciamento coletadas, mesmo que estas informações não sejam enviadas instantaneamente à estação gerente.

O protocolo *Remote Monitoring* (*RMON*) permite uma implementação neste sentido, devendo ser implementado em diversas máquinas ao longo da rede. É possível, ainda, que uma estação com implementação *RMON*, envie dados à estação gerente

apenas em uma situação de falha na rede. Isto contribuiria para redução do tráfego de informações de controle na rede (*overhead*).

4.5.1 Modelo de Gerenciamento:

Em redes *IP*, o sistema de gerenciamento segue o modelo gerente-agente, onde o gerente é o próprio sistema de gerenciamento e o agente é um software que deve ser instalado nos equipamentos gerenciados. A tarefa do agente é responder as requisições feitas pelo gerente em relação ao equipamento no qual o agente está instalado. Esta interação é viabilizada pelo protocolo de gerenciamento *Simple Network Management Protocol* (*SNMP*), o qual é como uma linguagem comum utilizada exclusivamente para a troca de informações de gerenciamento. Dessa forma, o gerente consegue conversar com qualquer máquina que fale *SNMP*, independente do tipo de hardware e sistema operacional. O conjunto de informações ao qual o gerente pode fazer requisições ou alterações é denominado de *Management Information Base* (*MIB*).

4.5.2 Simple Network Management Protocol (SNMP)

O protocolo *SNMP* foi inicialmente idealizado em 1989. Sua arquitetura é baseada no modelo Internet para redes, e sua localização é equivalente à da camada aplicação, no modelo *OSI*.

O *SNMP* é um protocolo designado para facilitar a troca de informações de gerenciamento entre dispositivos de rede. Usando os dados transportados pelo *SNMP*, os administradores de rede podem gerenciar mais facilmente a performance da rede, encontrar e solucionar problemas e planejar com mais precisão uma possível expansão da rede.

O protocolo de gerenciamento *SNMP* constitui atualmente um padrão operacional "de facto", e grande parte do seu sucesso se deve a sua simplicidade, sendo um protocolo *send/receive* com apenas quatro operações. Outro aspecto importante é a sua capacidade de gerenciar redes heterogêneas constituídas de diferentes tecnologias, protocolos e sistemas operacionais. Dessa forma, o *SNMP* pode gerenciar, por exemplo, redes *Ethernet*, *Token Ring* e etc, conectando *IBM PCs*, *Apple Machintosh*, estações de

trabalho *SUN* e outros tipos de computadores (ODA, 1994). As aplicações de gerenciamento utilizam o *SNMP* para:

- Fazer *polling* nos dispositivos de rede e coletar dados estatísticos para análise em tempo real.
- Receber um conjunto limitado de notificações de eventos significativos ou mensagens *trap*.
- Reconfigurar dispositivos de rede.

Como o *TCP*/*IP*, o *SNMP* é um protocolo Internet. Ele é uma parte da arquitetura de gerenciamento da Internet, que é baseada na interação de diversas entidades, como se segue:

- Elementos de rede - também chamados dispositivos gerenciados, os elementos de rede são dispositivos de hardware como os computadores, roteadores, e servidores de terminais que estão conectados a rede.
- Agentes - são módulos de software que residem nos elementos de rede. Eles coletam e armazenam informações de gerenciamento como o número de pacotes de erros recebidos pelo elemento de rede. São eles que respondem às solicitações dos gerentes.
- Objeto gerenciado - um objeto gerenciado é qualquer elemento que possa ser gerenciado. Por exemplo, uma lista dos circuitos *TCP* atualmente ativos em um *host* particular é um objeto gerenciado.
- *MIB* - a *MIB(Management Information Base)* é uma coleção de objetos gerenciados residentes em um armazenamento virtual de informações. Coleções de objetos gerenciados relacionados são definidas em módulos específicos da *MIB*.
- Notação sintática - é a linguagem usada para descrever os objetos gerenciados da *MIB* em um formato independente da plataforma. Um uso consistente da notação sintática permite que diferentes tipos de computadores compartilhem informações. Sistemas de gerenciamento Internet usam um subconjunto da *Open System Interconnection* (*OSI*) *Abstract Syntax Notation* 1 (*ASN.1*) da *International Organization for Standardization's* (*ISO*) para definir tanto os pacotes que são trocados pelo protocolo de gerenciamento quanto os objetos que ele deve gerenciar.

- *Structure of Management Information* (*SMI*) - o *SMI* define as regras para descrever as informações de gerenciamento. O *SMI* é definido usando *ASN. 1*.
- *Network Management Stations* (*NMS*) - também chamados consoles, estes dispositivos executam aplicações de gerenciamento para monitorar e controlar elementos de rede. Fisicamente, os NMS são usualmente *workstations* com *UCP* velozes, monitores coloridos de alta definição, memória substancial e um grande espaço em disco.
- Protocolo de gerenciamento - um protocolo de gerenciamento é usado para transportar informações de gerenciamento entre agentes e *NMS*. O *SNMP* é o protocolo de gerenciamento padrão da comunidade Internet.

Com base nesta arquitetura, o *SNMP* foi construído para minimizar a quantidade e a complexidade das funções necessárias para gerenciar um agente. O paradigma funcional de controle e monitoração do protocolo foi definido de maneira extensiva, para poder absorver mais facilmente novos aspectos das operações de rede e gerenciamento. Além disto, esta arquitetura é totalmente independente da plataforma dos elementos da rede e dos *NMS*.

Os processos que implementam as funções de gerenciamento Internet atuam ou como agentes ou como gerentes. Os agentes coletam junto aos dispositivos gerenciados as informações relevantes ao gerenciamento da rede. O gerente, por sua vez, processa essas informações com o objetivo de detectar falhas no funcionamento dos elementos da rede, para que possam ser tomadas providências no sentido de contornar os problemas que ocorrem como conseqüência das falhas.

Um objeto gerenciado representa um recurso e pode ser visto como uma coleção de variáveis cujo valor pode ser lido ou alterado. Para tanto o gerente envia comandos aos agentes. Para monitorar os dispositivos gerenciados, o gerente solicita ao agente uma leitura no valor das variáveis mantidas por estes dispositivos, através do comando Get, e o agente responde através do comando *Response*.

Para controlar os dispositivos gerenciados, o gerente modifica o valor das variáveis armazenadas nos dispositivos gerenciados, através do comando *Put*. Isto pode ser usado para disparar indiretamente a execução de operações nos recursos associados aos objetos gerenciados. Por exemplo, um *reboot* do elemento de rede pode ser

facilmente implementado, basta que o gerente modifique o parâmetro que indica o tempo até uma reinicialização do sistema.

O gerente pode ainda determinar que variável um dispositivo gerenciado suporta e colher informações de forma seqüencial, das tabelas de variáveis (como as tabelas de roteamento *IP*) nos dispositivos gerenciados. Para isto, ele utiliza as operações transversais (*transversal operations*).

Em alguns casos é necessário que a troca de informações seja em sentido inverso, isto é, o agente tem de passar informações para o gerente. O *SNMP* define a operação *Trap* para que um agente informe ao gerente a ocorrência de um evento específico.

4.5.2.1 Operações *SNMP*

O *SNMP* por si só é um protocolo de requisição/resposta simples. Os *NMS* podem enviar múltiplas requisições sem receber uma resposta. Quatro operações são definidas no *SNMP*(CISCO,1996):

- Get - permite que o *NMS* recupere uma instância de objeto do agente.
- Getnext - permite que o *NMS* recupere a próxima instância de objetos de uma tabela ou lista em um agente. Se o *NMS* quiser recuperar todos os elementos de uma tabela de um agente, ele inicia com uma operação *Get* seguida de uma série de operações *Getnext*.
- *Set* - permite que o *NMS* modifique valores de uma instância de objetos em um agente.
- *Trap* - usado pelo agente para informar assincronicamente o *NMS* sobre algum evento.

4.5.2.2 *SNMPv*.2 (1993)

A versão 2 do *SNMP*(*SNMPv*.2) busca corrigir algumas deficiências da versão 1. Ele basicamente surgiu da evolução do *SNMPv*.1 e do *RMON*. Ele utiliza a *SMI2*, que permite a presença de novos tipos *ASN. 1*. Além disto, ele permite a criação e exclusão de objetos, juntamente com a comunicação entre gerentes através da chamada *Manager* to *Manager MIB*.

As seguintes operaçõpes foram incluídas:

- *GetBulkRequest*: Pedido do gerente para leitura de trechos específicos da *MIB*.
- *InformRequest*/*Response*: Representa a implementação do *Trap* confirmado. O *InformRequest* é o *Trap* propriamente dito e o *InformeResponse*, a confirmação dada pelo gerente.

Aspectos de segurança também foram considerados. Nas versões denominadas *SNMPv2u* e *SNMPv2*, existe a segurança feita por usuário (*user-based*), o que só permite a realização de operações por usuários específicos, e não qualquer um.

43.5.2.3 *SNMPv*.3 (1997)

A versão 3 do *SNMP* (*SNMPv*.3) trouxe como principais vantagens aspectos ligados à segurança. Esta segurança busca evitar a alteração das mensagens enviadas. Além disto, barra-se o acesso a elementos estranhos à execução de operações de controle, que são realizadas através da primitiva *SetRequest*. Evita-se também a leitura das mensagens por parte de estranhos, além de se garantir ao gerente o direito de alteração da senha dos agentes.

A segurança é conseguida através da introdução de mecanismos de criptografia com o *DES* (*Data Scryption Standard*) e de algoritmos de autenticação que podem ser tanto o *MD5* quanto o *SHA* (Secure *Hash Algorithm*).

4.5.3 Protocolo *CMIP*

Assim como o protocolo *SNMP*, a *ISO* propõe como solução para gerenciamento de redes o protocolo *CMIP* (*Common Management Information Protocol*), que também define em seu escopo os papéis de gerente e agente que trocam informações sobre os recursos gerenciados e que são armazenados em *MIBs*.

O protocolo *CMIP* engloba vários tipos de *PDUs*, que são mapeadas em operações análogas ao *SNMP*. São elas:

- *M-Action*: Execução de alguma ação sobre um objeto gerenciado
- *M-Create*: Criação de uma instância de um objeto gerenciado
- *M-Delete*: Remoção de uma instância de um objeto gerenciado
- *M-Get*: Leitura de atributos dos objetos gerenciados

- *M-Set*: Modificação de atributos de objetos gerenciados
- *MEVENT-REPORT*: Notificação de um evento associado a um objeto gerenciado

Também são definidos recursos adicionais que permitem selecionar o grupo de objetos sobre os quais se aplica uma determinada operação. O *scoping*, como é chamado este recurso, permite selecionar um grupo instância de objeto sobre os quais se realizará uma única operação.

Por meio dos recursos de filtro, outra facilidade do *CMIP*, é possível definir um conjunto de testes aplicáveis a um grupo de instâncias de objetos, que fora anteriormente selecionado através do *scoping*. Assim sendo, é possível reduzir significativamente a extração sobre a qual se desenrolará uma operação de gerenciamento.

Além destes, existe o recurso de sincronização, que permite sincronizar diversas operações de gerenciamento realizadas sobre instâncias de objetos selecionados através de recursos de *scoping* e filtro.

Existe uma terceira proposta chamada de *CMOT* (*CMIP Over TCP/IP*) cujo objetivo é permitir o uso do *CMIP* em redes com o protocolo *TCP/IP*.

Se fizermos uma comparação entre o *SNMP* e o *CMIP*, veremos que o *SNMP* é excessivamente simples quando usado em aplicações que não foram previstas quando foi definido, e que apresenta deficiências em relação à segurança ao ser usado em aplicações mais críticas. Já o *CMIP* é um protocolo poderoso e abrangente, que já foi concebido com o objetivo de adequar-se à complexidade das redes. Mas apesar desta característica, ainda não alcançou um grau estável de aceitação pela comunidade.

As projeções de mercado demonstram que o *SNMP* continuará sendo muito usado em pequenas redes, enquanto que o *CMIP* deve dominar o mercado composto pelas grandes redes corporativas e redes públicas de telecomunicações.

Os serviços do *CMIS* e o protocolo *CMIP* são oferecidos na camada de aplicação, sendo usados para implementar sistemas desenvolvidos para vários propósitos, como gerenciamento de desempenho, do nível de falhas, de segurança, de configuração e de contabilidade, usando os recursos de uma rede baseada no modelo de comunicação *OSI*.

O Serviço de Informação de Gerenciamento (*CMIS - Common Management Information Service*) são os serviços prestados na camada de aplicação. São orientados à conexão, necessitando de um canal virtual (uma associação) para troca de informações. Permite definir os objetos através da seleção de sub-árvores (*scopping*), ou através do uso de predicados (filtragem). O *CMIS* apresenta os seguintes serviços as aplicações de gerenciamento:

- *GET*: Para obter informações (atributos) de um objeto gerenciado;
- *SET*: Para alterar os atributos de um objeto gerenciado;
- *ACTION*: Para executar um comando (ação ou operação) sobre um objeto gerenciado;
- *CREATE*: Para criar uma nova instância de um objeto;
- *DELETE*: Para descartar uma instância de um objeto (removê-la);
- *EVENT-REPORT*: Para o relato de ocorrências excepcionais (notificações sobre um evento associado a um objeto gerenciado).

Além das funções apresentadas pelo *CMIS* no protocolo *CMIP*, o *CMIS* apresenta facilidades adicionais que permitem selecionar um conjunto de objetos sobre o qual pode-se aplicar a mesma operação, e também a existência de respostas múltiplas para cada requisição (uma para cada objeto gerenciado). São três facilidades adicionais:

- O *scoping* permite que selecionemos um grupo de instâncias de objetos gerenciados sobre o qual aplicaremos uma única operação;
- O filtro nos dá a possibilidade de definir um conjunto de testes que serão aplicados a um grupo de instâncias de um objeto, selecionado por uma operação de *scoping* anterior, permitindo formar um grupo menor a partir deste, sobre o qual as operações de gerenciamento devem ser aplicadas;
- A sincronização permite que sincronizemos várias operações de gerenciamento a serem aplicadas a instâncias de objetos gerenciados, obtidas através do uso das operações de *scopping* e de filtragem.

O CMISE (*Common Management Information Service* Element) implementa os serviços definidos pelo *CMIS*, executando o protocolo *CMIP*. É correspondente ao mecanismo *SASE* (*Special* Application *Service Element*) da camada de aplicação, e

utiliza os elementos *ACSE* (*Association Control Service* Element) e *ROSE* (*Remote Operations Service Element*) que juntos correspondem ao mecanismo de *CASE* (*Common Application Service Element*) também da camada de aplicação.

O protocolo *CMIP* apresenta uma forma inteligível comum utilizada para transferir as informações de gerenciamento entre as entidades pares na comunicação de gerenciamento, sendo que uma destas atua como um gerente, enquanto a outra atua como agente, sendo que as informações são armazenadas em *MIBs* descritas através da linguagem *ASN*.1.

Um *framework* que utilize o protocolo *CMIP* tende a usar a modelagem da orientação a objetos, que *encapsula* as operações associadas a uma estrutura de dados na própria estrutura. Aqui um agente tem um servidor de objetos gerenciados que pode executar operações de gerênciamento nas variáveis relacionadas a um nó gerenciado. Se este agente for executado em outra máquina (separadamente ao resto do código de gerência), tem-se então o gerenciamento distribuído de rede.

Os serviços oferecidos pelo *CMISE* ao protocolo *CMIP* podem ser confirmados ou não confirmados. A tabela 4.1 mostra a relação entre os serviços *CMISE* e as classes de operação do protocolo CMIP. Estes serviços serão mapeados em operações aplicadas sobre os objetos gerenciados, que representam os recursos da rede a serem gerenciados.

SERVIÇO	TIPO	CLASSE DE OPERAÇÃO
M-EVENT-REPORT	confirmado/não-confirmado	2 ou 1/5
M-GET	confirmado	2 ou 1
M-CANCEL-GET	confirmado	2 ou 1
M-SET	confirmado/não-confirmado	2 ou 1/5
M-ACTION	confirmado/não-confirmado	2 ou 1/5
M-CREATE	confirmado	2 ou 1
M-DELETE	confirmado	2 ou 1

Tabela 4.1: CMISE e Suas Classes de Operação.

Os serviços oferecidos pelo *CMISE* e usados pelas aplicações de gerenciamento e para o informe de notificações, são:

- *M-EVENT-REPORT*: Reporta um evento de um objeto gerenciado;
- *M-GET*: Solicita a busca de informações de gerenciamento;
- *M-CANCEL-GET*: Solicita o cancelamento de um serviço *M-GET* previamente requisitado e ainda pendente;
- *M-SET:* Solicita a modificação da informação de gerenciamento;
- *M-ACTION*: Solicita a execução de uma ou mais ações sobre os objetos gerenciados;
- *M-CREATE*: Solicita a criação de uma instância de um objeto gerenciado;
- *M-DELETE*: Solicita a "deleção" de uma ou mais instâncias de objetos gerenciados.

Definem-se as seguintes classes de operação, das quais algumas estão presentes na Tabela 4.1:

- Síncrona relatando sucesso ou falha;
- Assíncrona relatando sucesso ou falha;
- Assíncrona relatando somente falhas (erro);
- Assíncrona relatando somente sucesso;
- Assíncrona com o resultado não relatado.

E definem-se também as seguintes classes de associação:

- Somente a entidade que iniciou a associação pode invocar operações;
- Somente a entidade que responde na associação pode invocar operações;
- Ambas as entidades, a que iniciou e a que responde em uma associação podem invocar operações.

4.5.4 O Protocolo *CMIP* x *SNMP*

A seguir serão apresentadas algumas diferenças entre os protocolos SNMP e *CMIP:*

Em primeiro lugar, o CMIP pode ser utilizado sobre a camada de transporte do modelo *OSI* ou sobre o *TCP/IP*, abordagem esta conhecida como *CMOT* (*CMIP Over TCP/IP*). Ele é baseado em conceitos de orientação a objeto e sua estrutura de dados é definida usando a especificação denominada *X.500*, que também é descrita em *ASN.1*.

As suas primitivas de serviço são em maior número e muito mais complexas que as definidas para o *SNMP*.

Além disto, o *CMIP* possui recursos de criptografia e autenticação, que permitem uma garantia relativa de segurança das informações de gerenciamento.

Conforme podemos observar, o *CMIP* é significativamente mais complexo do que o SNMP. Entretanto, muitas das vantagens oferecidas por ele serviram de base para inovações propostas nas versões posteriores a 1 do *SNMP*. Contudo, sua complexidade praticamente o inviabiliza como protocolo para gerenciamento de redes de Computadores, o que restringe sua utilização a telecomunicações, onde os requisitos de velocidade de processamento não são tão críticos. Por isto, temos o *SNMP* implementado na Internet e o CMIP, em algumas aplicações de telefonia, por exemplo. Resumindo, temos o quadro comparativo mostrado a seguir (Tabela 4.2)

SNMP	CMIP
Ambiente Internet	Ambiente OSI
Simples	Complexo
Primitivas Limitadas	Primitivas Expressivas
MIB's tabelas	MIB's classes

Tabela 4.2: Comparativo SNMP X CMIP

4.5.5 Remote Monitoring (RMON)

Tecnologias com o padrão *RMON* (*Remote MONitoring*) podem fornecer informações ao gerente através de estatísticas e dados em tempo-real que serão críticos para companhias com o intuito de realizar o controle proativo de suas redes. Alcançar esse objetivo requer emprego da tecnologia RMON nos dispositivos de rede bem como as melhorias nos protocolos *RMON* e *SNMP* para suportar serviços escaláveis e integração com outros aplicações de gerência.

O protocolo *SNMP*, sozinho, não é adequado para ambientes de redes corporativas constituídas de diversas redes locais conectadas através de outra de longa distância. Esses enlaces de rede de longa distância, por operarem a taxas de transmissão inferiores as *LAN* que a interconectam, passam a ter grande parte da sua banda de transmissão ocupada para informações de gerenciamento. Uma solução encontrada para dirimir este problema foi o *Remote MONitoring* (*RMON*).

O protocolo *RMON* (WALDBUSSER, 1991), (CARVALHO, 1997), oferece suporte à implementação de um sistema de gerenciamento distribuído. Nele, fica atribuída aos diferentes elementos, tais como estações de trabalho, *hubs*, *switches* ou roteadores, das redes locais remotas a função de monitor remoto. Cada elemento *RMON* tem, então, como tarefas, coletar, analisar, tratar e filtrar informações de gerenciamento da rede e apenas notificar à estação gerente os eventos significativos e situações de erro. No caso de existirem múltiplos gerentes, cada elemento *RMON* deve determinar quais informações de gerenciamento devem ser encaminhados para cada gerente.
Sendo assim, os objetivos do protocolo *RMON* são (CARVALHO 1997):

- Reduzir a quantidade de informações trocadas entre a rede local gerenciada e a estação gerente conectada a uma rede local remota.
- Possibilitar o gerenciamento contínuo de segmentos de redes locais, mesmo quando a comunicação entre o elemento *RMON* e a estação gerente estiver, temporariamente, interrompida.
- Permitir o gerenciamento pró-ativo da rede, diagnosticando e registrando eventos que possibilitem detectar o mau funcionamento e prever falhas que interrompam sua operação.
- Detectar, registrar e informar à estação gerente condições de erro e eventos significativos da rede.

- Enviar informações de gerenciamento para múltiplas estações gerentes, permitindo, no caso de situações críticas de operação da rede gerenciada, que a causa da falha ou do mau funcionamento da rede possa ser diagnosticada a partir de mais de uma estação gerente.

4.5.5.1 Motivação

A motivação original para *RMON* foi criada a partir da necessidade de coletar informações sobre múltiplos segmentos de rede *LAN* em uma base contínua. Com a complexidade da gerência das *LANs* aumentou, enviar pessoal dedicado com analisadores de rede portáteis para cada segmento se tornou um gasto proibitivo. Além disso, os monitores de rede eram proprietários, exigindo múltiplos consoles de gerência para que fosse possível suportar todos os fornecedores - uma confusão total.

A primeira versão do *RMON MIB* usa *SNMP*, o mais popular protocolo de gerência, para monitorar as operações básicas da *Ethernet* e *Token Ring*. O primeiro *RMON*, *RFC* 1757, define dois grupos específicos *Ethernet* e sete outros grupos que se aplicam tanto a *Ethernet* como a *Token Ring*.

O segundo padrão, RFC 1513, define extensões *Token Ring* para *RMON*. Com esses dois primeiros padrões, *RMON* criou uma base para futuras extensões da *MIB*. Hoje, o padrão inclui 13 grupos *Ethernet* e *Token Ring* que contribuem para uma meta comum: permitir monitoramento de todas *LANs* com agentes "baseados em *RMON*" independentes de qualquer marca ou vendedor.

4.5.5.2 Filosofia

RMON possibilita às implementações de monitoramento de redes proverem resultados mais robustos e precisos do que padrões teóricos. Em 1990, por exemplo, o grupo de trabalho examinou analisadores *LAN* e produtos de monitoramento bem conhecidos no mercado usando-os como a base para o desenvolvimento do padrão.

Esta confiança nas tecnologias "*tried-and-true*" como opostos dos padrões teóricos continua até hoje. A maioria das revisões e melhorias sendo consideradas pelo *RMON2* já são implementadas em produtos vendidos atualmente com extensões proprietárias da *MIB*.

Para ser considerado "base em *RMON*" um produto deve implementar uma versão completa de ao menos um dos 13 grupos definidos nas *MIBs Token Ring* e *Ethernet*. Vendedores podem implementar extensões adicionais sem prejudicar sua base *RMON* e, até certo ponto, isso é encorajado para que se consiga melhorias e atualizações do padrão. *RMON* também cobre o banco de dados usado para armazenar as informações (um agente local que faz o processamento baseado nos atributos dos elementos do banco) e as capacidades de controle pelo console que se comunicam com o agente local usando *SNMP*.

4.5.5.3 Benefícios

Os maiores benefícios proporcionados pelo padrão *RMON* são descritos abaixo:

- Análise e Monitoração Poderosas: *RMON* provê um poderoso mecanismo para monitoramento da rede e análise dos dados para gerenciar aplicações que coletam informações como estatísticas de segmento e tendências, análise do padrão de tráfego no nodo, largura de banda usada e alarmes. Os dados providos por agentes RMON permitem que vendedores criem aplicações de gerência que reduzam o trabalho da administração da rede e os custos com a solução de problemas.
- Tendências em Segmentos Locais: *RMON* provê uma história para tendências em segmentos locais. Esta habilidade faz que o gerente de rede procure por estatísticas chaves para identificar as tendências potenciais.
- Funções Tradicionais de Decodificação do Protocolo: *RMON* provê um mecanismo para análise detalhada das sete camadas de protocolos de segmentos remotos sem o custo de um analisador de protocolo dedicado. A maioria dos problemas das aplicações cliente/servidor podem ser diagnosticados usando o pacote de captura do *RMON* e grupos de filtro.
- Monitoramento Centralizado de *Sites* Remotos: *RMON* provê os dados necessários para monitorar *sites* remotos de uma localização central. Isto significa que gerentes de rede podem unir e analisar dados de *sites* remotos e solucionar problemas comuns - evitando o custo de uma viagem para a localização remota ou de técnicos dedicados no local.

- Interoperabilidade de Fornecedores: Com *RMON*, uma variedade de diferentes estações de gerência e agentes, fornecidos por diversos vendedores, podem se comunicar entre eles através da mesma rede.
- Criação de Eventos quando liminares predefinidos são atingidos: Muitas estações de gerenciamento *SNMP* periodicamente enviam pacotes *echo* para checar o status de cada dispositivo. Como o número de segmentos *LAN* e de *LANs* remotas crescem, torna-se necessário projetar uma arquitetura distribuída para impedir que a gerência dos dados não ocupe uma banda excessiva. *RMON* tem limiares predefinidos que podem remotamente monitorar dispositivos de rede e enviar exceções quando estes dispositivos não estão mais em um limite de operação aceitável.

4.5.5.4 Objetivos

Os dispositivos de monitoramento remoto de rede (dispositivos *RMON*) são instrumentos que existem com o propósito de gerenciar uma rede. Muitas vezes estes dispositivos são dedicados e devotam significativos recursos internos para o propósito de gerenciamento de rede.

Uma organização pode dispor de diversos destes dispositivos, um por segmento de rede, para gerenciar sua rede Internet.

A especificação *RMON* é uma definição de uma *MIB*. O objetivo, contudo, é definir padrões de monitoração e interfaces para a comunicação entre agentes/gerentes *SNMP*.

Os objetivos da implementação de um dispositivo *RMON* são os que se seguem (WALDBUSSER 1991):

- Operação *Off-Line* - esta é a condição em que a estação gerenciadora não está em contato constante com o dispositivo *RMON*. Presta-se para desenhar redes de baixo custo de comunicação (redes por acesso discado ou conexões com *World Area Networks - WAN)* ou para acidentes onde as falhas na rede afetam a comunicação entre a estação gerenciadora e os dispositivos *RMON*. Desta forma, o monitor é configurado para coletar estatísticas e fazer diagnósticos continuamente, mesmo se a conexão com o gerente não for possível ou

apresentar falhas. O monitor pode também notificar a estação de gerenciamento se eventos excepcionais ocorrerem.

- Monitoramento Preemptivo - se o monitor tiver recursos disponíveis, estes podem ser usados para executar diagnósticos continuamente e para analisar a performance da rede. Quando uma falha ocorrer, o monitor pode notificar a estação de gerenciamento e armazenar o histórico estatístico referente à falha. Posteriormente este histórico pode ser enviado à estação de gerenciamento com o objetivo de se fazer um estudo mais profundo e permitir a detecção e reparo da falha.
- Detecção de Problemas e Geração de Relatórios - o monitor pode ser configurado para reconhecer certas situações, mais notadamente condições de erro e checar continuamente por elas. Quando uma destas situações ocorrer, o monitor pode registrá-la e reportá-la à estação de gerenciamento.
- Análise de Dados - por ser um dispositivo dedicado exclusivamente ao gerenciamento de rede e por estar localizado diretamente no segmento monitorado da rede, os dispositivos *RMON* podem fazer uma análise significativa dos dados que coletam. Por exemplo, estes dispositivos podem determinar qual *host* gera maior tráfego ou mais erros na rede.
- Múltiplos Gerentes - uma configuração de rede pode ter mais de uma estação gerente para dar mais confiabilidade, executar funções diferentes e prover capacidades de gerência para unidades diferentes dentro da organização.

4.5.5.5 Controle de Monitores Remotos

Um monitor remoto (dispositivo *RMON*) pode ser configurado como uma função disponível em um sistema ou como um dispositivo dedicado. Configurado como dispositivo dedicado, o monitor é capaz de efetuar operações mais complexas.

A definição da *MIB RMON* contém características que suportam controle extensivo da estação de gerenciamento. Estas características dividem-se em duas categorias:

- Configuração: Tipicamente, um monitor remoto necessitará ser configurado para coletar dados. A configuração dita o tipo e a forma de dados para serem coletados. A *MIB* é organizada em grupos funcionais. Cada grupo terá uma ou

mais tabelas de controle e uma ou mais tabelas de dados. Uma tabela de controle contém parâmetros que descrevem o dado na tabela de dados, que é somente para leitura. Assim, a estação gerente envia os parâmetros apropriados para configurar o monitor remoto para coletar os dados desejados. Os parâmetros são configurados pela adição de um novo registro na tabela ou alterando uma existente. Desse modo, funções para serem executadas pelo monitor são definidas e implementadas na tabela. Por exemplo, uma tabela controle pode conter objetos que especifiquem a origem dos dados coletados, tipos de dados.Para modificar qualquer parâmetro na tabela de controle é necessário primeiro invalidar a entrada. Isto causa a retirada daquela entrada e de todos os registros associados em tabelas de dados. A estação gerente pode então criar um novo registro controle com os parâmetros modificados. O mesmo mecanismo é usado para apenas desabilitar uma coleção de dados. Quando um registro da tabela de controle é apagado, os registros das tabelas de dados associadas são apagados, e os recursos usados pelos registros são recuperados.

- Invocação de Ação: O *SNMP* providencia mecanismos não específicos para emitir um comando para um agente executar uma ação. O *SNMP* tem apenas capacidade de ler valores de objetos e "setar" valores de objetos com visão *MIB*. Contudo, isto é possível para usar o conjunto de operações *SNMP* para emitir um comando. Um objeto pode ser usado para representar um comando, assim que uma ação específica é alcançada se o objeto é "setado" para um valor específico. O número desses objetos é incluído na *MIB RMON*. Em geral, estes objetos representam estados, e uma ação é executada se a estação gerente trocar o estado (pela troca do valor do objeto).

4.5.5.6 Múltiplos Gerentes

Conforme já citado, um agente *RMON* pode ser submetido a múltiplos gerentes. A qualquer tempo acessos concorrentes são permitidos para um recurso disponível em um agente. Esta é uma característica potencial para conflitos e pode gerar resultados inesperados. No caso de agentes *RMON* compartilhados, as seguintes dificuldades podem surgir:

- Requisições concorrentes podem exceder a capacidade do monitor para fornecer estes recursos.
- Uma estação gerente pode capturar e ocupar recursos de monitor por um longo período de tempo, prevenindo seu uso por outras funções gerente desejadas por outras estações gerentes.
- Recursos podem ser designados para uma estação gerente onde ocorreu uma falha e os recursos não foram liberados.

Para proceder com esses problemas, uma combinação de características de resolução e prevenção é necessária. Ela pretende que uma simples característica na *MIB RMON* suporte estes requerimentos. Associado com cada tabela de controle está um objeto do tipo registro que identifica o proprietário de um registro particular da tabela e de funções associadas. O rótulo proprietário pode ser usado das seguintes formas:

- Uma estação gerente pode reconhecer recursos próprios.
- Um operador pode identificar a estação gerente que seja proprietária de um recurso em particular ou função e negociar para serem acessíveis para todos.
- Um operador pode ter autoridade para liberar recursos que outro operador tenha reservado.

A especificação sugere que o rótulo proprietário contenha um ou mais desses atributos: endereço *IP*, nome da estação gerente, nome do gerente de rede, localização ou telefone.

Apesar do rótulo ser proveitoso, é importante ressaltar que o rótulo não tem ação como uma senha ou mecanismo de controle de acesso.

Se múltiplos gerentes de rede têm acesso à tabela de controle, uma maior eficiência pode ser alcançada pelo compartilhamento. Quando uma estação gerente quer utilizar uma certa função no monitor, ela precisa verificar a tabela de controle relevante para ver que função, tem sido definida por outra estação gerente. Neste caso, a estação gerente pode compartilhar a função simplesmente observando os registros de dados *read-only* associados com o registro de controle. Contudo, a estação gerente que seja proprietária de uma tabela de controle pode modificar ou apagar aquele registro a qualquer hora.

Freqüentemente, um monitor será configurado com um conjunto padrão de funções que serão “setadas” quando ele for inicializado. Os registros de controle que definem estas funções são propriedades do monitor. Por convenção, cada rótulo relevante é configurado com uma cadeia de nome "monitor".

4.5.5.7 *RMON1* e *RMON2*

Dois padrões básicos de protocolo *RMON* são especificados: *RMON1* e *RMON2*, funcionalmente complementares.

O *RMON1* opera somente na camada *Media* Access Control (*MAC*) e, segundo (CARVALHO,1997) oferece recursos ao administrador da rede para monitorar o tráfego e coletar informações estatísticas da operação de um segmento de rede local, além de realizar o diagnóstico remoto de falhas e erros ocorridos no segmento de rede a partir de funcionalidades de um analisador de protocolo suportadas pelo correspondente elemento *RMON*.

O *RMON2*, por sua vez opera no nível da camada de rede e camadas superiores, complementando, portanto o *RMON1*, possibilitando coletar informações estatísticas e monitorar a comunicação fim-a-fim e o tráfego gerado por diferentes tipos de aplicação. A figura 4.1 ilustra esta diferença.

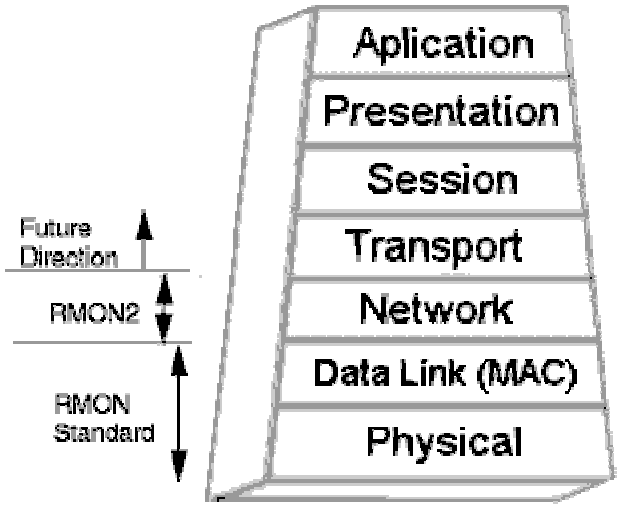

Figura 4.1: Diferenças na Atuação da *RMON1* e 2

4.5.5.8 Limitações Superadas pelo *RMON2*

O *RMON* especifica os dados a serem coletados de segmentos remotos, possibilitando uma visão agregada da rede. Mesmo sendo *RMON* um passo significante no monitoramento remoto de *LANs*, o uso da versão 1 revelou algumas desvantagens - fraquezas que foram consertadas em *RMON2*. Alguns exemplos:

- Monitoramento de tráfego ampliado: Análises tradicionais baseadas em *RMON* oferecem ao gerente de rede um poderoso sistema de correção de erros e uma capacidade de monitorar um segmento *LAN* no nível *MAC*, mas não conseguem enxergar o tráfego quando este ultrapassa as barreiras do roteamento. Com *RMON2*, gerentes de rede serão capazes de ver o tráfego desde a camada 3 até a 7 do modelo *OSI* . Isto significa que gerentes são capazes de ver como flui o tráfego de rede em cada um dos 7 níveis do modelo *OSI*, uma capacidade que terá um impacto significante no controle e resolução de erros em ambientes cliente/servidor. A Figura 4.2 mostra o nível de visibilidade que *RMON* e *RMON2* provêem dentro de um segmento *LAN* ou de uma rede em cada uma das camadas do *OSI*.

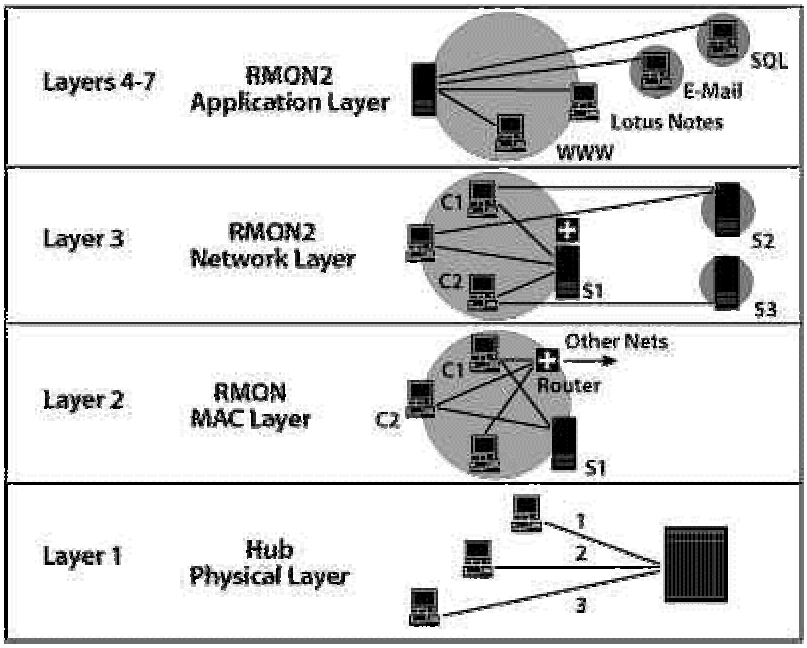

Figura 4.2 Nível de Visibilidade *RMON* e *RMON2*

- Maior confiança no fornecedor: Atualmente, para que algum produto seja "baseado em *RMON*" ele necessita apenas implementar um dos grupos RMON. Com isso, as capacidades *RMON* podem variar drasticamente de um fornecedor para outro, dependendo do número de grupos *RMON MIB* que o produto suporta. *RMON2* requer que os produtos a implementem mais amplamente para que possa ser considerado "baseado em *RMON2*".
- Correlação com a camada física: monitores baseados em *RMON* coletam estatísticas no nível 2, a camada *Data Link* do modelo *OSI*. Esta camada se concentra primordialmente com a formação dos pacotes: endereçamento e entrega. Embora esse tipo de informação seja muito útil para a resolução de problemas, não se tem nenhuma visão do nível físico. *RMON2* tem a habilidade de relacionar informação de porta ou cabeamento na estação final, aumentando incrivelmente a visibilidade da rede.
- História completa: com *RMON2*, qualquer objeto *MIB* pode ser localmente rastreado e gravado em um *log* histórico.

Os benefícios do *RMON* estendem a monitoração aos mais altos níveis das camadas do protocolo. Enquanto *RMON* somente provê estatísticas de tráfego na

camada *MAC* do protocolo, *RMON2* provê um discernimento das estatísticas de tráfego bem maior, especificando o protocolo e as aplicações que geraram aquele tráfego. Este conhecimento é de vital importância nos dias de hoje. Estender a visão do segmento RMON para uma mais ampla fornecida pelo *RMON2* faz com que gerentes de rede tenham mais informações e maior conhecimento para gerenciar, corrigir os erros e monitorar redes cada vez mais complexas. Como resultado, *RMON2* permitirá a análise cliente/servidor de uma nova classe inteira de aplicações; monitoramento e projeto inter-rede; e registro nas camadas de rede e aplicação.

4.5.6 Remote Monitoring Management Information Base (*RMON-MIB*)

A implementação das funções do protocolo *RMON* somente é viável mediante o suporte de uma base de dados de gerenciamento, a *RMON-MIB*, associada a cada elemento *RMON* da rede.

4.5.6.1 Grupos da *RMON1-MIB*

Para a *RMON1*-MIB, foram especificados dez grupos básicos de variáveis, que incluem(CARVALHO,1997):

- Estatístico - mantém estatísticas de utilização, tráfego e taxas de erros ocorridos em um segmento de rede.
- Histórico - permite controlar o processo de amostragem (definição dos intervalos de amostragem) de informações do grupo estatístico e registrar tais informações, empregadas na análise do comportamento de uma rede e que oferecem subsídios para um gerenciamento pró-ativo.
- Alarmes - possibilitam estabelecer condições limites de operação de uma rede que deve provocar a geração de alarmes.
- *Hosts* - contêm informações relativas ao tráfego gerado e recebido pelos hosts conectados através da referida rede.

- Classificação de n *hosts* (*top* n *hosts*) - permite classificar os *hosts* segundo critérios pré-definidos como, por exemplo, determinar quais os *hosts* conectados através da rede que geram maior tráfego em um dado período do dia.
- Matriz - contém informações de utilização da rede e taxa de erros na forma de matriz, associando pares de endereços MAC de elementos de rede.
- Filtro - define condições associadas a pacotes trafegados pela rede, que uma vez satisfeitas implicam captura de tais pacotes pelo elemento *RMON* ou no registro de estatísticas baseadas nos mesmos.
- Captura de Pacotes - determina como devem ser capturados os dados dos pacotes trafegados pela rede, a serem enviados aos gerentes. Como *default*, são capturados os cem primeiros *bytes* dos pacotes filtrados pelo elemento *RMON*.
- Evento - define todos os eventos que implicam a criação de registros (*logs*) de eventos e o envio de informações pertinentes do elemento *RMON* aos gerentes.

A implementação de todos os grupos é opcional, embora exista uma relação de dependência entre alguns deles, como é o caso do grupo de "classificação de n *hosts*" em relação ao grupo de *hosts*.

4.5.6.2 Grupos da *RMON2*-MIB

Para a *RMON2-MIB*, foram especificados, também, dez grupos básicos de variáveis, que incluem (CARVALHO,1997):

- Diretório de Protocolo - especifica uma lista dos protocolos - de rede, transporte e de camadas superiores - que o elemento *RMON* tem a capacidade de monitorar, sendo possível incluir, remover ou configurar entradas dessa lista.
- Distribuição de Protocolo - contém informações relativas ao número de bytes ou pacotes referentes a diferentes protocolos transferidos através de um determinado segmento de rede.
- Mapeamento de Endereços - relaciona endereços *MAC* e endereços de rede - ou endereços *IP*.
- Camada de rede do *Host* - contabiliza o tráfego gerado e recebido por um *host*, cujo endereço de rede é conhecido pelo *RMON*.

- Matriz da Camada de rede - contabiliza o tráfego existente entre um par de hosts, cujos endereços de rede são conhecidos pelo *RMON*.
- Camada de Aplicação do *Host* - contabiliza o tráfego, relativo a determinado protocolo, gerado e recebido por um *host*, cujo endereço de rede é conhecido pelo *RMON*.
- Matriz da Camada de Aplicação - contabiliza o tráfego, relativo a um determinado protocolo, existente entre um par de *hosts*, cujos endereços de rede são conhecidos pelo *RMON*.
- Histórico do Usuário - contém informações específicas de um usuário relativo ao tráfego gerado, período e intervalos de amostragem, entre outras informações.
- Configuração do *Probe* - contém a configuração dos parâmetros de operação do *RMON*.
- Conformidade *RMON* - define os requisitos de conformidade da *RMON-MIB*.

4.5.6.3 Estrutura da RMON2-MIB

Os objetivos definidos em RMON2-MIB são arranjados nos seguintes grupos:

- *Protocol directory* (*protocolDir*)
- *Protocol distribuition* (*protocolDis*)
- *Address mapping* (*addressMap*)
- *Network layer host* (*nlHost*)
- *Network layer matrix* (*nlMatrix*)
- *Application layer host* (*alHost*)
- *Application layer matrix* (*alMatrix*)
- *User history* (*usrHistory*)
- *Probe configuration* (*probeConfig*)

A figura 4.3 mostra os objetos definidos em RMON e RMON2

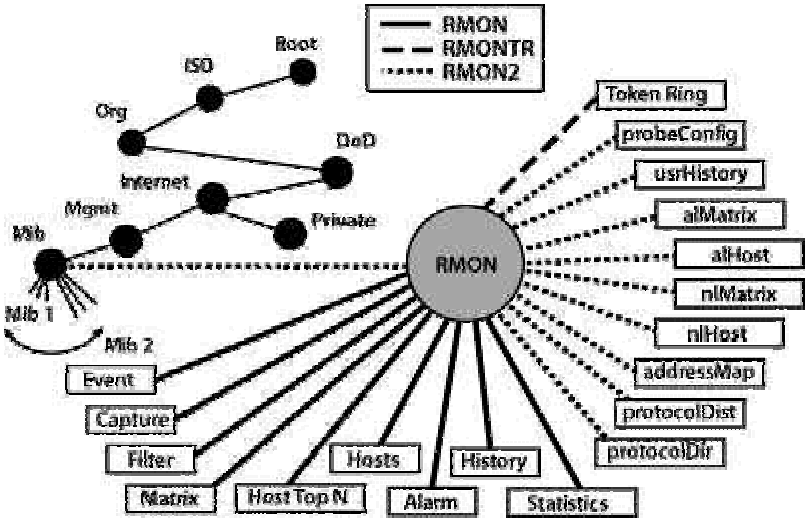

Figura 4.3: Objetos RMONe RMON2

Os 9 grupos apresentados formam a base de toda *RMON2-MIB*. Se um dispositivo de gerenciamento remoto implementa um grupo, então ele deve implementar todos os objetos naquele grupo. Por exemplo, um agente que implementa o grupo *network layer matrix* deve implementar o *nlMatrixSDTable* e o *nlMatrixDSTable*.

Implementações desta MIB devem também implementar os grupos *system* e interfaces da MIB II.

Pode-se, sem prejuízo nenhum, implementar outros grupos que sejam necessários.

Os grupos de *RMON2-MIB* são definidos para prover um significado no assinalamento dos identificadores dos objetos e para fornecer um método para que os agentes saibam quais objetos eles devem implementar.

Na tabela 4.3 temos uma comparação da *RMON2-MIB* comparada com as outras versões anteriores.

Data	RMON2	RMON	MIB II	Repeater MIB	Bridge MIB	Host MIB
IP, TCP, UDP, SNMP statistics						
Host lab Counts						
Host File System Information						
Link Testing						
Spanning Tree Performance						
Configurable Statistics						
Network Segment Statistics						
Historical Segment Statistics						
History for Any Variable						
Port/MAC Address Mapping, Port Statistics						
Host Statistics by MAC Address, Top N Studies						
Traffic Matrix by Host (MAC) Address						
MAC/Network Address Translation						
Network Layer Statistics by Network Address						
Traffic Matrix by Network Address, Top N Studies						
Application Layer Statistics by Application Address						
Traffic Matrix by Application Address, Top N Studies						
Packet Capture and Filter for Analysis						
Protocol Distribution Statistics						
Thresholds for Any Variable						
Event Logging						

Tabela 4.3: Comparativo RMON2-MIB

4.5.6.4 Melhorias Implementadas pelo RMON2-MIB

A RMON2-MIB não pára na tarefa de definir novos grupos: ela contém melhoramentos nas tabelas definidas em RMON-MIB. Esses melhoramentos incluem:

- Adiciona as convenções DroppedFrame e LastCreateTime para cada tabela definida em RMON-MIB
- Amplia a tabela de filtro RMON com um mecanismo que permite filtragem baseada em um offset do início de um protocolo, mesmo se o cabeçalho do protocolo tiver tamanho variável.
- Amplia o filtro RMON e o status de captura com bits que são definidos na Tabela 4.4:

Bit	Definição
6	Para redes WAN, este bit é setado para pacotes vindos de uma direção e zerado para pacotes vindos de outra.Não é importante qual bit represente qual direção, apenas que seja consistente com todos os pacotes recebidos pelo agente.Entretanto se o agente sabe qual é o fim "local" e qual fim é "rede", o bit deve ser setado para pacotes do lado "local" e zerado para pacotes do lado "rede"
7	Para qualquer rede (não apenas Wan), este bit é setado em pacotes com um erro da camada física. Este bit deve ser setado em conjunto com outros bits que têm a mesma função mas para meios específicos.
8	Para qualquer rede, este bit é setado para pacotes pequenos demais para o meio. Este bit deve ser setado em conjunto com outros bits que têm a mesma função mas meios específicos.
9	Para qualquer rede, este bit é setado em pacotes longos demais para o meio. Deve ser setado em conjunto com outros bits que têm a mesma função mas para meios específicos.

Tabela 4.4:Captura dos Bits RMON2-MIB

4.5.7 Management Information Base (MIB)

Antes de definir o que é uma *MIB*, será introduzido o conceito de objetos gerenciados.

Um objeto gerenciado é a visão abstrata de um recurso real do sistema. Assim, todos os recursos da rede que devem ser gerenciados são modelados, e as estruturas de dados resultantes são os objetos gerenciados. Os objetos gerenciados podem ter permissões para serem lidos ou alterados, sendo que cada leitura representará o estado real do recurso e, cada alteração também será refletida no próprio recurso.

Dessa forma, a *MIB* (*Management Information Base*) é o conjunto dos objetos gerenciados, que procura abranger todas as informações necessárias para a gerência da rede, possibilitando assim, a automatização de grande parte das tarefas de gerência.

A especificação *MIB* define as variáveis necessárias à monitoração e controle de vários componentes em redes Internet. Nem todos os grupos de variáveis definidas pela especificação *MIB* são obrigatórios para todos os componentes de redes Internet.

Em uma implementação *SNMP*/*RMON* os objetos gerênciados são acessados através de um banco de dados virtual, chamado *MIB*. Os objetos de uma *MIB* são definidos usando o padrão *ASN. 1*.

Uma *MIB* pode ser descrita como uma árvore abstrata com um *root* anônimo. Os níveis da árvore são compostos pelos itens de dados individuais. Identificadores de objetos (*ID*) identificam ou nomeiam unicamente os objetos da *MIB* na árvore. Identificadores de objetos são como números de telefones, eles são organizados hierarquicamente com um específico dígito associado por diferentes organizações.

O *RFC* 1066 apresentou a primeira versão da *MIB* para uso com o protocolo *TCP/IP*, a *MIB-I*. Este padrão explicou e definiu a base de informação necessária para monitorar e controlar redes baseadas no protocolo *TCP/IP*. O *RFC* 1066 foi aceito pela *IAB* (*Internet Activities Board*) como padrão no *RFC* 1156.

O *RFC* 1158 propôs uma segunda *MIB*, a *MIB-II*, para uso com o protocolo *TCP*/*IP*, sendo aceita e formalizada como padrão no *RFC* 1213. *A MIB-II* expandiu a base de informações definidas na *MIB-I.*

A *MIB* e a *MIB-II* padrão para a Internet, contém 171 objetos. Estes objetos são agrupados por protocolos (incluindo *TCP*, *IP*, *UDP*, *SNMP*, e outros) e outras categorias, incluindo "sistemas" e "interfaces".

A árvore *MIB* é extensiva por força das ramificações experimentais e privadas. Fabricantes podem definir suas próprias ramificações para definir instâncias em seus produtos. Uma estrutura básica da *MIB* é mostrada na Figura 4.4.

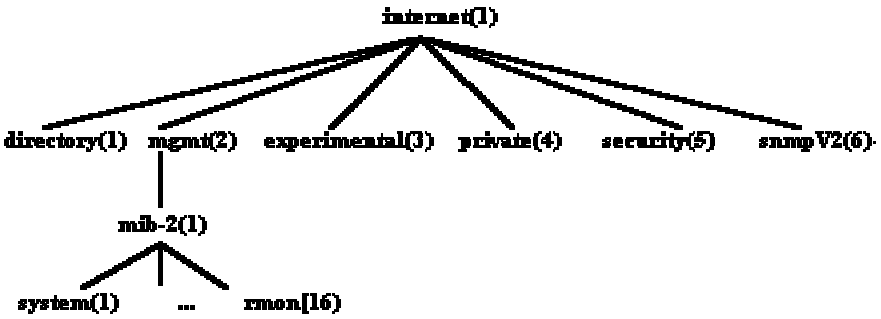

Figura 4.4: Estrutura Básica de uma MIB

4.5.7.1 Definições *SMI*

A *Structure* of *Management Information* (*SMI*) especifica que todo objeto gerenciado deve ter um nome, uma sintaxe e um código. O nome é o identificador de objeto, discutido no tópico anterior. A sintaxe define o tipo de dados dos objetos (por exemplo, inteiro ou string). Um subconjunto da definição *ASN.1* é usado para a sintaxe *SMI*. O código descreve como a informação associada a um objeto gerenciado é formatada como uma série de itens de dados para transmissão na rede. Outra especificação *ISO*, chamada *Base Encoding Rules* (*BER*), detalha os códigos *SMI*. Os tipos de dados *SMI* são divididos em três categorias:

- Tipo simples
- Tipo de grandes aplicações
- Tipo construtor simples.

Os tipos simples incluem quatro tipos *ASN.1* primitivos:

- Inteiros - valores negativos ou positivos de todos os números, inclusive o zero.
- Cadeia de octetos - seqüência ordenada de zero ou mais octetos.
- Identificadores de objetos - conjunto de todos os identificadores de objetos alocados de acordo com as regras especificadas pelo *ASN.1*.

Tipos de dados de grandes aplicações referem-se aos tipos de dados especiais definidos pelo *SMI*:

- Endereços de rede - representa um endereço de uma família particular de protocolos.
- Contadores - inteiros não negativos são incrementados de um em um até atingirem um valor máximo, quando eles são “resetados” e voltam a zero. O número total de bits recebidos em uma interface é um exemplo de contador.
- Medidas - inteiros não negativos que são incrementados ou decrementados, porém atrelados a um valor máximo. O tamanho da fila de saída de pacotes é um exemplo.
- Checagem de tempo - o tempo de um evento. O tempo necessário para uma interface chegar ao estado corrente é um exemplo.

- Opaco - representa uma codificação arbitrária. Este tipo de dados é usado para passar uma cadeia de informações arbitrárias que não está de acordo com a tipagem de dados usada no *SMI*.
- Inteiros - representa uma informação com valores inteiros sinalizados. Este tipo de dados redefine o tipo de dados simples "inteiro" do *ASN.1*, que tem uma precisão arbitrária no *ASN.1* porém uma precisão determinada no *SMI*.
- Inteiros sem sinal - representa uma informação com valores inteiros não sinalizados. Ele é útil quando os valores são sempre não negativos. Este tipo de dados redefine o tipo de dados simples "inteiro" do *ASN.1*, que tem uma precisão arbitrária no *ASN.1* porém uma precisão determinada no *SMI*.

O tipo construtor simples inclui dois tipos *ASN.1* que definem múltiplos objetos em tabelas e listas:

- Linha - referência a uma linha de uma tabela. Cada elemento de uma linha pode ser um tipo simples ou um tipo de grandes aplicações.
- Tabela - referência a uma tabela com zero ou mais linhas. Cada linha pode ter um número qualquer de colunas.

A especificação *BER*, definida por [8825,*Specification of Basic Encoding Rules for ASN.1*] citado por (CISCO,1996), permite que máquinas diferentes troquem informações de gerenciamento especificando a posição de cada bit nos octetos transmitidos e a estrutura dos bits. A estrutura de bits é definida pela descrição do tipo de dados, tamanho e valor.

4.5.7.2 *MIB* da *OSI*

O padrão *OSI* define três modelos para gerência de redes: o modelo organizacional, o modelo informacional e o modelo funcional. O modelo organizacional descreve a forma pela qual a gerência pode ser distribuída entre domínios e sistemas dentro de um domínio. O modelo funcional descreve as áreas funcionais e seus relacionamentos. Já o modelo informacional provê a base para a definição de objetos gerenciados e suas relações, classes atributos, ações e nomes.

Na definição de objetos gerenciados é utilizada a orientação a objetos. Objetos com características semelhantes são agrupados em classes de objetos. Uma classe pode ser uma subclasse de outra, e a primeira herda todas as propriedades da segunda. Uma classe é definida pelos atributos da classe, pelas ações que podem ser invocadas, pelos eventos que podem ser relatados, pela subclasse à qual ela deriva e pela superclasse na qual ela está contida.

Para a definição dos objetos gerenciados deve-se considerar três hierarquias: hierarquia de herança, de nomeação e de registros usados na caracterização e identificação de objetos gerenciados. A seguir serão descritas cada uma das hierarquias mencionadas acima.

- Hierarquia de Herança - Também denominada hierarquia de classe, tem como objetivo facilitar a modelagem dos objetos, através da utilização do paradigma da orientação a objetos. Assim podem ser definidas classes, superclasses, subclasses. Trata-se de uma ferramenta para uma melhor definição de classes.
- Hierarquia de Nomeação - A hierarquia de nomeação, também chamada hierarquia de *containment*, descreve a relação de "estar contido em" aplicado aos objetos. Um objeto gerenciado está contido dentro de um e somente um objeto gerenciado.Um objeto gerenciado existe somente se o objeto que o contém existir, e dependendo da definição, um objeto só pode ser removido se aqueles que lhe pertencerem forem removidos primeiro.
- Hierarquia de Registro - A hierarquia de registro é usada para identificar de maneira universal os objetos, independentemente das hierarquias de heranças e nomeação. Esta hierarquia é especificada segundo regras estabelecidas pela notação ASN.1

Assim, cada objeto é identificado por uma seqüência de números, correspondente aos nós percorridos desde a raiz, até o objeto em questão.

4.5.7.3 Comparação entre a *MIB* da *OSI* e a *MIB* da Internet

As *MIB's* da *ISO* e da Internet são modeladas através de técnicas de programação por objeto. Dentro deste contexto, os recursos a serem gerenciados são representados através de objetos gerenciados.

A diferença entre estas duas *MIB's* reside nas hierarquias usadas para representar os objetos. Na *MIB* da *ISO* são definidas três hierarquias: hierarquia de herança, hierarquia de nomeação e hierarquia de registro.

A hierarquia de herança ou de classes está relacionada às propriedades associadas a um determinado objeto. .Dentro desta hierarquia diz-se que objetos da mesma classe possuem propriedades similares.

No caso da Internet não são usados os conceitos de classes de objetos e seus respectivos atributos. São definidos tipos de objetos. A definição de tipo de objetos contém cinco campos: nome textual com o respectivo identificador de objeto (*Object Identifier*), uma sintaxe *ASN.1*, uma descrição do objeto, o tipo de acesso e o status.

A hierarquia de nomeação ou de *containment* é usada para identificar instâncias de objetos. Este tipo de hierarquia não é definido no caso da Internet.

Finalmente tem-se a hierarquia de registro que é especificada em ambos padrões.

Capítulo 5

5.1 Gerência Pró Ativa de redes com redes neurais Artificiais e Agentes

5.2 Por que utilizar redes neurais?

De acordo com diversas estruturas neurais e algoritmos de aprendizagem propostos por vários pesquisadores, redes neurais possuem certas características exclusivas de sistemas biológicos. Tais características entram em conflito com os tradicionais métodos computacionais. Sistemas de computação baseados em redes neurais têm a capacidade de receber ao mesmo tempo várias entradas e distribuí-las de maneira organizada. Geralmente, as informações armazenadas por uma rede neural são compartilhadas por todas as suas unidades de processamento, característica que contrasta com os atuais esquemas de memória, onde a informação fica confinada em um determinado endereço.

Em um sistema de rede neural, a informação pode parecer ter representação redundante, porém, o fato de que ela se encontre distribuída por todos os elementos da rede significa que mesmo que parte da rede seja destruída, a informação contida nesta parte ainda estará presente na rede, e poderá ser recuperada. Portanto, a redundância na representação de informações em uma rede neural, diferente de outros sistemas, transforma-se em uma vantagem, que torna o sistema tolerante a falhas. Os atributos de uma rede neural, tais como aprender através de exemplos, generalizações redundantes, e tolerância a falhas, proporcionam fortes incentivos para que as redes neurais sejam a escolha apropriada para ajudar no processo de Gerência Pró-Ativa de redes de computadores . Todo o potencial de uma rede neural pode ser enumerado nos parágrafos seguintes.

O modelo de rede neural tem muitos neurônios conectados por pesos com capacidade de adaptação que podem ser arranjados em uma estrutura paralela. Por causa

deste paralelismo, a falha de alguns neurônios não causa efeitos significantes para a performance de todo o sistema, o que é chamado de tolerância a falhas.

A principal força na estrutura de redes neurais reside em suas habilidades de adaptação e aprendizagem. A habilidade de adaptação e aprendizagem pelo ambiente significa que modelos de redes neurais podem lidar com dados imprecisos e situações não totalmente definidas. Uma rede treinada de maneira razoável tem a habilidade de generalizar quando é apresentada a entradas que não estão presentes em dados já conhecidos por ela.

Redes neurais podem ter várias entradas e várias saídas, elas são facilmente aplicáveis a sistemas com muitas variáveis, e a sistemas distribuídos.

Com o avanço em tecnologias de hardware, existem componentes com funções voltadas a sistemas com implementações voltadas para redes neurais, o que traz uma velocidade adicional à computação neural.

5.3 Porque usar Agentes?

Quando se trata de gerenciamento de redes de computadores, estamos trabalhando com um ambiente, que em primeiro lugar, é distribuído, e que em segundo lugar , esta é afetado por uma série de fatores, que acabam por alterar seu parâmetros muito rapidamente. Para que uma rede neural possa ser utilizada na solução dos problemas propostos, é necessário que ela seja alimentada com dados sobre a situação atual da rede.

Como a proposta é fazer um gerenciamento Pró- ativo da rede, de forma automática, a utilização de agentes se configura numa escolha lógica, para prover a rede neural com as informações necessárias ao seu funcionamento.

Os agentes, uma vez desenvolvidos, ficariam ativos na rede, colhendo informações que seriam repassadas aos respectivos módulos de gerenciamento, onde seriam processadas por redes neurais. Conforme os resultados , o administrador da rede seria avisado para tomar as medidas corretivas necessárias, ou dependendo do problema e do grau de automação escolhido, poderiam até mesmo ser tomadas as medidas corretivas pelo próprio sistema de gerenciamento.

Neste escopo, fica claro que deveremos utilizar agentes móveis, inteligentes ou não, e umas das escolhas mais apropriadas seria a utilização dos *Aglets*, que foram mostrados com maiores detalhes no capítulo 3 , item 3.15.

Neste trabalho será dada maior ênfase às redes neurais propostas para solução dos problemas mais comuns no gerenciamento de redes de computadores, deixando o desenvolvimento dos agentes para um estudo posterior.

5.4 Problemas que afetam o Desempenho da rede

Aqui serão descritos alguns dos problemas mais comuns que afetam o desempenho de uma rede de computadores. Como base para essa descrição, são definidos os objetos necessários para serem gerenciados com o objetivo de melhorar o desempenho da rede e de aplicar a tão necessária gerência pró-ativa nas redes atuais.

Considerando a descrição de problemas de desempenho, é necessário detalhar ainda mais o que é a gerência de desempenho, e como essa função de gerência em conjunto com as funções de gerência de configuração e de falhas, podem ser trabalhadas para alcançar uma gerência pró-ativa de redes como um todo.

A gerência de desempenho é a encarregada de garantir que a rede se mantenha acessível e descongestionada para que os usuários possam utilizá-la eficientemente. Para garantir isso, a gerência de desempenho executa as seguintes ações (LEINWAND,1993):

- Coleta de dados sobre a utilização de enlaces e dispositivos da rede;
- Elabora uma análise relevante de dados para discernir tendências de alta utilização;
- Estabelece os limiares de utilização;
- Utiliza simulação para determinar como a rede pode ser alterada para melhorar seu desempenho.

É importante ressaltar que a gerência pró-ativa de redes não só envolve gerência de desempenho, como também as funções de gerência de configuração e de falhas. São essas funções, em seu conjunto, que garantem um bom nível de desempenho na rede. A identificação de falhas que afetam o desempenho da rede e a sua configuração adequada

é que garantem um bom nível de serviços aos usuários e, portanto, ajudam a estabelecer a gerência pró-ativa propriamente dita.

A gerência de falhas permite aos administradores de redes detectar e isolar problemas. Os principais procedimentos que a gerência de falhas executa são o relato das ocorrências de falhas, a manutenção de *logs* dos eventos de falhas recebidos e a sua correção.

A gerência de configuração permite aos administradores exercer controle sobre a configuração da rede. Essa função proporciona procedimentos para colecionar e propagar dados relacionados ao estado corrente dos recursos da rede, modificar os sistemas e atributos da rede, e trocar a configuração do sistema.

Cada uma das funções de gerenciamento, dentro do contexto de gerência pró-ativa, têm características diferentes das que são cumpridas na gerência reativa de redes. Com a gerência de desempenho pró-ativa, o administrador da rede deve ser capaz de saber quando se está atingindo o limite da capacidade da rede, através de uma análise do seu perfil que mantenha informações sobre o tráfego normal da rede num determinado período de tempo (MOTOROLA,1993).

Ao fazer a análise do perfil da rede, é possível identificar mudanças que poderiam ultrapassar os limites pré-estabelecidos pelo administrador da rede. Após essa análise, em certos casos, é útil alterar a configuração de alguns dispositivos específicos para manter ou melhorar o desempenho e é claro, evitar que o usuário tome conhecimento disso. Por exemplo, se numa estação da rede acontece um problema que provoca danos na memória, com a gerência de configuração pró-ativa é possível carregar automaticamente registros de configuração nessa estação. Assim não são necessários técnicos especializados para resolver o problema. Também com esse tipo de gerência podem ser definidas alternativas de configuração para variar a operação da rede em diferentes situações e períodos de tempo.

O principal objetivo da gerência de falhas é encontrar e determinar problemas em uma rede. Para isso são estabelecidos limites. Na gerência de falhas pró-ativa, a idéia é definir problemas potenciais e identificá-los antes que ocorram.

Como é possível observar, a gerência de falhas, de configuração e de desempenho estão muito relacionadas e são as principais funções nas quais deve estar baseada a gerência pró-ativa como um todo.

5.5 Problemas Mais Comuns

A seguir são descritos os problemas comumente identificados que afetam direta ou indiretamente o desempenho de uma rede. Dos problemas descritos aqui, alguns são obtidos de bibliografia sobre *troubleshooting* e outros estão baseados nas informações obtidas de administradores de redes, através das listas de discussão eletrônica sobre tópicos de redes.

5.5.1 Problemas de degradação no nível físico

No nível físico, conectores de redes e placas em mau estado são alguns dos principais problemas que afetam o desempenho da rede. Isso provoca uma alta taxa de colisões, ainda num nível de utilização baixo, que logicamente pode provocar congestionamento e, portanto a degradação do desempenho.

Nesse nível, o interessante para monitorar e poder determinar se há problemas de hardware ou mesmo de software, são as taxas de erros de transmissão. Os erros de transmissão estão classificados como(NEMZOW,1992):

- Erro *CRC* (*Cyclic Redundancy* Check): erro de *checksum* no pacote *ethernet*, usualmente descoberto com os outros tipos de erros;
- Erro de alinhamento (*alignment errors*): ocorre quando o pacote não é múltiplo de 8 bits e é usualmente indicativo de um erro de *framing*, ou seja, que não têm o comprimento especificado pelas normas;
- Pacotes com o tamanho em bytes abaixo do limite inferior permitido (*Undersize Packets*); conforme (ISO,1992), o limite inferior permitido é 64 bytes;
- Pacotes com o tamanho em bytes acima do limite superior permitido (*Oversize Packets*); conforme (ISO,1992), o limite superior permitido é de 1518 bytes.

As taxas significativas desses tipos de erro indicam mau funcionamento de hardware. Por exemplo, erros em pacotes com o tamanho excedido resultam quando os *transceivers* pegam os pacotes com o número de bytes superior a 1518, indicando a

possibilidade do *transceiver* estar defeituoso. Também esse tipo de erro pode indicar problemas com a placa controladora da rede.

Entre os problemas que podem ser originados no nível físico, encontra-se o problema de colisões. Mais adiante se detalha um pouco mais esse problema, porém vale mencionar que uma má configuração de equipamentos do nível físico pode provocar um incremento na taxa de colisões. Por exemplo, quando os *transceivers* estão muito perto um do outro. Essa limitação aplica-se a repetidores ou outros dispositivos que podem ter *transceivers* incluídos em seus próprios equipamentos.

Considerando a quantidade de problemas que a má configuração e localização do equipamento de rede provoca, são apresentadas algumas sugestões que ajudam a evitá-los:

- Os cabos coaxiais e cabos de par trançado são inertes, sendo prudente deixá-los afastados de fontes de água, que podem cortar a transmissão elétrica quando há contato no cabo, num conector ou em uma placa controladora;
- É aconselhável afastar os cabos coaxiais e cabos de par trançado de fontes elétricas de alta tensão, que podem interferir com o sinal causando uma ruptura da transmissão;
- Os campos magnéticos ou elétricos podem entrar em conflito com a transmissão *ethernet* e introduzir altas taxas de erros; por isso, é aconselhável manter o cabo de transmissão afastado das seguintes fontes:
    - Fontes de rádio de alta freqüência;
    - Aparelhos de laboratórios médicos (esses tipos de aparelhos podem corromper o tráfego na rede);
    - Aparelho telefônico;
    - Antenas de rádio;
    - Condutos de ar condicionado.

5.5.1.1 Temperatura inapropriada para a resistência dos cabos

Em ambientes onde as variações de temperatura são muito drásticas (por exemplo, em lugares onde o verão é muito forte) pode-se observar, por exemplo, que os cabos coaxiais que são utilizados para interligar dois prédios estão expostos a essa temperatura extrema e sofrem contrações e expansões, que resultam em microfraturas no núcleo do cabo, alterando as propriedades elétricas do mesmo (por exemplo, a sua impedância elétrica). A conseqüência disso é uma alta taxa de colisões, o que sem dúvida afeta o desempenho da rede. Por essa razão, como já foi mencionado, é aconselhável que ao fazer a instalação do cabo coaxial, ele fique afastado de fontes de calor (por exemplo, o canal condutor do ar condicionado) (NEMZOW,1992).

Segundo (LOGG,1994), no SLAC (*Stanford Linear Accelerator Center*) foi notado que em certas localizações onde o verão é muito forte, freqüentemente surgem problemas associados a altas temperaturas. Em consultas a administradores de redes, através das listas eletrônicas aqui no Brasil, esse tipo de problema não foi identificado.

5.5.2 Congestionamento e Controle de Fluxo

Como o protocolo IP é um protocolo sem conexão, os roteadores não podem reservar memória ou recursos de comunicação antes da recepção dos datagramas. Por conta disso, sua capacidade pode ser excedida com o tráfego, provocando o que se conhece como congestionamento.

O congestionamento poderá ocorrer por duas razões. A primeira seria quando um computador de alta velocidade gera tráfego mais rápido do que a rede pode transportá-lo. Por exemplo, um supercomputador que gera tráfego para uma interconexão de redes, eventualmente os datagramas gerados podem precisar atravessar uma outra rede de baixa velocidade, embora o próprio supercomputador esteja em uma rede de alta velocidade. Nesse exemplo, o congestionamento acontecerá no roteador da rede de baixa velocidade, pois os datagramas chegam até ele em uma velocidade maior do que sua capacidade de encaminhá-los ao destino. A segunda situação seria quando muitos computadores simultaneamente necessitam enviar datagramas através de um único roteador. Com essa sobrecarga de processamento, o roteador poderá entrar em congestionamento.

Quando os datagramas chegam ao *host* ou roteador mais rápido que a sua capacidade de processamento, cria-se uma fila em memória temporária. Se os datagramas são parte de um pequeno conjunto, esse armazenamento temporário resolve o problema. Porém, se o tráfego é contínuo, o roteador eventualmente esgotará sua memória e descartará aqueles datagramas que não podem ser armazenados. Um roteador pode usar uma mensagem *ICMP* (*Internet Control Message Protocol*) *source quench* para liberar o congestionamento. Uma mensagem *source quench* é uma requisição para que o *host* fonte reduza sua taxa atual de transmissão de datagramas. Usualmente, roteadores congestionados enviam uma mensagem *source quench* para cada datagrama que eles descartam. Não há uma mensagem *ICMP* em resposta a uma mensagem *source quench*. Para isso, um *host* que recebe uma mensagem *source quench* de uma máquina M, diminui a taxa de transmissão dos datagramas dirigidos a essa máquina até que não sejam mais recebidos mensagens *source quench*. Após um determinado tempo, o *host* incrementa gradualmente a sua taxa de transmissão até atingir o nível normal, desde que ele não mais receba mensagens *source quench*.

O problema de congestionamento na rede afeta fortemente o seu desempenho, pois o tempo de resposta resultante é muito alto. O congestionamento e o roteamento estão estreitamente relacionados, pois as causas mais importantes que ocasionam o congestionamento são as decisões deficientes de roteamento(COMER,1991).

5.5.3 Problemas de Configuração

Uma rede com uma topologia mal configurada pode causar problemas como altas taxas de colisões e de utilização. Por exemplo, um segmento com muitos *hosts* ou muitos *hosts* em diferentes segmentos ligados a um mesmo servidor (essa topologia pode saturar o segmento da rede onde o servidor está localizado).

Uma má configuração da rede pode provocar uma degradação no seu desempenho. Em redes pequenas, a causa principal de problemas de desempenho é a especificação errada na configuração da rede. Em (NEMZOW 1992), são citados alguns exemplos dessa má configuração:

- Quando o segmento da rede excede os 200 e 500 metros de comprimento (*cheapernet*(10Base 2) e *ethernet* (10Base 5) respectivamente);

- Quando se excede o número máximo de estações permitidas em um segmento (esse número, segundo (ISO 1992), é de 100 estações em redes *ethernet* 10Base 5);
- Quando os cabos dos *transceivers* (*AUI*, *Attachment Unit Interface*) excedem 50 metros de comprimento máximo, segundo (ISO 1992), esse comprimento deve ser menor ou igual a 50 metros.

5.5.4 Problemas com Tormenta de Pacotes Broadcast

Uma tormenta de pacotes broadcast é um conjunto de pacotes enviados ao endereço *Broadcast ethernet* em um volume suficientemente alto para causar problemas em muitos *hosts* na rede.

Em certas ocasiões, uma tormenta de pacotes *broadcasts* acontece junto com outro problema conhecido como avalanche de pacotes, que ocorre quando um único pacote dispara uma avalanche de respostas, obtendo-se assim uma tormenta *broadcasts* de tamanho máximo que sem dúvida afetará bastante o desempenho da rede.

Existem três principais causas que provocam uma tormenta de pacotes *broadcasts* e avalanche de pacotes: problemas nos protocolos, má configuração e implementações com defeitos(SPURGEON,1989).

5.5.4.1 Problemas nos Protocolos

Implementações de protocolos dos níveis superiores da rede que dependem do uso de endereços *ethernet* broadcast no lugar de endereços *multicast* são a principal causa de tormenta de pacotes broadcast e avalanche de pacotes. Um endereço *multicast* pode ser restringido a um pequeno conjunto de *hosts*, ao contrário dos endereços broadcast que são escutados por todos os *hosts* conectados em uma rede *ethernet*.

Todavia, os primeiros controladores *ethernet* ou não tinham um suporte para recepção de *multicast* ou este era muito primitivo. Provavelmente, essa é uma das razões do amplo uso de endereços broadcast por alguns conjuntos de protocolos.

Normalmente um protocolo deveria limitar o uso de broadcast, pois em termos de recursos de rede sua utilização é uma técnica muito onerosa. Por exemplo, no protocolo *Apple Talk* (Fase I), cada *host*, ao fazer uma requisição de nome (*name request*), envia

um conjunto de 20 pacotes ao endereço *broadcasts*. Desse modo, o efeito acumulativo de um conjunto de machintosh II conectados em um mesmo segmento, sendo ligados todos ao mesmo tempo, é igual a uma pequena tormenta de pacotes *broadcasts*. A fase II do protocolo *Apple Talk* inclui o uso de *multicasts* em sua implementação, o que deve ajudar a resolver esse problema.

Um outro exemplo desse tipo de problema é o do protocolo da *Silicon Graphics*, que inclui em suas estações um jogo simulador de vôo, que pode ser executado entre duas estações na rede. O envio dos dados do jogo de uma estação a outra é dirigido ao endereço broadcast *IP*. Esses datagramas correspondentes aos dados do jogo são colocados em pacotes *ethernet* e enviados ao endereço broadcast *ethernet*. Nessa fase, cada *host* na mesma rede onde está sendo executado o programa escuta cada movimento do jogo. Se a rede usa pontes para construir uma *WAN*, cada *host* em toda a *WAN* também escutará os pacotes broadcast do jogo. Se for usado um endereço IP broadcast incorreto, uma rápida seqüência de tormentas *broadcasts* baseada na avalanche de pacotes pode ser gerada.

Outra causa de tormenta de pacotes broadcast, mesmo com o endereço IP broadcast correto, é o envio de um fluxo de datagramas *UDP* (*User Datagram Protocol*) encapsulados em pacotes *IP broadcast* que estão destinados a portas *UDP* desconhecidas (desenvolvidas para software de jogos). Os *hosts* supostamente não deveriam responder a pacotes broadcast, porém muitos o fazem. Nesse caso, as simultâneas transmissões de pacotes ICMP queixando-se das portas UDP desconhecidas nas estações que escutaram os pacotes broadcast causam colisões locais que sem dúvida afetam fortemente o *throughput* da rede.

Um outro exemplo de problemas com tormenta de pacotes *broadcast* é visto no mecanismo *ICMP* para descobrir a máscara da rede. Um *host* procurando pela máscara da rede pode enviar uma requisição ao endereço *IP broadcast*. A maioria dos *hosts* atuais estão programados para responder a essa requisição, produzindo uma rajada de respostas de máscaras da rede. Algumas vezes, as respostas a essas requisições de máscara são também enviadas ao endereço *broadcast*.

5.5.4.2 Má Configuração

Em certas implementações do protoclo *TCP/IP* se tem observado que o contecimento de tormentas *broadcasts* é consideravelmente alto. Isso especificamente foi observado na versão *BSD 4.2* do sistema operacional *UNIX* (nesse sistema está incluída a pilha do protocolo *TCP/IP*), o qual foi projetado com algumas funções que podem em um determinado momento produzir erros, entre essas funções pode-se citar:

- O software do roteador *IP* está ativo para cada *kernel*, incluindo o modo de encaminhamento (*ipforwarding*). Assim, cada *host* na rede pode responder a um roteador quando recebe um pacote dirigido para um destino desconhecido, gerando na rede tráfego suficiente para produzir uma tormenta de pacotes *broadcasts*;
- O endereço broadcast na versão *BSD 4.2* está definido com o campo que identifica os *hosts*, com os bits em "zero". As versões posteriores a *BSD 4.2* têm todos esses bits em "um". Um exemplo de um endereço *broadcasts* na versão *BSD 4.2* é 128.83.0.0 e na versão *BSD 4.3* é o endereço 128.83.255.255 (os dois últimos bytes do endereço estão indicando todos os *hosts* na rede 128.83).

Com essa definição de endereços *broadcasts* na versão *BSD 4.2* do *UNIX*, pode-se ter problemas de tormentas de pacotes *broadcasts* quando em uma mesma rede se têm *hosts* com as duas versões do *UNIX* (*BSD 4.2 e BSD 4.3*). Isso porque os *hosts* com o *Unix BSD 4.2* não reconhecem como endereço válido o tipo de endereço broadcast definido na versão BSD 4.3 (com os bits que identificam os *hosts* em "1"). De igual forma, os *hosts* com o *Unix BSD 4.3* não reconhecem o tipo de endereço definido na versão *BSD 4.2* (com os bits que identificam os *hosts* em "0"). Por exemplo, na rede 128.83, a tormenta *broadcasts* inicia quando um *host* com a versão do *Unix* BSD 4.3 envia um pacote ao endereço *broadcasts* 128.83.255.255. Isso porque os *hosts* que nessa rede estão com o *Unix BSD 4.2* não reconhecem esse endereço broadcast como um endereço válido. Esses *hosts,* ao não reconhecerem o endereço, consultam os seus códigos de roteadores para instruções, e o código do roteador diz: "Tente enviar o pacote ao destino apropriado". Então cada *host* tenta encontrar um endereço *ethernet* associado com o endereço IP 128.83.255.255, enviando simultaneamente uma requisição *ARP* ao endereço *broadcast*. Isso provoca uma avalanche de pacotes que

produz uma tormenta broadcast de tamanho máxima, que também é conhecida como "tormenta *ARP*". Desde que a maioria dos *hosts* na rede tem a mesma arquitetura, ao tentarem resolver o endereço para eles desconhecido, acessam simultaneamente ao barramento, produzindo assim, uma alta taxa de colisões locais.

Finalmente, o software de roteadores nesses *hosts* envia de volta ao *host* que gerou o pacote *IP broadcast*, um pacote *ICMP* indicando "Destino Inalcançável". Dado que esses pacotes são direcionados, somente os *hosts* que estão enviando vêem todos esses pacotes. Ainda é provável que se tenha outro distúrbio de colisões no barramento local, devido à transmissão simultânea de todos esses pacotes. O efeito de colisão é local ao barramento, no qual estão conectados os *hosts* emissores. Nesse caso, a avalanche de pacotes *ICMP* está causando uma saturação de colisões na rede que é simultânea com a tormenta de broadcast. Se o *host* emissor está enviando uma quantidade de pacotes ao endereço IP broadcast incorreto, as conseqüências desse comportamento podem alcançar proporções enormes.

Com esse problema de tormenta *broadcasts*, em (MILLER 1991), recomenda-se que ao ter uma rede com computadores usando ambas as versões (*BSD 4.2 e BSD 4.3*) do sistema Unix, se identifique o computador que está emitindo as requisições broadcast, usando o campo de endereço fonte e fazendo que o endereço broadcast desse computador combine com os outros computadores na rede.

5.5.4.3 Implementações com Defeitos

Implementações de software com defeito também podem gerar tormenta de pacotes broadcast e avalanche de pacotes. Os problemas de configuração descritos anteriormente podem ser melhorados. Porém, é comum surgirem softwares com defeito, embora se consiga convencer os fornecedores de software a não usarem broadcast indiscriminadamente. Alguns exemplos desse tipo de problemas são descritos a seguir.

Segundo (SPURGEON 1989), na universidade de Texas, em Austin, dois roteadores *AppleTalk* interagiram causando uma tormenta broadcast devido à geração incorreta de requisição de nomes (*name lookup request*). Essas requisições foram enviadas ao endereço broadcast em uma taxa de 400 pacotes por segundo e a tormenta resultante foi vista em toda a rede devido à arquitetura de pontes utilizada.

Em (SPURGEON 1989) também se comentava, que na universidade de Stanford, em certa ocasião, uma estação com o sistema *ULTRIX* gerou uma quantidade de pacotes de respostas suficiente para congelar a conexão externa do roteador. Um timer de retransmissão *TCP/IP* permitiu ao sistema *ULTRIX* instantaneamente responder ao pacote *ICMP* de "roteador inalcançável", com uma retransmissão do pacote que gerou o erro. O resultado foi um *VAX* 8800 jogando infinita e rapidamente pacotes no roteador baseado em *MC68000*. Naturalmente que o roteador ficou indisponível para os outros usuários e o acesso à rede parou. É Interessante notar que essa falha foi propagada através dos roteadores *IP*, o que é um exemplo de que esses roteadores não são uma parede corta-fogo perfeita para todas as possíveis falhas de protocolos de rede. A causa que provocou esse problema foi, que nessa rede cada roteador *AIX* continha a máscara de rede errada. Essas máscaras foram propagadas para toda a rede através de *broadcasts ethernet*. Assim, cada *host* que escutou essa máscara parou de enviar pacotes por ter uma máscara totalmente incorreta para roteamento. Esse problema foi corrigido adicionando verificação de sanidade nas máscaras de envio e recepção. Ainda, respostas de rede de máscara normal podem causar problemas quando um host recentemente iniciado envia uma requisição de máscara para broadcast e obtém de volta as máscaras também enviadas a broadcast. Esses eventos normais causam uma mini tormenta de broadcast e à medida que mais *hosts* estão habilitados a responder à requisições de máscaras, o problema piora. Numa grande rede segmentada por pontes, não é difícil para um *host* mal configurado enviar uma resposta de máscara incorreta, porque isso permite que todas os *hosts* adotem um valor incorreto da máscara, o que afeta a habilidade de rotear os pacotes.

Uma rede complexa está sujeita a um número de maus comportamentos devido à interação dos sistemas na rede. Por essa razão é sugerido organizar a rede em segmentos de tamanho razoável e assegurar isolamento entre os segmentos. Uma organização razoável obviamente depende da tecnologia da rede e do tipo de tráfego que está sendo projetado, porém é possível dar algumas sugestões gerais.

O mais importante ao segmentar uma rede é manter o controle dos pacotes broadcast. Se os pacotes broadcast são produzidos em um único segmento, o maior perigo é eliminado. Em alguns casos, tipos específicos de broadcast terão que ser passados de um segmento para outro (geralmente, os *broadcasts* que são passados são

aqueles usados por *hosts* para se reconhecer entre si, como *ARP* para *TCP/IP* e *HELLO* para *DECnet*). O impacto disso deve ser cuidadosamente considerado, pois se a quantidade de pacotes *broadcasts* que passam de um segmento para outro é grande, sem dúvida, o desempenho da rede cairá significativamente.

Para segmentar uma rede muitas vezes são usadas pontes, mas essas, por serem dispositivos do nível dois (camada de enlace da *ISO*) fazem uma comunicação de *host* a *host* (*end-to-end*) e podem passar de um segmento para outro muitos pacotes *broadcasts*. Por essa razão, acredita-se que atualmente é improvável construir uma rede grande usando pontes(BOSACK 1988).

5.5.5 Endereços IP duplicados

O problema de aparecimento de endereços IP duplicados em uma rede deve-se principalmente ao fato de que muitos usuários de PC mexem na configuração de seus microcomputadores e duplicam o endereço do roteador. Como resultado, o roteador marca a *interfaçe* como *down* até que o problema seja resolvido.

Também pode ocorrer o problema de endereços duplicados devido à multiplicidade de lugares onde são registrados os endereços *IP*. O melhor seria ter um lugar único para fazer essa configuração.

Segundo alguns administradores de redes consultados, esse problema de endereços *IP* duplicados não é significativo, pois não afeta o desempenho da rede.

5.5.6 Perda da Conectividade

Este problema é principalmente causado quando o computador solicitado está fora de serviço, por falta de energia elétrica na rede destino ou por falhas nos enlaces e/ou nos roteadores e pontes. Esse tipo de problema afeta o usuário final, mas não se pode dizer exatamente que afeta o desempenho de uma rede ativa.

Todavia, se devido a um segmento da rede estar inativo, um roteador usa uma rota *default* para reencaminhar os pacotes para os segmentos inatingíveis, pode haver um problema de *ping-pong*, isso devido a que o roteador da rota *default* devolve os pacotes, considerando que não há outro caminho senão aquele. Assim os pacotes ficam em *ping-pong* até o *TTL* chegar a zero.

5.5.7 Problemas na Modificação do Software do Sistema

Os problemas nem sempre ocorrem apenas na rede. Deve-se estar consciente das modificações no software do sistema, que podem causar anormalidades na atividade da rede. Por exemplo, *bugs* no software que está sendo instalado ou que está sendo atualizado são comuns.

5.5.8 Roteamento

O roteamento *IP* escolhe um caminho através do qual um datagrama poderia ser enviado. O algoritmo de roteamento IP deve escolher como enviar um datagrama através de múltiplas redes físicas.

Roteamento em uma interconexão de redes pode ser difícil, especialmente entre máquinas com múltiplas conexões físicas. O ideal seria que, quando o software de roteamento está selecionando a melhor rota, sejam examinados itens como a carga da rede, o comprimento dos datagramas e o tipo de serviço especificado no cabeçalho do datagrama. A maioria do software de roteamento é muito menos sofisticado, apenas selecionando rotas baseando-se na suposição fixa do caminho mais curto. O algoritmo de roteamento IP usa uma tabela de roteamento IP onde são armazenados os possíveis destinos e como alcançá-los (COMER 1991).

Depois de dar uma descrição breve da forma de roteamento em redes *TCP/IP*, são apresentados aqui alguns dos problemas que os algoritmos de roteamento provocam e como eles afetam o desempenho da rede(BOSACK 1988).

Muitos dos algoritmos de roteamento comumente usados estão sujeitos à instabilidade de um tipo ou outro. Por exemplo, na mudança de uma rota devido ao fato de que um dos enlaces está fora do ar, várias atualizações podem ser necessárias antes que o sistema concorra com um novo conjunto de rotas. Para uma classe comum de algoritmos, esse comportamento é chamado *Counting to Infinity*, normalmente requerendo aproximadamente 10 atualizações por roteador para que ocorra a convergência. A menos que sejam tomadas certas precauções, o resultado desse comportamento é uma alta quantidade de mensagens de atualização em um curto período de tempo. A maioria dessas mensagens de atualização são broadcast e isso pode

saturar os processos de entradas de outros sistemas na rede. As redes que têm linhas de comunicação lentas podem ser inundadas por vários segundos.

Há várias medidas defensivas para esse problema. Uma delas é a construção de protocolos de roteamento que convergem rapidamente. Outra solução é introduzir um atraso no processo de atualização. Quando uma mudança de rota é realizada, ela é propagada para outros sistemas depois de um tempo de atraso calculado de tal forma que o tráfego de roteamento não inunde a rede.

Tabelas de roteamento erradas podem provocar um ciclo de roteamento (rota circular) para um dado destino. Um ciclo de roteamento pode consistir de dois roteadores (A e B) - o roteador A tem como rota para o destino D o roteador B, e vice-versa - ou esse ciclo pode consistir de vários roteadores. Quando vários roteadores formam o ciclo, cada um deles tem definido como rota para o destino D o próximo roteador no ciclo. Assim um datagrama passa no ciclo indefinidamente. Mas em redes *TCP/IP*, cada datagrama *IP* contém um contador que indica o seu tempo de vida na rede. Esse contador é o *TTL*. Um roteador decrementa o contador *TTL* quando ele processa o datagrama e o descarta quando o contador está em zero. O problema de rotas circulares geralmente acontece durante a recuperação de uma falha em um enlace, pois a maioria dos algoritmos de roteamento não garantem a entrega de mensagens que precisam passar perto do enlace com falhas(COMER 1991).

5.5.9 Colisões

A taxa de colisões é o principal parâmetro em uma rede para saber como está o seu desempenho. Esse parâmetro está relacionado com o parâmetro de utilização da rede, pois se a taxa de utilização sobrepassa 30 %, a taxa de colisões é significativamente incrementada. Assim uma rede sobrecarregada apresentará uma taxa de utilização alta (acima de 35 ou 40 %) e conseqüentemente a taxa de colisões será maior que 2 % do seu tráfego. Uma rede com uma taxa de colisões menor que 2 %, apresenta um nível de desempenho aceitável. Se esse valor for superado, deve ser investigado onde está o problema antes que ele atinja 5 %, pois isso deve sempre ser evitado(NASSER 1994).

Pode-se concluir que, para manter um nível de desempenho aceitável em uma rede, o nível de colisões deve ser cuidadosamente controlado. Até aqui foi apresentado

que altas taxas de colisões podem surgir, já seja por ter uma rede sobrecarregada (fora da sua capacidade) ou por algum problema físico como os mencionados em alguns dos itens anteriores.

5.6 Uma proposta para o problema de Roteamento.

Com foi mostrado anteriormente, existem uma série de problemas que afetam o desempenho de uma rede de computadores. Como seria inviável apresentar propostas de soluções para todos estes problemas neste documento, será escolhido somente um dos problemas apresentados, deixando os outros para um estudo mais aprofundado. Desta forma serão apresentadas, a seguir, algumas propostas para solução do problema de roteamento em redes de computadores utilizando técnicas de redes neurais. Será também apresentado um método, considerado por muitos como a melhor solução para o problema de roteamento, utilizando a abordagem simbólica tradicional. A apresentação deste método será feita para servir de base de comparação com as soluções baseadas em redes neurais.

Já as soluções apresentadas utilizando redes neurais, irão se basear nos Modelos de Hopfield, que foram mostradas com mais detalhes no capítulo 2 , item 2.12.1.

Para utilização em problema de otimização, os modelos de Hopfield devem utilizar uma função de ativação Sigmoidal, que pode ser vista com mais detalhes no capítulo 2 item 2.7.3., o que torna estes modelos altamente recomendáveis para solução de problemas de otimização. A utilização de redes neurais na solução do problema de roteamento vem trazer uma nova abordagem, já que a solução deste problema algoritmicamente falando, pode esbarrar em problemas NP completos, ou seja, não existe um algoritmo que possa resolvê-lo em um tempo polinomial. Na maioria das vezes, uma solução baseada em redes neurais não permite obter a solução ótima para o problema, mas apenas soluções aproximadas. Mas a principal vantagem de sua utilização reside na potencial velocidade com que elas resolvem o problema. Esta grande vantagem advém do fato do processamento da rede neural ser maciçamente paralelo, ao contrário do processamento seqüencial das soluções algorítmicas.

5.6.1 Algoritmo Shorteset Path First de Dijkstra.

Este algoritmo será usado com base de comparação, por se tratar de um algoritmo que visa encontrar a solução ótima para o problema de roteamento, o que na maioria das vezes demanda um tempo de resposta muito alto. Existem outros algoritmos para resolver o mesmo problema, como o proposto por *Belam-Ford*, que segundo (*HUITEMA*, 1995) é o algoritmo mais utilizado. O algoritmo *Dijkstra* segundo (*HUITEMA* , 1995) funciona da seguinte forma:

- Consideremos dois conjuntos de nós na rede: o conjunto dos nós avaliados, *A*, para os quais os caminhos mais curtos são conhecidos, e o conjunto dos nós restantes, *R*.
- Consideremos também um conjunto de Caminhos, *O*, ordenados por ordem crescente de custos.

O algoritmo consiste em:

1. Inicializar o conjunto *A*, para conter apenas o nó de origem *S,* e o conjunto *R* para conter todos os outros nós. Inicializar a lista dos caminhos *O* para conter os caminhos de um só segmento com origem em *S*. Cada um destes caminhos tem um custo correspondente à métrica da ligação física. Ordenar *O* por ordem crescente do custo.
2. Se a lista *O* está vazia, ou se o primeiro caminho em *O* tem uma métrica infinita, marcar todos os nós restantes do conjunto *R* como inatingíveis. O algoritmo Terminou.
3. Primeiro examinar *P*, o caminho mais curto da lista *O*. remover *P* da lista *O*. Seja *V* o último nó de *P*. Se *V* já está no conjunto *A*, continuar no passo 2 . Senão *P* é o caminho mais curto até *V*. Mover *V* de *R* para *A*.
4. Construir um novo conjunto de caminhos candidatos, concatenando *P* com todas as ligações que começarem em *V*. O custo destes caminhos é a soma do custo de *P* com a métrica da ligação acrescentada. Inserir ordenadamente as novas ligações na lista ordenada *O* Continuar no passo 2.

Este algoritmo é de ordem (M. Log N) onde M é o número de ligações físicas existentes entre os nós e N é o número de nós (se M > N o que em geral é verdade).

5.6.1.1 Avaliação do algoritmo *Dijkstra*

Conforme pode ser visto claramente na descrição do algoritmo, ele visa encontrar a solução ótima para o problema do roteamento, ou então não encontra nenhuma solução.

É claro que a solução ótima é a mais desejada, mas o custo de se encontrar esta solução nem sempre vale a pena. Neste caso, com o aumento do número de nós, e com o aumento do número de ligações entre eles, em pouco tempo teremos um problema de altíssimo custo computacional.

A utilização de redes neurais para a solução do problema de roteamento nos traz a possibilidade de trabalhar com um caminho que não é exatamente o ótimo(mas que está bem próximo dele) , mas que apresenta um custo computacional mais razoável.

5.6.2 Solução de *Rauch* e *Winarske*

Nesta solução, a rede neural é configurada na forma de uma matriz em que cada linha corresponde a um nó da rede e a cada coluna, corresponde a um *link* no percurso. Por exemplo, em uma rede com 20 nós e que fossem necessários 4 link´s para se chegar da origem até o destino, seria uma matriz com 20 X 5 = 100 neurônios.

Isto se constitui na principal limitação desta solução, pois seria necessário conhecer previamente o número de *links* usados no percurso, o que geralmente não faz muito sentido. Para determinar este número, é possível utilizar um método que será descrito agora.

Comecemos por definir a matriz de custos *C*, quadrada de ordem *N*, aonde *N* é o número de nós da rede, e onde cada elemento representa a capacidade de ligação da ligação entre o nó *i* e o nó *j*. A diagonal principal da matriz (que representa as ligações de cada nó com eles próprios), bem como todas as ligações que não existam correspondem a capacidades nulas (zero).

Se o elemento C_{sd}, onde S é a origem, e D o destino existir, e se $C_{sd} <> 0$, então há uma ligação direta entre S e D. Se $(CxC)_{sd}$ existir, onde (CxC) é o produto da matriz C por si própria , então há um caminho entre *S* e *D* com duas ou menos ligações. Em

geral, se $(C^L)sd$ existir, então há um caminho entre *S* e *D* que requer *L* ou menos ligações.

Este método permite determinar o número de *links* para atingir um destino mas, como será evidente, o caminho com o menor número de link´s não é necessariamente o melhor caminho. Esta é na realidade, uma das limitações inerentes à arquitetura da própria rede neural. Admitindo este fato, utilizamos este método para determinar o número de colunas necessárias na matriz de neurônios.

Considerando o caso de uma rede com 16 nós (enumerados seqüencialmente) em que a origem de um percurso é o nó 1 , e o destino é o nó 13, e em que são necessários pelo menos 4 link´s para atingir o destino. Neste caso a matriz de neurônios vai ser construída com 16 linhas e 5 colunas, num total de 16 x 5 – 80 neurônios.

Na tabela 5.1 estão representados os níveis internos de atividade de cada um dos neurônios que compõe a rede neural inicial. Estes valores constituem um exemplo e dependem da topologia da rede em questão. Em (*RAUNCH* & *WINARSKE*, 1988), está descrita a forma de inicializar a matriz. O elemento U_{ij} da matriz representa a percentagem do tráfego que é encaminhada pelo nó i no (j-1)-ésimo *link* do percurso. Por este motivo o total da soma de cada coluna da matriz deverá ser 1, o que garantirá que todo o tráfego é roteado. Na primeira coluna o neurônio que corresponde ao nó de origem é fixado em 1, enquanto que na última coluna é fixado a 1 o neurônio que corresponde ao destino. Para conseguir a convergência da rede neural é definida uma Função de perdas, *J*. Esta função *J* é representada na Equação 5.1 e define-se a partir de duas parcelas J_1 e J_2 representadas respectivamente nas equações 5.2 e 5.3. Nestas equações U_k representa a coluna *K* da matriz e U_{ik} representa o elemento na coluna k

$$j = \frac{1}{2}\sum_{K=1}^{4} U_k^t w u_{K+1} + \frac{\beta}{2}\sum_{K=2}^{4}\left(\sum u_{ik} - 1\right)^2 \qquad (5.1)$$

$$J_1 = \frac{1}{2}\sum_{K=1}^{4} U_K^T W U_{K+1} \qquad (5.2)$$

Elemento	U1	U2	U3	U4	U5
1	1	0	0	0	0
2	0	0.33	0.06	0	0
3	0	0.33	0.06	0	0
4	0	0.33	0.06	0	0
5	0	0	0.06	0	0
6	0	0	0.06	0	0
7	0	0	0.06	0	0
8	0	0	0.06	0	0
9	0	0	0.06	0	0
10	0	0	0.06	0	0
11	0	0	0.06	0.25	0
12	0	0	0.06	0.25	0
13	0	0	0	0	1
14	0	0	0.06	0.25	0
15	0	0	0.06	0	0
16	0	0	0.06	0.25	0

Tabela 5.1 Niveis de atividade iniciais para uma matriz de 16 X 5 neurônios

$$J_2 = \frac{\beta}{2}\sum_{K=2}^{4}\left(\sum_{I} U_{ik} - 1\right)^2 \qquad (5.3)$$

Na equação 5.1, J_1 está representada em forma matricial; *W* representa a matriz de pesos entre os neurônios (que não são mais do que custos).

A forma de calcular os elementos da matriz de pesos está representada na equação 5.4, onde A_{ij} é o trafego na ligação do nó i para o nó j e C_{ij} é a capacidade da mesma ligação.

$$F_1(A_{ij}, Ci_j) = \frac{\rho}{(1-\rho)} \qquad (5.4)$$

$$\rho = \left(\frac{Ai_j}{Ci_j}\right)$$

A expressão em 5.4 apresenta um problema que se prende com o custo F_1 , que é difícil de usar em termos computacionais, uma vez que o crescimento é ilimitado quando $p \rightarrow 1$ .Por este motivo, é adotada uma expressão diferente , que se encontra representada na equação 5.5, onde F_0 é o tempo de transmissão da ligação . A função F_3 representa também o custo da ligação de i para j (W_{ij}).

$$F_3(Aij, Cij) = F_0 + \rho^2 \qquad (5.5)$$

Como facilmente se verifica por observação desta equação, W_{ij} depende do tráfego consignado à ligação, razão pela qual os pesos da rede neural (matriz *W*), a partir dos quais é determinado precisamente este tráfego, devem ser reajustados ao fim de um determinado número de iterações da rede neural. Em termos de concretização em hardware, esta poderia ser uma questão difícil de resolver, mas que não levanta problemas maiores enquanto a avaliação ficar restrita à simulação por software.

Depois de descrito o significado da Matriz *W*, facilmente se compreende que a parcela J_1 será mínima, quando o tráfego tiver sido distribuído pelas ligações de menor custo.Como se poderá verificar pelas equações 5.4 e 5.5, o custo será tanto mais baixo quanto menor for a ocupação e o tempo de transmissão das ligações, o que significa que deverão ser selecionadas as ligações mais curtas e menos saturadas.

A parcela J_2 será nula se o total da soma de cada coluna da matriz da rede neural for igual a 1. Esta parcela impede que a rede neural tente fazer " desaparecer o tráfego" , o que permitiria minimizar a parcela J_1. J_2 constitui, portanto o conjunto de restrições.

As variações na função de perdas J, produzida pela variação apenas numa das colunas *K* da matriz de ativação *U* será dada pela equação 5.6 onde e_i é um vetor com todos os elemento iguais a 1.

$$\Delta J = \Delta U_{kk}^{U}\left(\frac{1}{2}\right)\left[WU_{k+1} + 2\beta e_i\left(\sum_i uik - 1\right)\right] \tag{5.6}$$

O decrescimento da função de perdas *J* é Máximo quando a variação de U_k ΔU_k se opõe à direção do gradiente da função *J*, que está expressa entre colchetes na equação 5.6. Sendo assim ΔU_k é dado pela equação 5.7, embora esta equação esteja escrita em forma vetorial, para toda a coluna U_k.

$$\begin{aligned}\Delta U_k &= -\alpha\left(\frac{1}{2}\right)\left[WU_{k-1} + \beta e_i\left(\sum_i u_{ki} - 1\right)\right] \\ &-\alpha\left(\frac{1}{2}\right)\left[WU_{k+1} + \beta e_i\left(\sum_i u_{ik} - 1\right)\right]\end{aligned} \tag{5.7}$$

Observando com atenção esta equação , facilmente se verifica que o estado interno de atividade de cada neurônio depende não só do estado de seus vizinhos da coluna

anterior, mas também dos da coluna seguinte e de todos os neurônios da sua própria coluna, incluindo ele próprio.

5.6.2.1 Avaliação do método *Rauch* e *Winarske*

Este é considerado um trabalho pioneiro no que diz respeito à utilização de redes neurais para fazer o roteamento de redes de dados . Talvez por este motivo, a solução apresentada está sujeita a algumas limitações que dizem respeito ao fato do número de *links* do percurso ter que ser determinado anteriormente. Outra limitação que não ocorre em outras soluções prende-se ao fato que os pesos (bem como a própria disposição da rede neural) terem que ser reconfigurtados de forma única em cada problema, Mesmo na resolução de um único problema, os pesos da rede neural tem que ser dinamicamente alterados porque dependem da evolução da solução calculada.

De qualquer forma, segundo os autores, a rede neural converge em 250 iterações para uma rede com 16 nós e com percursos de até 4 ligações.

5.6.3 Solução proposta por *Ali* e *Kamoun*.

Ao contrario de outras soluções com redes neurais, onde os neurônios representam os nós da rede de dados, em (*ALI & KAMOUN*, 1993), é apresentada uma solução onde os neurônios representam as ligações físicas entre os nós.

O modelo proposto para esta rede neural, tem $N^2 - N$ neurônios, aonde N é o número de nós da rede. Assim, o número Máximo de ligações físicas desta rede será $N(N-1)$, se a conectividade for total, como se impõe em um modelo de *Hopfield*.

O neurônio na linha m, coluna i representa a ligação do nó m para o nó i, sendo a numeração dos nós feita seqüencialmente. Assim, se o neurônio na linha m, coluna i estiver ativo $(Xmi = 1)$, isto significa que a ligação física representada pelo neurônio fará parte do roteamento calculado pela rede neural.

5.6.3.1 Função de energia:

Vamos definir C_{mi} como o custo do nó m para o nó i (0 se não existir), e ρ_{mi} da seguinte forma:

- $\rho_{mi} = 1$ se o arco do nó m para o nó i não existe, e
- $\rho_{mi} = 0$ caso contrário.

A função de energia para esta rede está representada na equação 5.8, onde os índices d e s se referem , respectivamente ao destino e à origem.

$$E = \frac{\upsilon}{2}\sum_{m=1}^{n}\sum_{\substack{i=1 \\ i\neq m \\ (m,i)\neq(d,s)}}^{n} C_{mi}x_{mi} + \frac{\upsilon_2}{2}\sum_{m=1}^{n}\sum_{\substack{i=1 \\ i\neq m \\ (m,i)\neq(d,s)}}^{n} \rho_{mi}x_{mi} + \tag{5.8}$$

$$\frac{\upsilon_3}{2}\sum_{m=1}^{n}\left\{\sum_{\substack{i=1 \\ i\neq m}}^{n} x_{mi} - \sum_{\substack{i=1 \\ i\neq m}}^{n} x_{mi}\right\} + \frac{\upsilon_4}{2}\sum_{m=1}^{n}\sum_{\substack{i=1 \\ i\neq m}}^{n} x_{mi}(1-x_{mi}) + \frac{\upsilon_5}{2}(1-x_{ds})$$

Nesta equação, cada uma das parcelas tem uma contribuição bem definida em termos do comportamento global da função de energia:

- $\mu 1$ minimiza o custo total do percurso
- $\mu 2$ elimina ligações inexistentes do percurso
- $\mu 3$ Esta parcela será nula se todos os nós que forem destino de uma ligação ativa forem origem de outra. Isto assegura que todos os percursos são circulares, sendo fechados por uma ligação forçada do destino para a origem.
- $\mu 4$ força a rede neural a convergir para um estado válido, que estará num dos cantos do *Hipercubo* definido pelos $X_{mi} E \{0,1\}$
- $\mu 5$ força o percurso a fechar-se num ciclo , incluindo para isso uma ligação do destino para a origem, independente dela existir ou não . Esta parcela não faz parte do percurso, mas tem que ser incluída por causa da parcela μ3.
- O resultado final para a rede na figura 4.1, que seve de exemplo, em termos de ativação dos neurônios está representada na tabela 4.2, admitindo que o caminho mais curto do nó 1 para o nó 3 é 1-2-5-3. Note-se que no percurso está incluída uma ligação que não existe de fato de 3 para 1 , para fechar o circuito com foi mencionado anteriormente.

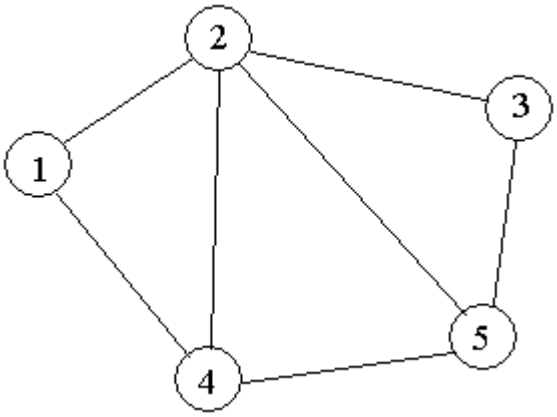

Figura 5.1 Configuração de uma rede e respectivo Caminho mais curto.

		D E S T I N O				
O		1	2	3	4	5
R	1		1	0	0	0
I	2	0		0	0	1
G	3	1	0		0	0
E	4	0	0	0		0
M	5	0	0	1	0	

Tabela 5.2 Ativação final da rede neural.

Além disso, temos ainda dois outros aspectos. Em primeiro lugar, a existência de ciclos é evitada na parcela μ1. Em segundo lugar, não há nenhuma restrição quanto ao número de *links* do percurso, que pode ser qualquer um. Neste caso podemos constatar que a linha e coluna número 4 ficaram em branco porque o percurso não passa pelo nó 4 . Logo, não há nenhuma ligação com origem ou destino em 4, que participe nesse mesmo percurso.

5.6.3.2 Avaliação do método *Ali Kamoun*.

Pelo resultado apresentado pelos autores, a rede converge para problemas que vão até os 15 nós (com 20 fluxos de dados diferentes na rede).Em (*PARK* & CHOI, 1998) existe a afirmação que este tipo de rede neural não converge para a solução válida quando a rede tem mais de 20 nós.

De qualquer forma, a grande vantagem deste método reside na capacidade que ele tem de se adaptar a novas situações de calculo, o que se deve ao fato de a configuração da rede ser introduzida no limiar. Isto permite também configurar a rede neural de forma a exprimir redes de dados que apresentem ligações que não sejam bidirecionais, o que é essencial em termo de aplicação prática.

5.6.4 Solução proposta por *Park & Choi.*

Este método é uma variante do método de *Ali & Kamoun*, que se aplica também para múltiplos destinos.

Nesta variante, será incluída uma Função representada por ϕi que está definida na equação 5.9, e que foi incluída na função de energia porque agora, não é incluído no percurso a ligação de retorno, que liga o destino à origem(por que podem haver vários destinos), de forma que o nó de origem não é destino de qualquer ligação , assim como os nós de destino não são origem de nenhuma outra. Para mais detalhes veja-se (*PARK & CHOI*, 1998)

$$\Phi_i = \begin{Bmatrix} 1, se(i = s) \\ -1, se(i = d) \forall d \\ o, se(caso - contrario) \end{Bmatrix} \tag{5.9}$$

Aplicando–se neste método o mesmo raciocínio que utilizamos no método *Ali Kamoun*, podemos facilmente deduzir os pesos e os limiares que devem afetar cada neurônio da rede, representado nas equações 5.10 e 5.11 respectivamente.

$$w_{mi,zj} = D\delta_{mz}\delta_{ij} - F\delta_{mj}\delta_{iz} + C\left(-\delta_{mz} - \delta_{ij} + \delta_{jm} + \delta_{iz}\right) \tag{5.10}$$

$$I_{mi} = -\frac{A}{2}C_{mi} - \frac{B}{2}\rho_{mi} - \frac{D}{2} + C\left(\delta_{ms} - \delta_{is} - \delta_{md} + \delta_{id}\right) \tag{5.11}$$

As constantes A,B,C,D e o significado das respectivas parcelas têm um correspondente direto com as constantes $\mu 1 \mu 2 \mu 4$ da equação 5.8. Não existe

correspondência com a parcela $\mu 5$ desta equação pelo motivo que já citamos de que o percurso já não tem que ter retorno do destino para a origem. Em vez disso, é acrescentada a parcela F que tenta assegurar que $X_{ij} \; x \; X_{ij} = 0$. A idéia desta parcela é impedir ciclos diretos (por exemplo, a ligação de 3 para 5 e a ligação de 5 para 3 ativa simultaneamente). Nota-se que isso não aconteceria normalmente, porque se a equação de energia estiver bem formulada não poderá haver ciclos. Acontece, porém que esta parcela, segundo os autores, acelera consideravelmente o processo de convergência da rede neural.

5.6.4.1 Método *Screening*.

Uma idéia apresentada em (*PARK & CHAOI*, 1998) consiste num método designado por *Screening*. O método consiste em reduzir a dimensão do problema à custa da eliminação de nós da rede que não se afigurem como candidatos a entrar no percurso.

A idéia básica é eliminar todas as ligações físicas com custo superior a um determinado limite, se for possível encontrar um caminho desde a origem até o destino, com as ligações que sobrarem. No processo de eliminação são postos de lado todos os nós que percam a conectividade.

5.6.4.2 Avaliação do método *Park & Choi*.

Para a rede proposta no artigo, e para os pares {origem, destino} considerados, a rede neural depois do método de *screening* logrou sempre atingir , não só resultados superiores aos atingidos pela rede neural de *Ali & Kamoun*, mas inclusive a solução ótima. Porém não são claros os seguintes pontos:

- Se o desempenho é superior num conjunto mais vasto de situações.
- Se a vantagem advém, sobretudo do método *screening* ou das alterações introduzidas na função de energia.

Capitulo 6

6.1 Simulação por Software

No capítulo anterior foram apresentadas algumas soluções baseadas em redes neurais para o problema do roteamento em redes de dados.

Agora será apresentado um conjunto de cenários que vão permitir estabelecer uma comparação entre as soluções com Redes Neurais apresentadas, usando como referência o algoritmo de Dijkstra.

Com esta comparação pretende-se demonstrar o desempenho das soluções apresentadas, e chegar a conclusões sobre a sua utilização pratica.

Antes de iniciar a apresentação dos resultados, serão definidos os parâmetros em cima dos quais a simulação foi realizada:

Foram escolhidas para o teste, apenas duas das três propostas apresentadas anteriormente, por serem as mais indicadas para solução do problema. Deste modo vamos usar as Redes Neurais propostas por Ali e Kamoun e a de Park e Choi, além é claro, do algoritmo de Dijkstra, que será usado como base de comparação.

Ainda sobre o Algoritmo de Dijkstra vale a pena ressaltar que ele sempre procura pela solução ótima, enquanto que, nas Redes Neurais, isso pode não acontecer. Sendo assim, não podemos estabelecer uma base de comparação neste sentido, pois as soluções baseadas em Redes Neurais estariam em clara desvantagem. Baseado nisto, os pontos principais a serem levantados com esta comparação serão:

- Que vantagem, em termos de tempo de execução, poderá advir da utilização de uma Rede Neural no roteamento de uma rede de dados.
- O quão piores são as soluções baseadas em redes neurais quando utilizadas em um ambiente que simula o real.

Os parâmetros levados em conta na simulação foram:

- O tempo de atraso médio das ligações
- A taxa de ocupação do circuito
- Percentual de rejeição das ligações
- Percentual de ligações falhadas
- Percentual de ligações aceitas

O tempo de atraso das ligações é calculado levando em consideração o tamanho do pacote de dados, e o tempo decorrido entre o momento do envio do pacote e a sua chegada ao nó de destino.

A ligações rejeitadas são as ligações que não foram aceitas por falta de banda de transmissão. As ligações falhadas são as ligações que foram aceitas, mas que não chegaram ao destino, por incapacidade do algoritmo de roteamento (a rede não converge), ou por não haver um caminho possível (ou recursos disponíveis) entre o nó origem e o nó destino.

A taxa de ocupação é calculada com base na largura da banda, e da quantidade de informações que está trafegando pela rede em um determinado momento.

6.2 Limitações do Ambiente de Teste.

O ambiente de teste não suporta mensagens Broadcast, para isso acontecer seria necessário um estudo mais aprofundado, o que propositalmente foi deixado para um trabalho posterior.

Não existe controle do fluxo de dados das aplicações, que se assumem bem comportadas, ou seja, não são levadas em conta na simulação situações onde uma aplicação possa prejudicar a performance da rede, devido a algum erro ou simplesmente por ser uma aplicação que ocupe toda a banda da rede disponível.

Todo o mecanismo necessário ao estabelecimento das ligações foi excluído do cenário de testes. Nestes cenários os acontecimentos ocorrem em três tempos diferentes:

- Estabelecimento da ligação
- Troca de pacotes
- Fim da ligação.

Não se considera o aparecimento de novas ligações e a eliminação das ligações já existentes.

6.3 Descrição do Simulador.

Para a realização deste experimento foi utilizado o Simulador de redes de dados ns-2.1b4 Network Simulator Versão 2.1b4. Ele é um simulador modular e de uso gratuito e com o código fonte aberto, o que permitiu a inclusão dos algoritmos de roteamento, mas apesar disso ele possui algumas deficiências, como o fato de não permitir a reserva de recursos para as ligações. Desta forma, para que a simulação pudesse ocorrer, foi utilizada uma versão modificada do simulador, que foi apresentada e realizada por (FILIPE, 1999).

As alterações têm reflexo basicamente no comportamento das aplicações, que antes de poderem enviar os dados têm que estabelecer uma ligação até o destino. Caso não o façam, seus pacotes são simplesmente descartados. As ligações são estáticas, na medida que não há a possibilidade de alterar o seu percurso em caso de falha de algum nó ou ligação física da rede, embora possam ser dimanicamente iniciadas e terminadas a qualquer momento.

No apêndice A se encontra um manual que descreve o funcionamento do simulador, e no apêndice B estão detalhadas as alterações realizadas no simulador de acordo com (FILIPE,199).

Além destas alterações, foi necessária a inclusão dos algoritmos de roteamento, de acordo com as soluções testadas.Estas soluções foram codificadas em linguagem C, conforme as condições descritas em (Ali & Kamoun, 1993) e podem ser vistas com mais detalhes no apêndice B, juntamente com as outras alterações realizadas no simulador.

6.4 Ambiente de Testes.

As redes utilizadas para a simulação foram redes de 20, 30, e 40 nós, sem hierarquia, com uma largura de banda de 1 Mbps para todas as ligações físicas. Os layouts foram retirados de projetos de redes implantados ou analisados pelo SERPRO (Serviço Federal de Processamento de Dados), com algumas alterações, como:

- A inclusão de alguns nós, ou a retirada de outros para chegar a números exatos (ou próximos dos exatos), e também para deixar a rede mais balanceada; e;
- A inclusão e retirada de algumas ligações(Link´s), que ficaram aproximadamente iguais ao número de nós da rede.

Como fonte de tráfego foi utilizado um débito constante de 400 Kbps, com pacotes 200 Bytes espaçados por 0,004s. Desta forma, cada ligação física suporta no máximo duas conexões, uma vez que a largura da banda é de 1Mbps e o fluxo anunciado das aplicações é de 400Kbps.

6.5 Resultados

Os resultados apresentados a seguir se dividem em 2 grupos:

O primeiro grupo contém dados relativos a avaliações dos parâmetros exclusivos da rede de dados, que como já foi dito antes são a taxa de ocupação da rede, e o atraso médio das ligações. Os seus resultados serão apresentados nas figuras 6.1, 6.2, 6.3, 6.4, 6.5 e 6.6.

O segundo grupo, avalia os algoritmos de roteamento propriamente ditos, contabilizando os números de reservas que foram pedidas à rede, as que foram aceitas, as que foram rejeitadas, e as que falharam. No caso das soluções baseadas em redes neurais também será apresentado o número de interações para convergência da rede, e seus resultados serão apresentados nas figuras 6.7, 6.8, 6.9, 6.10, 6.11 e 6.12.

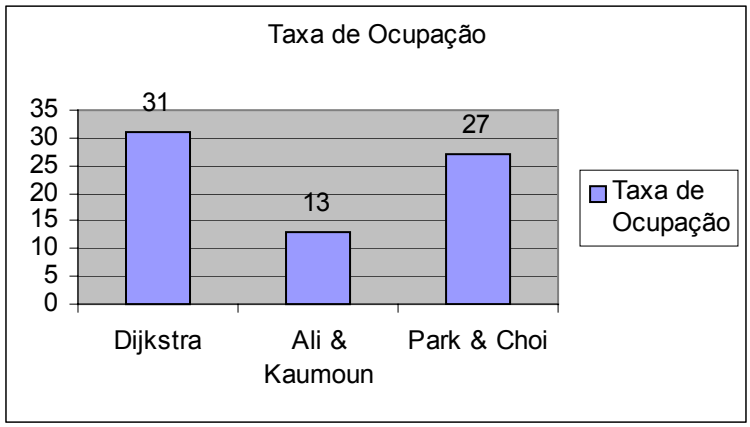

Figura 6.1: Taxa de Ocupação na Rede com 20 Nós

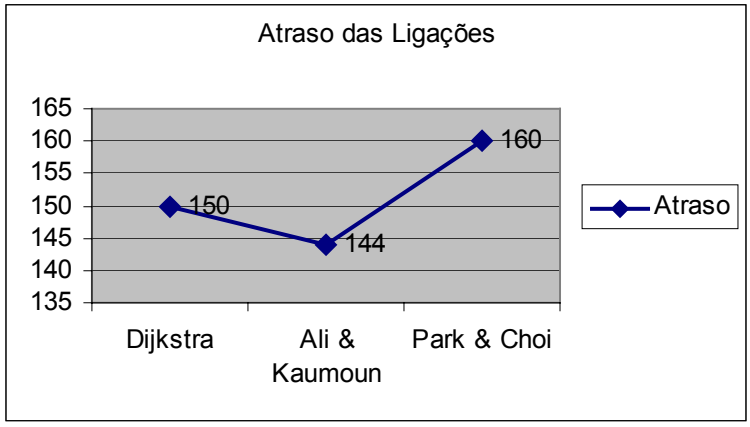

Figura 6.2: Atraso das Ligações na Rede de 20 Nós

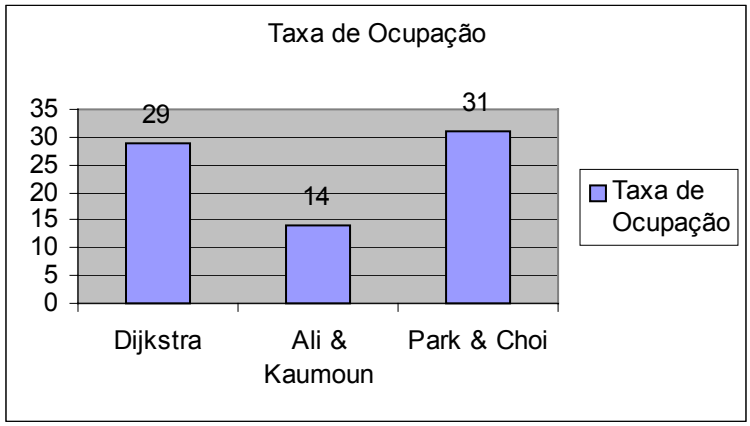

Figura 6.3: Taxa de Ocupação na Rede de 30 Nós

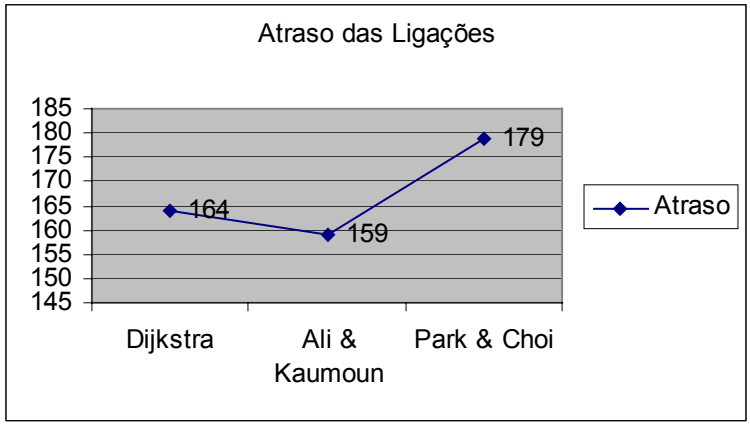

Figura 6.4: Atraso das Ligações na rede de 30 Nós

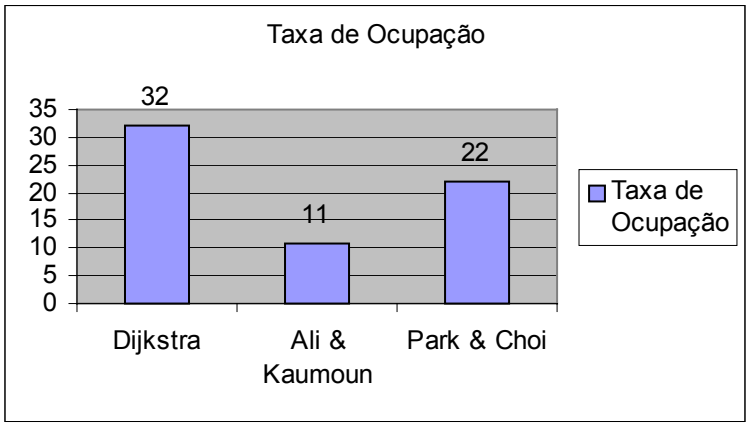

Figura 6.5: Taxa de Ocupação na rede de 40 Nós

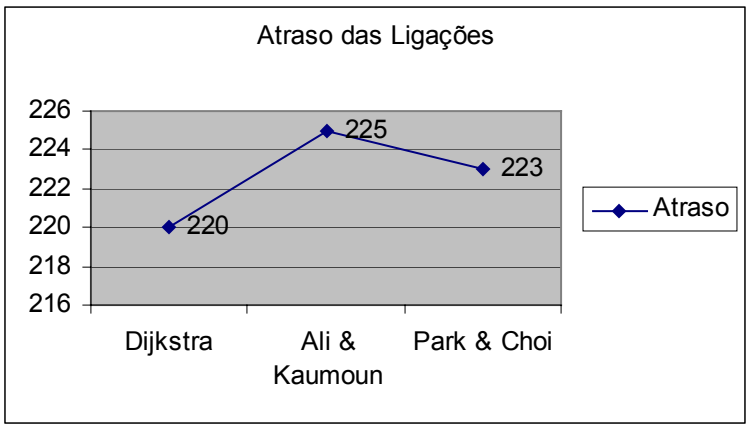

Figura 6.6: Atraso das ligações na Rede de 40 Nós

6.5.1 Análise dos Resultados do Primeiro Grupo.

De modo geral, podemos dizer que as redes neurais apresentam resultados piores que o algoritmo de Dijkstra, e o contrário, aliás seria de se estranhar. Acontece em alguns cenários que com a rede neural, se atinge um débito menor, acompanhado de atrasos mais elevados. É o caso da rede neural de Ali e Kamoun, nos cenários de 20 e 30 nós.

Em termos de débito (que na prática pode ser quantificado pelo número de reservas bem sucedidas), o algoritmo de dijkstra leva sempre a melhor sobre as redes neurais, mas como se falou antes, este comportamento já era esperado.Neste aspecto a solução de Ali e Kaumoun é que apresenta os piores resultados. Comparativamente a esta, na rede neural de Park e Choi, a degradação não é tão significativa, mas é, ainda assim, bastante sensível.

Estes resultados demonstram, que como foi sugerido em alguns artigos (Park& Choi, 1998), a rede neural de Ali e Kamoun não se adequa a redes de dados com mais de 30 nós, número a partir do qual o aproveitamento da rede de dados cai abruptamente. Na rede neural de Park e Choi, a degradação não é tão rápida, mas ainda assim acontece, sobretudo se comparadas aos resultados do algoritmo de Dijkstra.

De forma geral, a rede de Park e Choi, é a que mais se aproxima do desempenho dos algoritmos de Dijkstra, mas estes resultados vão piorando à medida que o número de nós aumenta.

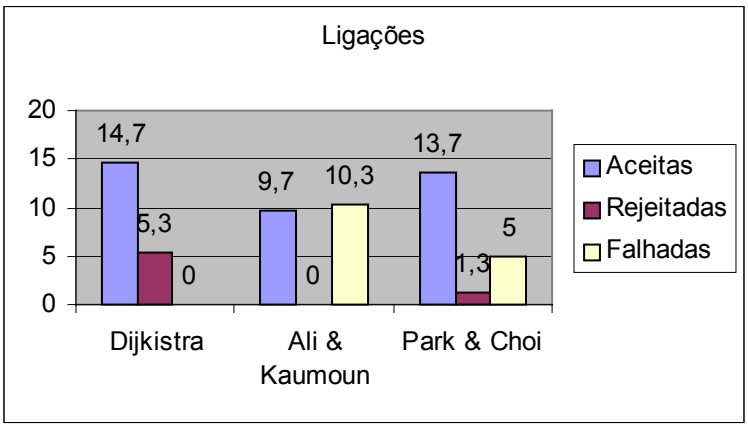

Figura 6.7 Ligações na rede de 20 Nós

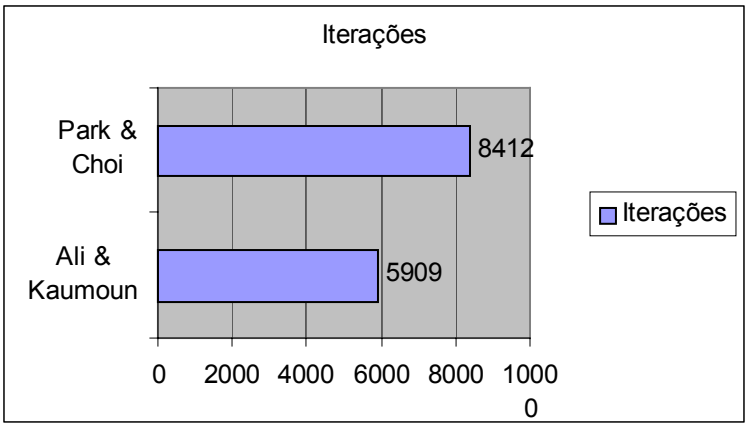

Figura 6.8: Número de iterações na rede com 20 nós.

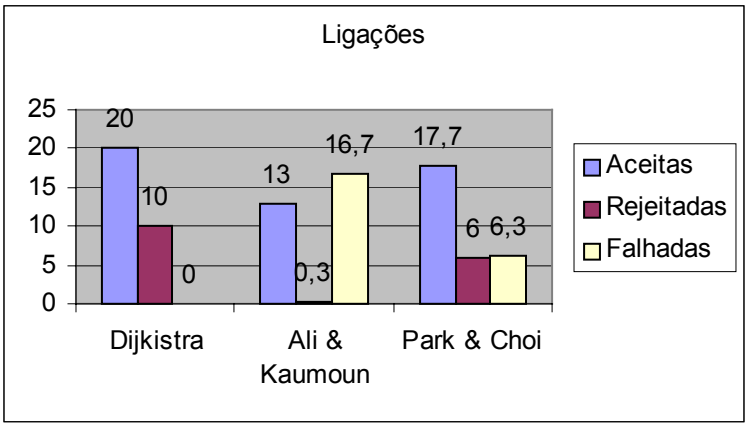

Figura 6.9: Ligações na rede de 30 nós

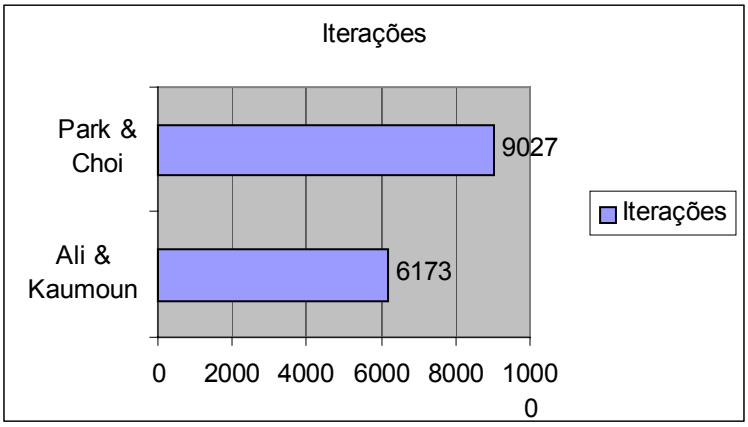

Figura 6.10: Número de iterações na rede de 30 nós

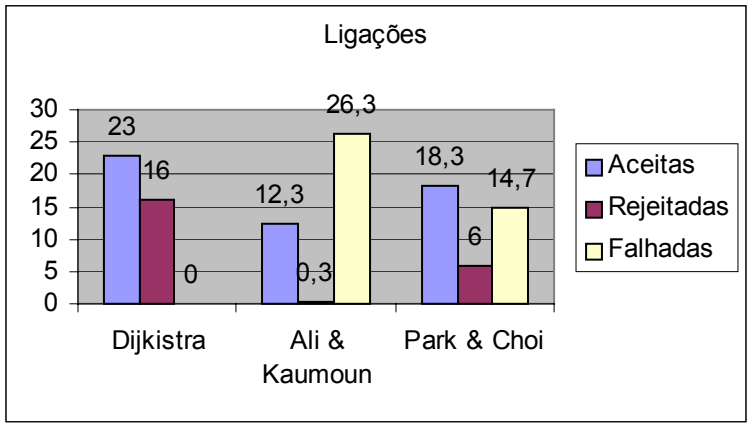

Figura 6.11: Ligações na rede de 40 nós

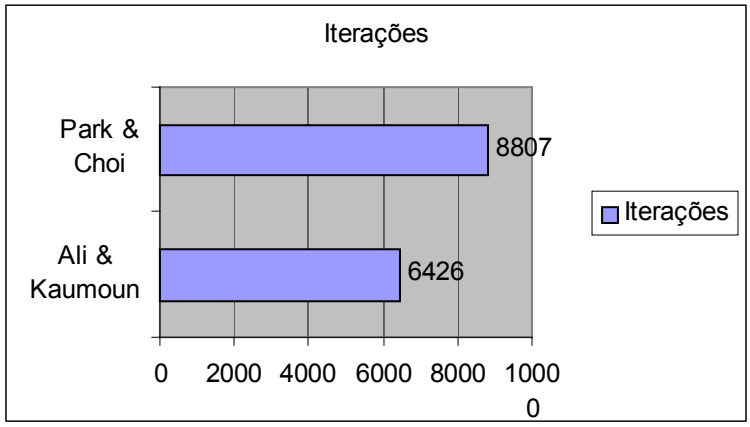

Figura 6.12: Número de iterações na rede com 40 nós

6.5.2 Análise dos Resultados do Segundo Grupo.

Um aspecto que é importante salientar, é que a partir do número de reservas rejeitadas, não é possível tirar conclusões confiáveis. Uma reserva rejeitada acontecerá, em princípio, por deficiência da arquitetura da rede, e não do algoritmo de roteamento. Este tipo de situação acontecerá normalmente quando o nó que faz o caminhamento tem informações desatualizadas relativas aos recursos disponíveis em algumas ligações físicas da rede. Isto não é um problema fácil de se resolver, mas não pode ser atribuído qualquer demérito ao algoritmo de roteamento.

Pelo contrário, o número de reservas falhadas é um importante parâmetro de avaliação. Uma reserva falhada resulta sempre de uma incapacidade do algoritmo de roteamento em encontrar uma solução. Esta incapacidade pode ser justificada pelo fato de não existir nenhum percurso elegível com largura de banda suficiente, mas também pode ocorrer por incapacidade do próprio algoritmo de roteamento. Obviamente, que a segunda hipótese nunca ocorre com o algoritmo de Dijkstra. Aliás, por análise dos resultados, podemos verificar que a primeira hipótese também nunca ocorre com este algoritmo, e também que, se não todas, mas quase todas as reservas falhadas das redes neurais, se devem a sua própria incapacidade em encontrar um resultado, e não por insuficiência de largura de banda de rede.

Em termos de número de reservas falhadas, duas conclusões são imediatas:

- As redes de Ali e Kamoun falham muito mais, e;
- A proporção de falhas aumenta com a dimensão da rede, não só para a rede de Ali Kamoun, mas também para a rede de Park e Choi.

Este problema é relevante por duas razões importantes:

- Em primeiro lugar, seja qual for a dimensão da rede, é inadmissível que as aplicações vejam suas reservas bloqueadas por falha no algoritmo de roteamento, quando existe recurso disponível.
- E em segundo, em redes com reduzido número de nós, é difícil justificar a substituição do algoritmo de Dijkstra por uma rede Neural, porque as vantagens em termo de rapidez não poderão ser tão significativas em termos absolutos.

Quanto ao número de interações necessárias para convergência das redes neurais, não deixa de ser interessante constatar que a rede neural de Ali e Kamoun converge mais rápido do que a rede que lhe é posterior, a de Park e Choi, pelo menos nos cenário testados.

6.6 Análise Geral dos Resultados

Em termos gerais, os dados da simulação servem para mostrar, é possível a utilização de redes neurais no roteamento de redes de dados, mas ainda é necessário algumas pesquisas mais aprofundadas sobre o assunto.As soluções testadas talvez não apresentem um desempenho satisfatório, mas com os avanços nas pesquisas, talvez novas soluções possam ser propostas, e desta forma, o atual cenário possa ser alterado. A proposição de novas soluções para este problema seria uma evolução natural da pesquisa iniciada nesta dissertação.

Conclusão:

Com foi dito no início deste trabalho, é muito difícil arriscar uma previsão das tendências e novas tecnologias que serão utilizadas na área de informática, mas os estudos e as pesquisas realizadas no transcorrer da elaboração deste documento, permitem chegar a algumas conclusões.

A tecnologia de redes neurais ainda tem muito que crescer, e ainda tem muito que oferecer ao relacionamento homem –máquina, principalmente, o fato de adicionar certas qualidades quase humanas aos programas de computadores.

Um exemplo claro disso, é a comparação do Algoritmo de *Dijkstra*, com os métodos baseados em redes neurais, apresentados no capítulo 5 deste documento. Podemos ver claramente, que o primeiro método busca sempre a solução ótima, o que acaba por inviabilizar a sua utilização com uma rede com um grande número de nós, e com um grande número de ligações entre estes nós.

A utilização dos modelos baseados em redes neurais consegue resolver este problema, utilizando uma capacidade quase humana, deixando a solução ótima de lado, quando ela não for possível, e optando por uma solução, que apesar de não ser ótima, será satisfatória nas condições propostas, isto é, deixando a lógica pura e binária dos métodos tradicionais, para trabalhar com possibilidades que antes somente poderiam ser percebidas através dos intrincados processos do raciocínio humano.

É claro, que as qualidades das redes neurais não se comparam à impressionante capacidade do cérebro humano, e que os avanços nesta área podem até mesmo ser desprezáveis, em comparação com o que a mente humana é capaz, mas com certeza, eles representam o início do que poderá ser tornar uma das maiores revoluções na forma como homem e máquina coexistem.A simples idéia de conseguir aliar a impressionante capacidade para cálculos e a rapidez dos computadores, com as impressionantes qualidades do raciocínio humano, fazem das redes neurais uma das mais fascinantes e promissoras áreas para pesquisa em informática na atualidade.

Neste documento, foi abordada somente a utilização das redes neurais na área de gerenciamento pró-ativo das redes de computadores, mas especificamente, foram apresentadas algumas propostas para resolução do problema específico do roteamento em redes de dados. Mas as possibilidades nesta área são muito grandes, principalmente quando se utilizam em adição às qualidades das redes neurais, as impressionantes

capacidades dos agentes inteligentes. Os agentes, também têm um papel muito importante reservado no futuro da coexistência homem-máquina, pois representam uma forma de automatizar, ou até mesmo, uma forma de delegar a responsabilidade para a execução de atividades em substituição do usuário responsável. A utilização dos agentes no contexto do problema proposto neste documento foi muito restrita, mas a sua utilidade para resolução dos outros problemas citados no capítulo 5, se torna praticamente inquestionável.

É claro que este documento representa somente o início de uma pesquisa. Os problemas encontrados na gerência das redes de computadores são muitos, e tendem a aumentar na mesma proporção do crescimento da utilização das redes, e também de sua complexidade.Se levarmos em conta que as redes de computadores tiveram um aumento de utilização impressionante, mesmo em se tratando da área de informática, e também a importância que as redes assumiram no dia a dia das pessoas, teremos uma idéia da quantidade de problemas com que um gerente de rede tem que lidar, e também dos prejuízos ocasionados com a parada desta rede. Desta forma espera-se que futuramente seja possível, utilizar as tecnologias citadas neste documento, para, finalmente, se implementar uma gerência de redes quase que totalmente pró-ativa, capaz de evitar estes prejuízos e facilitar a vida dos administradores.

Anexo A

Network Simulator-2

A.1 Introdução

A.1.1 Objetivos

O ns-2 *(network simulator* versão 2[1]) é, como o nome indica, um simulador de redes, que permite estudar o desempenho de redes de dados, neste caso, sem ligações.

A análise duma rede pode incidir em múltiplos aspectos: na topologia, nas aplicações e protocolos utilizados, no tráfego gerado, no comportamento das filas, etc. Para que estes tipos de análise sejam feitas com um simulador, numa primeira fase e antes de se iniciar a simulação, tem que ser possível especificar, de preferência duma forma simples, configurações arbitrárias para a rede e para os componentes que nela participam. Concluída esta configuração, deve igualmente ser possível a recolha e análise dos dados obtidos durante a simulação.

As questões que se prendem com a configuração da rede e com a simulação propriamente dita, dizem respeito à utilização da interface oferecida pelo simulador. Pelo contrário, o objetivo do presente capítulo é o de apresentar com o detalhe possível o funcionamento interno do simulador.

Nesta descrição, a interface de programação oferecida pelo simulador é deixada para segundo plano, porque se julgam suficientes, para a maioria das aplicações, os exemplos disponibilizados com o simulador. Ainda assim, uma compreensão aprofunda da do funcionamento do simulador e das suas linguagens de programação são pressupostos essenciais para a utilização de algumas partes não documentadas dessa interface.

[1] A *release* que apresentamos neste documento é a 2.1b4.

A.1.2 Componentes do Simulador

Dos muitos componentes existentes no ns-2, vamos debruçar-nos, para já, naqueles que são mais importantes e que permitem obter uma visão geral do funcionamento do simulador. Destacamos, então, os seguintes:

- Nó;
- Ligação física;
- Agentes;
- Aplicações.

Sobre os dois primeiros componentes pouco há a dizer, uma vez que a sua necessidade e as suas funções são óbvias. Em conjunto, compõem a topologia da rede. é de referir que no ns-2 não existe uma fronteira clara entre nó terminal e nó de encaminhamento, pois todos os nós pertencem, por omissão, a esta segunda categoria. Porém, e como é evidente, este aspecto não constitui uma limitação, pois essa diferença quanto ao papel a desempenhar pode ser introduzida explicitamente na configuração da rede, bastando para tal introduzir nós terminais, que tenham apenas uma ligação a outros nós, esses sim, responsáveis pelo encaminhamento.

Definida a configuração da rede, cabe às aplicações introduzir tráfego necessário à simulação. À semelhança do que acontece em qualquer rede, as aplicações (ou se quiséssemos ser mais precisos, as partes duma aplicação distribuída) executam-se num determinado nó em particular, embora num mesmo nó possam coexistir múltiplas aplicações.

Estas aplicações, naturalmente, podem diferir entre si no tipo de tráfego que geram. A regra no ns-2 é a de utilizar aplicações que tanto quanto possível se assemelhem a aplicações com existência real: *ft*p, *telne*t, servidor e cliente de http, etc. é de referir a existência duma aplicação, capaz de gerar tráfego com débito constante (cbr — *Constant Bit Rat*e), embora neste caso particular, seja complicado encontrar alguma aplicação real vulgarizada com características semelhantes.

Para enviar os seus dados, as aplicações utilizam um protocolo de transporte como o UDP ou o TCP (que, por sua vez, utilizam o IP, que é o protocolo do nível de rede do simulador). Estes protocolos de transporte são também responsáveis por diferenças nos padrões de tráfego, uma vez que o seu funcionamento também difere. A título de exemplo, basta-nos pensar que o TCP é um protocolo orientado a conexões, enquanto que o UDP não é.

É importante referir que não existe uma igualdade estrita entre os protocolos do ns-2 e os protocolos reais da pilha de protocolos TCP/IP. Na verdade, existe apenas uma igualdade aproximada, mas parece ser obviamente vantajoso utilizar os mesmo nomes e os mesmo acrônimos, de forma a identificar rapidamente as funcionalidades dos protocolos[2] .

Geralmente, para concretizar os protocolos existentes é usado um mecanismo genérico com funcionamento autônomo — o agente. Um agente é uma entidade que aparece nas diferentes camadas da rede, capaz de receber, gerar e enviar pacotes. Estas características permitem, inclusivamente, que um agente desempenhe papeis que vão para além da realização de protocolos, podendo assumir-se, por vezes, como verdadeiras aplicações.

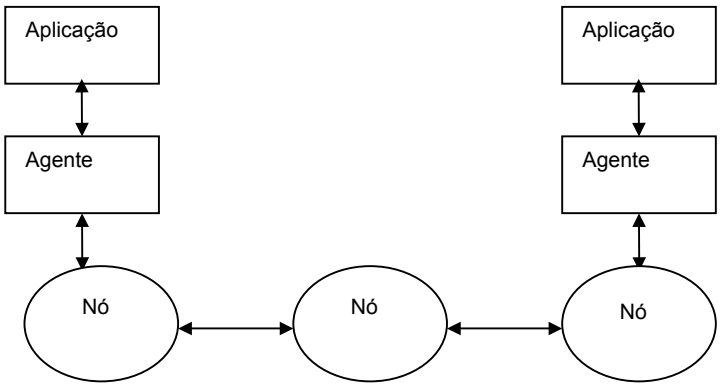

Figura A.1: Elementos constituintes duma rede típica no ns-2

[2] Note-se que os protocolos existentes no ns-2 não se limitam aos que aqui foram enumerados.

A figura A.1 procura ilustrar a posição de cada uma das partes descritas até aqui e respectivas interações.

A.1.3 Escalonador

O ns-2 é um simulador de eventos discretos, isto é, toda a dinâmica da rede é controlada por um escalonador responsável por gerar determinados acontecimentos em momentos predefinidos. A definição desses momentos é feita com base num relógio autônomo e sem relação com o relógio de tempo real. Os acontecimentos a definir podem ir da recepção de pacotes, à chamada de uma função de um qualquer componente da rede ou à geração de um acontecimento específico.

Para ilustrar estas situações, podemos para o primeiro caso referir, por exemplo, a chegada dum pacote que foi enviado através duma ligação e que vai demorar 10 ms, *de tempo do escalonado*r, a chegar ao seu destino. Quanto à invocação de uma função de um componente, e já no que diz respeito ao segundo caso, podemos apontar o exemplo de um protocolo que, no instante X, necessite de reenviar um pacote, caso não obtenha uma dada confirmação. A ativação da função responsável pelo reenvio vai ser feita pelo escalonador. O último exemplo diz respeito à geração de acontecimentos que podem ser algo como a desativação ou ativação duma ligação, para simular uma avaria ou recuperação dessa avaria. Na verdade, este último caso não passa duma concretização particular do caso anterior.

Note-se mais uma vez que não existe qualquer relação entre o tempo do simulador e o tempo real, isto é, os resultados duma determinada simulação terão que ser precisamente iguais (ignorando fatores de natureza aleatória que possam estar contidos na própria simulação), quer sejam obtidos num computador lento, onde a simulação leve um minuto a completar-se, quer num computador onde essa simulação não leve mais do que uns segundos.

A.1.4 Linguagens de Programação

A configuração duma rede para efeitos de simulação apresenta o seguinte problema: usando uma linguagem de programação tradicional compilada, como o C ou o C++, seria necessário recompilar o simulador sempre que quiséssemos alterar aquela

configuração. Por outro lado, usando uma linguagem interpretada incorreríamos sempre numa penalização ao nível do desempenho. Para resolver este problema é usada uma linguagem interpretada para fazer a configuração — OTcl[3] , no caso — que serve de fachada ao simulador que é escrito em C++.

Esta solução, que é nitidamente de compromisso, apresenta, no entanto, alguns inconvenientes importantes, nomeadamente no que diz respeito à interação entre os objetos existentes nas duas linguagens.

Na prática, a fronteira entre o C++ e o OTcl (isto é, entre o que deve ser compilado e o que deve ser interpretado) não é muito clara e o resultado é que por vezes a estrutura do código resultante é de difícil compreensão.

De qualquer forma, é importante referir que o simulador contém duas hierarquias de classes de objetos: uma delas compilada e outra interpretada, embora muitos dos objetos existentes nessas classes sejam os mesmos. Ou seja, a maioria dos objetos é constituída por duas partes: uma compilada com métodos e variáveis definidos em C++ e outra interpretada com métodos e variáveis definidos em OTcl. Note-se que nem todos os objetos são definidos com estas duas metades[4].

É importante compreender que muitos destes objetos não são mais do que os componentes da rede que temos vindo a apresentar (agentes de diversos tipos, aplicações, etc.) além de outros componentes de que ainda não falamos.

A.2 Componentes da Topologia

A.2.1 Nó *unicast* (`Node`)

Um nó é um objeto que só está definido em OTcl, isto é, é um objeto que não existe na hierarquia das classes compiladas do simulador. Contém, como parte integrante, os seguintes objetos, que serão estudados mais adiante: um classificador de endereços e um classificador de portos (estes dois objetos são normalmente definidos exclusivamente em C++, embora isso não impeça que seja possível criar objetos desse tipo em OTcl). Além destes dois objetos classificadores, possui algumas variáveis, que

[3] Tcl orientado aos objetos.
[4] Os nós, por exemplo, são uma importante excepção a esta regra, uma vez que só têm existência na classe hierárquica interpretada, embora contenham objetos com existência dual.

armazenam valores que dizem respeito ao nó em questão, como o seu endereço e a sua identificação, o nome dos objetos classificadores de endereços e de portos, o tamanho da tabela de encaminhamento do nó (esta tabela de encaminhamento não é mais do que o classificador de endereços, como teremos oportunidade de ver), entre outras. A figura A.2, que existe também no manual do ns-2 representa a constituição dum nó. É importante referir que o nó não recebe nem gera qualquer pacote, limitando-se antes a conter os objetos envolvidos na comunicação — agentes e aplicações. Quando um agente que pertence ao nó recebe um pacote, este é enviado para o classificador de endereços do nó onde está alojado o agente. O classificador de endereços encaminha o pacote para o classificador de portos que, por sua vez, o entrega no agente correto (daqui ainda seria entregue a uma aplicação, se fosse caso disso).

Caso uma aplicação existente no nó pretenda enviar um pacote, começa por entregá-lo ao classificador de endereços, que o envia de imediato para o nó seguinte do percurso do pacote (como veremos depois, o pacote não é entregue exatamente ao nó seguinte, mas antes, à ligação física que há de conduzir a esse nó).

Há, ainda, uma terceira situação que pode produzir tráfego num nó e que ocorre quando este não é origem nem destino, mas intermediário dum percurso. Neste caso, é o classificador de endereços que, ao receber um pacote, o reenvia para o salto seguinte do percurso.

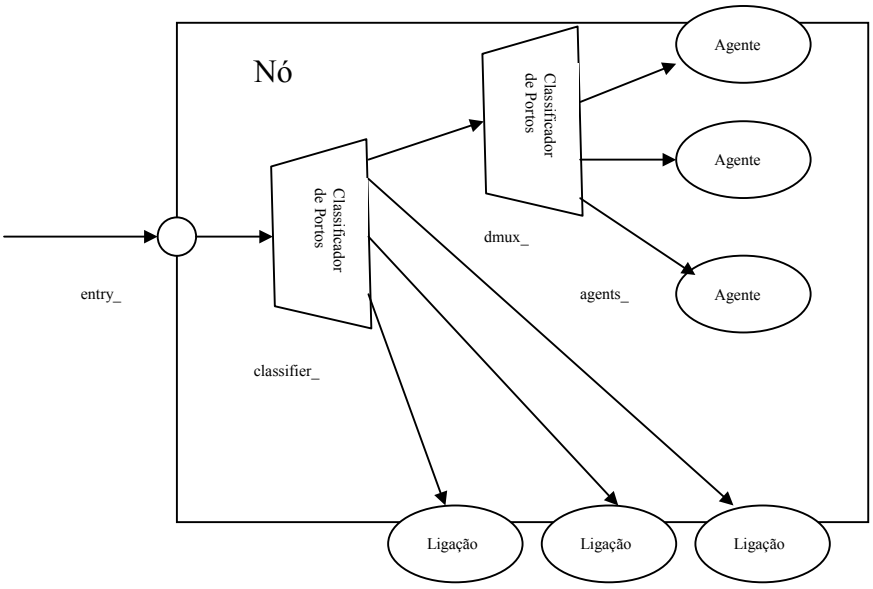

Figura A.2: Elementos constituintes dum nó

Um nó é criado por uma instrução OTcl como a que exemplificamos de seguida:

```
set n0 [$ns node]
```

No momento da inicialização[5] dum nó é imediatamente criado um classificador de endereços. A criação do classificador de portos fica relegada para o momento em que lhe for adicionado o primeiro agente com uma instrução do gênero:

```
$ns attach-agent $n0 $null0
```

Nesta instrução, instala-se o agente identificado pela variável `null0` no nó `n0`, invocando para isso o procedimento `attach-agent` do objeto simulador (`ns`). Este procedimento, por sua vez, vai resultar numa chamada ao procedimento `attach` definido na classe `Node` (onde é definido o nó), que completa, de fato, a instalação.

O objeto `Node` é, em suma e, como vimos, responsável por agrupar os componentes envolvidos na comunicação e por lhes fornecer o suporte de que esta necessita.

A.2.2 Agentes

O papel dos agentes já foi sumariamente explicado atrás. Na arquitetura apresentada na figura A.1 é evidente qual é a posição dos agentes em toda a hierarquia. Como é também óbvio, os agentes vão ter que possuir um endereço individual dentro de cada nó, de forma a que o classificador de portos os possa distinguir. Talvez sem surpresa, o nome dado ao endereço dos agentes é o *porto*. Os agentes, tal como outros objetos do simulador têm partes escritas em C++ e outras em OTcl — depende dos

[5] Os objetos OTcl também têm um construtor, à semelhança do que acontece no C++. Sempre que é criado um novo objeto com a instrução `new` é invocado o construtor para esse objeto. Note-se que o objeto nó (`Node`) vai ser criado dentro do procedimento `node` do simulador `ns`. Interessa referir, que o tipo de nó a criar pode ser determinado com a variável estática do objecto `Simulator`, `no-de_factory_` e que o procedimento `node` da classe `Simulator`, a que pertence o simulador, verifica se o limite máximo de nós que podem ser criados foi ultrapassado.

agentes. Algumas das variáveis mais importantes existentes no agente base, a partir da qual todos os outros são derivados, são as seguintes:

- `nsaddr_t addr_, dst_`: endereço de origem e de destino, que são, respectivamente, o endereço completo (nó + porto) do próprio agente e do agente a que se destinam os pacotes;
- `size_, type_`: tamanho e tipo dos pacotes gerados pelo agente.

Uma situação subjacente à existência destas variáveis é a de que, geralmente, o destino dos pacotes gerados por um agente é único (sendo esse destino outro agente).

A ligação entre dois agentes é feita com o procedimento `connect` do objeto `Simulator`, como está exemplificado na instrução seguinte:

```
$ns connect $udp0 $null0
```

sendo que, aqui, os agentes são identificados pelas variáveis `udp0` e `null0`.

Note-se que este fato constitui uma séria limitação às funcionalidades do simulador, uma vez que não é possível escolher dinamicamente o destino dos pacotes.

Terá ainda algum interesse explicar, neste ponto, como é que se procede à criação dinâmica de pacotes para envio. Essa criação é feita por duas funções definidas na classe `Agent` (que serve de classe base a todos os tipos de agentes): `Packet * allocpkt()` ou `Packet * allocpkt(int n)`. Em ambos os casos é criado um pacote que é totalmente preenchido com todos os dados relativos ao agente (isto é feito na função `void initpkt(Packet * p)`), embora na segunda função seja possível especificar um tamanho diferente do estabelecido por omissão para o pacote.

O momento em que um pacote é criado e enviado pela rede é decidido pelo código do agente, mas estas funções permitem esconder do programador muitos dos deta-lhes relativos ao preenchimento dos cabeçalhos dos pacotes, que assim são resolvidos automaticamente.

Como se depreenderá pela lógica do funcionamento do agente, tem que haver uma ou mais funções responsáveis por enviar um pacote e uma função que seja invocada sempre que um pacote é recebido. São elas, respectivamente:

```
void send(Packet *, Handler *);
void recv(Packet *, Handler *);
```

Enquanto que a primeira destas duas funções está previamente definida e se limita a entregar um pacote ao classificador de endereços do nó, a segunda é, por definição, invocada assincronamente sempre que um pacote se destina ao agente em questão e tem, forçosamente, que ser definida pelo programador (uma vez que dela depende o comportamento do agente).

A.2.3 Ligações Físicas *(Link*s)

Duma forma simplificada podemos definir uma ligação física como sendo a entidade existente entre dois nós que se encontram ligados entre si. Esta entidade, na realidade é composta por múltiplos objetos: uma fila, um atraso que representa a ligação física propriamente dita, um verificador de TTL[6] e opcionalmente, um número arbitrário de objetos de monitorização[7].

Em termos gráficos, a configuração mínima numa ligação física está representada

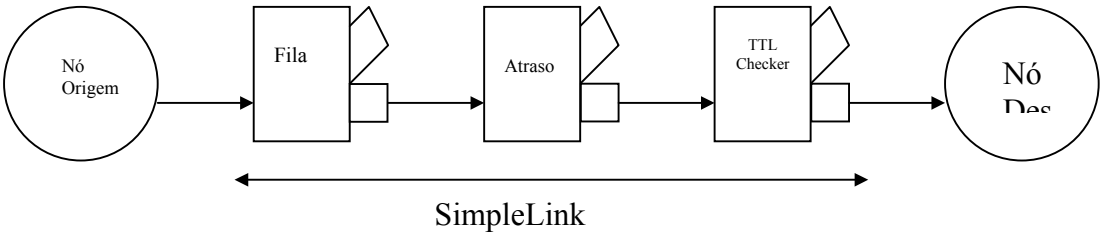

Figura A.3: Elementos constituintes duma ligação física *(lin*k)

na figura A.3. Da figura A.3 transparece a seguinte sequência no percurso efetuado por um pacote: nó de origem (ou intermediário); fila, onde um pacote aguarda pela sua vez e onde poderá ficar retido por tempo indeterminado; atraso, onde fica retido por uma quantidade de tempo predeterminada; TTL *checke*r, onde se verifica o tempo de vida do pacote; e, finalmente, o nó de destino (ou, apenas, o próximo salto), se o pacote não tiver sido descartado em algum dos elementos anteriores.

---

[6] TTL —*Time To Liv*e, que em inglês significa *Tempo De Vida.*

[7] Este tipo de objetos permite controlar por exemplo, grandezas relativas a fluxos de dados, pacotes descartados, etc...

Acerca duma ligação física há alguns aspectos a salientar:

- as filas não são todas iguais. Há filas que se limitam a descartar o último pacote que chega em caso de saturação, enquanto que outras, mais complexas, procuram garantir alguma justiça na repartição da largura de banda. A fila é selecionada no momento da criação da ligação física;
- todos os objetos que compõem uma ligação física possuem um canal alternativo para onde enviam pacotes que são dados como perdidos (por estar a fila cheia ou por excederem o TTL, por exemplo). Esse canal é conhecido por *drop-target* e, por omissão, está ligado a um agente `Null`, que descarta tudo o que recebe;
- podem ser adicionados à fila um conjunto de objetos de monitorização das várias operações que acontecem. Como a fila tem um canal de entrada e dois de saída podem ser observadas todas as operações de entrada na fila, saída da fila e operações de descarte, adicionando agentes a esses canais. A sequência destas operações pode inclusivamente ficar registada em ficheiro, situação esta de máximo interesse, uma vez que estamos a falar dum simulador de redes.

Note-se que os objetos representados na figura A.3 (exceto os nós, obviamente) estão. todos incluídos num objeto único do tipo `SimpleLink` (ou de tipo derivado) que é definido em OTcl.

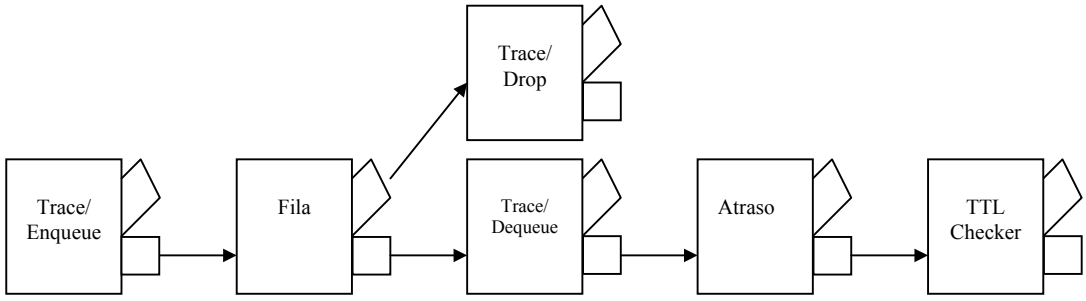

Figura A.4: Ligação física com monitores incluídos

No manual do ns-2 são abordados os métodos que permitem adicionar objetos de monitorização e de verificação às operações feitas em toda a ligação. A figura A.4 re-

presenta uma ligação com alguns desses objetos já incluídos. Neste contexto, "interface de saída" de uma ligação será, geralmente, o nó de destino, enquanto que "interface de entrada" será o primeiro objeto existente na cadeia.

Resulta disto, que os classificadores de endereços dos nós não enviam os pacotes diretamente para os outros nós, mas antes, para as ligações. Daqui resulta que há que ter algum cuidado com o momento em que se faz a adição de um objeto adicional no início da ligação. Se o nó e a ligação já estiverem conectados, um objeto que seja inserido antes da ligação pode ficar fora do percurso.

A.2.4 Endereços

Como é evidente, tem que existir uma regra de endereçamento global a toda a rede que se está a simular.

O endereçamento deve ser hierárquico, isto é, a partir de um determinado endereço deve ser possível conhecer o nó especificado nesse endereço e, dentro desse nó, qual o agente em particular.

Por omissão, um endereço tem 16 dígitos binários, representando os 8 dígitos binários mais significativos o nó e os outros 8 o porto. Estas dimensões limitam o número de nós e de agentes possíveis a 256 sendo, porém,o número de nós extensível a 2^{22}, usando-se, para o efeito, endereços com 30 dígitos binários.

Os objetos classificadores têm duas variáveis, `shift_`e `mask_`, que permitem separar com operações binárias básicas as duas componentes presentes num endereço. Este tipo de operações aparece repetidamente no código (quer compilado, quer interpretado).

É ainda importante referir dois aspectos:

- cada nó tem uma identificação única, que vai desde 0 até N-1, sendo N o número de nós existentes na simulação. Esta identificação coincide como o endereço do nó;
- o endereço do agente envolvido na comunicação costuma surgir nos ficheiros que resultam opcionalmente duma simulação, em forma de endereço do nó seguido do endereço do agente, mas separados por um ponto (por exemplo, 3.2 significa nó 3, porto 2).

A.3 Dinâmica do Simulador

De entre os objetos que temos vindo a mencionar e respectivas classes onde são definidos, destacamos duas classes chave na arquitetura do ns-2: as classes `classifier` e `connector` (que por vezes designaremos como "classificador" e "conector", respectivamente).

Em termos simples, os classificadores e os conectores são elementos que se ligam entre si e que, no seu conjunto, constituem a arquitetura da rede que se quer construir. Com efeito, comecemos por olhar para os grandes blocos constituintes duma rede e para os objetos que os compõem:

- Nó: classificador de endereços e classificador de portos — ambos classificadores;
- Ligação física: fila, objetos de monitorização, atraso, TTL *checker* — todos conectores.

Feita esta breve análise e sabendo que os próprios agentes são também eles conectores, constatamos que um pacote, quando segue através de nós e ligações, isto é, em todos os pontos do seu percurso, atravessa apenas classificadores e conectores, com excepção, eventualmente, da origem e do destino que podem ser aplicações.

Esta concepção é simples, mas flexível, na medida em que dá ao projetista liberdade para adicionar e subtrair todas as partes de que necessita ou prescinde de uma forma modular e transparente para as outras partes da rede.

Claro que, para ser possível adicionar partes de forma transparente tem que haver uma "entrada comum" a todos os classificadores e conectores. Essa *entrada* é a função recv() duma classe base de ambas, que é invocada sempre que o objeto em questão deva receber um pacote.

A ideia é, quando, em algum ponto da configuração da rede, se instala o conector B a jusante do conector A, por exemplo, A sempre que quer entregar um pacote ao conector B faz B->recv(). Aplica-se o mesmo princípio a classificadores, pese embora o fato de a jusante dum classificador poder haver múltiplos objetos como aliás dever ser evidente da figura A.2.

Note-se que as classes `Classifier` e `Connector` são classes básicas abstratas a partir das quais devem ser construídas classes específicas para os objetos concretos a introduzir na rede.

A.3.1 Conector

Começamos por descrever o objeto mais simples dos dois. Um conector tem uma entrada e duas saídas. Uma destas saídas é a saída normal *(target)*, enquanto que a outra é a saída de descarte *(drop-target)*. É nestas saídas, algures num conector da rede, que se podem perder pacotes.

No caso do conector, julgamos apropriado apresentar no fragmento de código A.1 o código fonte relativo à declaração dos métodos e das variáveis da classe, que se encontram no ficheiro "connector.h". Desta declaração, destacamos as seguintes variáveis e os seguintes métodos:

Fragmento de Código A.1 Classe `Connector`

```
class Connector : public NsObject {
public:
        Connector();
        inline NsObject* target() { return target_; }
        virtual void drop(Packet* p);
protected:
        int command(int argc, const char*const* argv);
        void recv(Packet*, Handler* callback = 0);
        inline void send(Packet* p, Handler* h) {
            target_->recv(p, h);
        }
        NsObject* target_;
NsObject* drop_;     // drop tar-
get for this connector
};
```

- `target()`: devolve o objeto a jusante do conector;

- `drop()`: envia o pacote pela saída `drop` . Se esta não estiver definida descarta-o;
- `recv()`:método invocado quando o objecto a montante passa um pacote para este objeto;
- `send()`: usado para enviar um pacote pela saída normal `(target )`;
- `target_` e `drop_`: objetos a jusante na saída normal e de descarte, respectivamente. Instalados com o método `command()`, que, além destas, inclui outras funcionalidades.

Os agentes e as filas são exemplos de objectos derivados dos conectores básicos.

A.3.2 Classificador

O classificador é um objeto mais complexo do que o conector. Para começar, o classificador tem, normalmente, mais do que duas saídas, podendo este número variar dinamicamente.

Isto significa que o classificador tem que ser capaz de determinar por qual das suas saídas deverá ser enviado um pacote. Isto implica decompor o pacote e verificar qual a saída correta ou, dito de outra forma, qual o próximo salto do pacote (que pode ser o classificador de portos dentro do mesmo nó. Vide figura A.2).

Constatamos, então, que aos classificadores está reservado o papel do encaminhamento, uma vez que são eles que separam os pacotes com base na informação relativa ao destino (e, eventualmente, origem e identificação de fluxo) que estes têm indicado nos seus cabeçalhos.

Claro que, antes de poderem fazer encaminhamento, os classificadores têm que ser configurados. Esta configuração também é abordada neste texto, mas é deixada para depois, já que constitui um assunto à parte.

A declaração da classe genérica Classifier é apresentada no fragmento de código A.2. Note-se que, como já tínhamos referido, também aqui, existe um método `recv()`, com o mesmo significado que tinha no objeto `Connector`.

No contexto desta classe, *slot* designa uma posição dentro de um vetor onde são armazenadas as saídas para os objetos a jusante do classificador.

Os métodos e variáveis mais importantes desta classe são:

- `nslot_` e `maxslot_` : respectivamente, número de *slots* ocupados e número de *slots* existentes no classificador;
- `maxslot()`: devolve o número máximo de *slots* existente;
- `slot(x)`: devolve o objeto a jusante, presente no *slot* x;
- `shift_` e `mask_`: respectivamente, deslocamento e máscara usados para determinar o endereço dum nó a partir dum endereço completo;
- `mshift(val)`: devolve o nó codificado no endereço `val`;
- `classify()`: esta é uma das funções chave da classe. Determina o número do *slot* onde se encontra a saída para um dado pacote;

Fragmento de Código A.2 Classe `Classifier`

```
class Classifier : public NsObject {
 public:
        Classifier();
        ~Classifier();
        void recv(Packet* p, Handler* h);
        int maxslot() const { return maxslot_; }
        inline NsObject* slot(int slot) {
            if ((slot >= 0) || (slot < nslot_))
                return slot_[slot];
            return 0;
        }
    int mshift(int val) { return ((val >> shift_) & mask_); }
        NsObject* find(Packet*);
        virtual int classify(Packet *const);
protected:
        void install(int slot, NsObject*);
        void clear(int slot);
        int getnxt(NsObject *);
        virtual int command(int argc, const char*const* argv);
        void alloc(int);
```

```
            /* table that maps slot number to a NsObject */
        NsObject** slot_;
        int nslot_;
        int maxslot_;
        int offset_;            // offset for Packet::access()
        int shift_;
        int mask_;
};
```

- `find(p)`: devolve o objeto a jusante para onde deverá ser enviado o pacote p;
- `install(slot, objeto)`: instala o objeto `objeto` no *slot* `slo`t;
- `clear()`: limpa um *slo*t.

Destas funções, interessa-nos, em particular, a função `find()`, que é invocada pela função `recv()`, para determinar o destinatário do pacote[8]. Esta função está definida no fragmento de código A.3. O trabalho de descobrir o *slot* correcto para onde há-

Fragmento de Código A.3 Funcao `Classifier::find()`

```
NsObject* Classifier::find(Packet* p)
{
        NsObject* node = NULL;
        int cl = classify(p);
        if (cl < 0 || cl >= nslot_ || (no-
de = slot_[cl]) == 0) {
                /*
                * Sigh. Can't pass the pkt out to tcl becau-
se it's
                * not an object.
                */
                Tcl::instance().evalf("%s no-slot %d", na
me(), cl);
/*
                * Try again. Maybe callback pat-
ched up the table.
        */
                cl = classify(p);
                if (cl < 0 || cl >= nslot_ ||
                    (node = slot_[cl]) == 0)
                        return (NULL);
        }
        return (node);
}
```

[8] Note-se que o classificador é uma mera passagem para o pacote, isto é, um pacote que entra é, imediatamente, reencaminhado para uma das saídas.

de ser enviado o pacote é feito pela função `classify()`, mas esta só é definida para subclasses concretas da classe `Classifier`, sendo inclusivamente definida como virtual. Esta opção é, aliás, natural, porque a decisão do próximo salto a seguir por um pacote pode depender de vários fatores: só do endereço de destino, como no caso das redes tradicionais TCP/IP, por exemplo, ou de combinações várias de três elementos: endereço de origem, endereço de destino e identificação de fluxo (que doravante designaremos por *fi*d).

A primeira classe de classificadores chama-se de endereços e já temos vindo a men-cionála múltiplas vezes[9]. A segunda classe de classificadores, chamados de classificadores de *has*h, permite efectuar classificação com base nos seguintes elementos:

- origem, destino e *fi*d;
- origem e destino;
- *fi*d;
- destino (resultado igual ao que é obtido com os classificadores de endereços, embora internamente seja construído de maneira diferente, uma vez que inclui uma tabela de *has*h).

Note-se que, em todos os tipos de classificadores, a situação normal será ter na posição *(slo*t), que será escolhida para enviar um pacote para um dado destino, o nó seguinte, que corresponde ao próximo salto do percurso.

Classificador de Endereços

Este classificador é relativamente simples. O próximo salto para o nó com endereço *i*, ocupa o *slot i*. O objeto colocado no *slot* correspondente ao endereço do próprio nó é o classificador de portos.

[9] Também é usada para o classificador de portos. Para isso são alteradas a máscara e o deslocamento.

Classificador de *Hash*

Existem quatro classificadores deste tipo: um para cada uma das combinações atrás apresentadas. Aqui, os *slots* vão sendo ocupados por ordem de introdução na tabela, sendo o trabalho de classificação deixado para uma tabela de *hash* adicional. O tamanho desta tabela é determinado no momento da criação do classificador.

A.4 Encaminhamento

A.4.1 Coexistência de Diversos Protocolos de Encaminhamento

Como seria à partida de esperar dum simulador de rede, o ns-2 suporta a configuração de diferentes protocolos de encaminhamento. Inclusivamente, como teremos oportunidade de ver, o ns-2 suporta a coexistência (na mesma simulação e até no mesmo nó) de diferentes protocolos de encaminhamento, sendo, para isso, possível especificar os protocolos a utilizar em cada nó.
Os objetos e respectivas classes no simulador que permitem alcançar esta coexistência estão descritos com algum detalhe no manual. Aqui vamos apenas revê-los rapidamente.

Classe `RouteLogic`

Resumidamente, esta classe fornece, entre outras, as funcionalidades de uma tabela de encaminhamento global a toda a simulação. Isto porque esta classe inclui um método (lookupfg{}), que dados dois nós, nó1 e nó2, por esta ordem, permite determinar o vizinho que nó1 utiliza para enviar um pacote para nó2.

Nalguns protocolos de encaminhamento — ditos estáticos — é o resultado do cálculo efetuado com lookup{} introduzido nos classificadores dos nós. Noutros

protocolos — ditos dinâmicos — a situação é algo mais complexa, como teremos ocasião de verificar.

É importante referir que numa simulação não existe mais do que um objeto desta classe.

Classe `rtObject`

Para podermos usar um protocolo dinâmico em vez dum protocolo estático, de forma a tornar o simulador mais realista[10] , existe a classe rtObject. Esta classe tem um função dupla, na medida em que também é ela que permite a um nó utilizar mais do que um algoritmo de encaminhamento dinâmico.

Ao contrário do que acontecia na classe anterior, existem múltiplos objetos desta classe. Mais exatamente, existe um objeto em cada nó que utiliza um protocolo de encaminhamento dinâmico.

[10] É possível usar um protocolo baseado em vetores de distâncias (DV — *Distance Vector* ou nos esta-dos das ligações (LS — Link State)

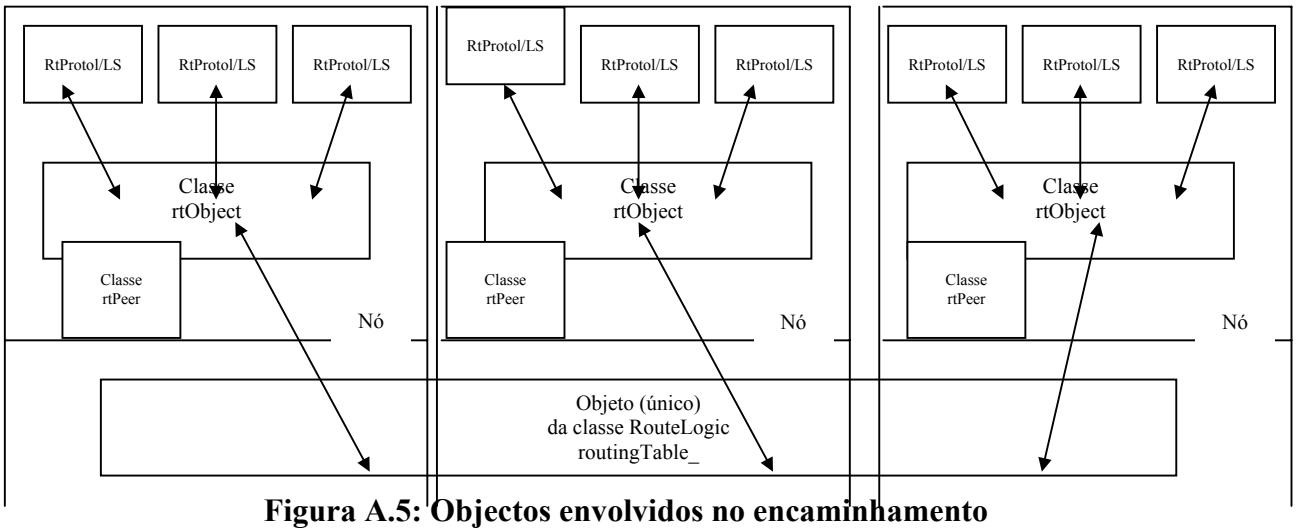

Figura A.5: Objectos envolvidos no encaminhamento

Outras classes

Além destas duas classes, existem mais duas, que são relevantes para a compreensão do código relativo ao encaminhamento:

- classe `rtPeer`: existe um objeto deste tipo em cada nó. Guarda informação sobre os vizinhos diretos do nó em questão. Possui um conjunto de métodos que permitem alterar ou consultar valores das ligações físicas existentes com esses vizinhos.
- classe `Agent/rtProto`: esta é a classe base para todas as classes que definem protocolos de encaminhamento. Por exemplo, os protocolos DV e LS são definidos, respectivamente, nas classes `Agent/rtProto/DV` e `Agent/rtProto/LS`. Esta classe possui um conjunto de métodos que devem ser sobrepostos pelas classes derivadas e que são invocados, normalmente, pelo objeto da classe `rtObject` existente no nó. é suposto também, que os protocolos tenham definidas algumas variáveis que teremos oportunidade de apresentar, acessíveis pela classe `rtObject`.

A figura A.5 pretende ilustrar graficamente as relações existentes entre os objetos existentes nestas classes. A classe `rtProto/Direct` será apresentada mais adiante.

A.4.2 Criação dos Objetos Necessários ao Encaminhamento

O momento em que os diversos objetos apresentados na figura A.5 são criados difere.

O primeiro objeto a ser criado é `aroutingTable` , que é um objeto da classe `RouteLogic`, incluído na classe `Simulator`. Este objeto pode ser criado em dois momentos distintos. Se existir, no código que define a configuração da rede, uma chamada ao procedimento `rtproto` da classe `Simulator`, como por exemplo

```
$ns rtproto LS
```

então, a criação é feita imediatamente.

Se não (e será o caso que se dá quando é usado um protocolo de encaminhamento estático) a criação é protelada para o momento em que a simulação se inicia (procedimento `run` da classe `Simulator`). Em qualquer dos casos, a criação da `routingTa-ble` é feita por um procedimento chamado `get-routelogi{}` da classe `Simulator`.

Outra das tarefas do procedimento `rtproto` é o de registar todos os nós onde o protocolo deve ser instalado. Para isso a classe `Simulator` tem uma variável vectorial, `rtprotos_`, que é indexada pelos nomes dos protocolos e que armazena em cada posição uma lista de todos os nós onde o protocolo deve ser instalado.

Já depois de criada a `routingTable` é invocado o procedimento `configur{}` da classe `RouteLogic` (também no procedimento `Simulator run`), que percorre todo o vetor `rtprotos` , se este existir, e invoca para cada protocolo que indexa este vetor, o procedimento estático de inicialização respectivo, chamado `init-all{}`. `Is`to permite aos diversos protocolos de encaminhamento fazer as suas inicializações globais. Se o protocolo em causa for o LS, o protótipo deste procedimento será:

```
Agent/rtProto/LS proc init-all args
```

Em `args` é passada a lista dos nós onde deve ser instalado o protocolo, lista essa que é o valor armazenado no vetor `rtprotos` . Dito de outra forma a lista indica em que nós é que devem ser criados os agentes desse protocolo.

É de salientar que o procedimento estático `init-all{}` não faz a criação do agente. Este procedimento começa por invocar o procedimento estático homónimo da classe `rtObject`, que para cada um dos nós fornecidos em `args` vai instalar um objeto da classe `rtObject` se este ainda não existir.

Depois disso, e ainda no procedimento `init-all {} do rtObject`, são calculados e instalados os caminhos mais curtos através duma chamada ao procedimento `compute-routes` do `rtObject` (voltaremos a estes assunto mais adiante). Note-se que o mais que se consegue saber neste ponto é o caminho para os vizinhos imediatos, através do protocolo *Direct*,até porque os próprios agentes protocolares podem não existir, já que só são criados a seguir, além dos agentes protocolares de outros protocolos, que podem ainda nem ter sido inicializados com o mesmo procedimento `init-alg`.

Terminado o `init-all{}` do `rtObject`, `oinit-all{}` do agente vai criar um agente em cada um dos nós da lista, com o procedimento `add-proto{}` do `rtOb-ject`, já garantidamente existente em cada nó. As tabelas A.1 e A.2 resumem as sequências de procedimentos que acabámos de descrever. A primeira das duas tabelas expõe a sequência que ocorre na fase de configuração, enquanto que na segunda tabela está descrita a sequência que ocorre já no procedimento `run` do simulador (classe `Simulator`).

No arquivo de configuração da rede	`$ns rtproto nome`
`Simulator rtproto ...`	`[$self get-routelogic] register nome`
`Simulator get-routelogic...`	Devolve a `routingTable` ou cria-a se não existir.
`RouteLogic regis-ter...`	Adiciona o protocolo de nome `(nome)` ao vetor `rtpro-tos`, de forma que `rtprotos_(nome) =` lista dos nós onde deve correr o protocolo `nome`.

Tabela A.1: Criação dos objetos de encaminhamento — fase de configuração.

`RouteLogic confi-gure`	`foreach p [array names rtprotos ]{`
	`Agent/rtProto/$p init-all $rtprotos ($p)`
	`}`
	Para todos os protocolos chama-se o procedimento

	`init-all{}`.
`Agent/rtProto/nome proc init-all`	eval rtObject init-all $nodes-list; [$no-de rtObject?] add-proto nome $node Chama o procedimento init-all{} da classe rtOb-ject. Para cada nó adiciona um objeto do protocolo designado por nome.
`rtObject proc init-all args`	Instala em cada nó (presente na lista `args`) um novo objeto `rtObject`. Chama `compute-routes` para instalar os caminhos mais curtos.
`rtObject instproc proc add-proto proto no`	Cria um novo agente do protocolo dado no nó dado.

Tabela A.2: Criação dos objetos de encaminhamento —fase de execução

A.4.3 Inicialização

Depois de criados todos os objetos necessários ao encaminhamento, o processo não fica terminado, porque há, ainda, que calcular os caminhos e introduzir os resultados nos classificadores dos nós. Antes desse momento não é possível à rede funcionar em pleno, já que os nós desconhecem o caminho para a maior parte dos destinos, excetuando os seus vizinhos.

Se voltarmos atrás à figura A.1, verificamos que na sequência nó-ligação-nó, só o último elo está ligado, uma vez que a maior parte das saídas dos nós ainda permanecem em aberto.

É importante referir a existência dum protocolo chamado *Direc*t, que é instalado em todos os nós, sempre que é criado um `rtObject`. Mais precisamente, é dentro de cada construtor da classe `rtObject` (e como j á vimos existem múltiplas instâncias desta classe), ou seja, dentro do procedimento `init`, que é criado o agente `Agent/rtProto/Direct`, para instalar no mesmo nó onde vai ser instalado o objeto `rtObject`. Estes agentes conhecem sempre o caminho para os vizinhos diretos do nó em que são instalados, pelo que o nó apresenta sempre conectividade, pelo menos, com os seus vizinhos. Este fato é essencial, para que protocolos de encaminhamento distribuídos como são o DV e o LSpossam iniciar os procedimentos necessários ao seu funcionamento, que passam por enviar mensagens para os seus vizinhos diretos.

Vimos já, então, que o cálculo dos caminhos feito por cada `rtObject` é efectuado logo após a sua criação. Nesta fase, só são considerados os caminhos para os

vizinhos obtidos a partir dos agentes *rtProto/Direc*t. Veremos de seguida como é que são depois considerados os outros protocolos de encaminhamento, quando é que os caminhos são recalculados e onde é que os classificadores dos nós são preenchidos. Para fazer essa descrição e sempre que acharmos necessário particularizar vamos usar o protocolo LS.

A.4.4 Cálculo dos Caminhos

Protocolos de encaminhamento como o LS e o DV usam trocas de mensagens para passarem informação entre agentes existentes em cada nó. Sempre que chega uma nova mensagem dirigida a um destes agentes, pode haver informação relevante que resulte em alterações nos cálculos do encaminhamento.

No caso do protocolo LS, onde são trocadas informações relativas à topologia da rede isso pode acontecer se houver alguma alteração a essa topologia. Nesse caso, têm que ser recalculados os caminhos para todos os destinos. Outra situação em que há necessidade de recalcular esses caminhos ocorre quando há alguma alteração numa interface local.

Mais adiante, apresentaremos, para o LS, a sequência de funções/procedimentos que são invocados quando acontece uma destas duas situações.

Quando um dos protocolos recalcula os caminhos mais curtos num determinado nó, o `rtObject` desse nó tem que efetuar também os seus próprios cálculos, de forma a poder instalar os resultados no classificador do nó. O procedimento onde são feitos esses cálculos chama-se `rtObject instproc compute-route`g. Os resultados da execução desses cálculos são armazenados em quatro vetores, existentes nos objetos da classe `rtObjec`t, indexados pela identificação dos nós. São eles:

- `nextHop_(x)`:próximo salto para o destino x;

- `rtpref_(x)`:número que indica o ”grau de preferência” do nó em utilizar o próximo salto em questão, para o destino x;

- `metric_(x)`:número que representa a distância, segundo uma dada métrica (do protocolo vencedor, como veremos) para o nó de destino x. Note-se que a preferência é mais importante do que a métrica;

- `rtVia_(x)`: protocolo a partir do qual foi escolhido o próximo salto para o destino x.

Cada protocolo de encaminhamento utiliza também vetores iguais a estes com as suas próprias informações. A informação destes vetores é introduzida nos procedimentos onde os agentes de cada protocolo efetuam os seus próprios cálculos. Sendo assim, o objeto rtObject, no procedimento compute-routesfg, vai percorrer os vetores de todos os protocolos em busca do que oferece as condições mais favoráveis de encaminhamento para um determinado destino.

Selecionado o protocolo que oferece essas condições, preenche as entradas para esse destino nos seus próprios vetores copiando, para isso, os valores presentes nos vetores homónimos do agente vencedor. O único vetor que é preenchido de forma diferente é `ortVia_`, onde é armazenado o nome do protocolo cujas informações estão a ser utilizadas para fazer o encaminhamento desde o nó em questão para o destino que estava a ser calculado.

Sempre que é terminado o cálculo relativo a um destino, ainda no procedimento `rtObject compute-route{},` o nó vizinho a utilizar para atingir esse destino é armazenado no classificador com o procedimento `Node instproc add-routeg`. Se, por ventura, já existisse informação antiga e que entretanto tenha ficado desatualizada é utilizado primeiro o procedimento `Node instproc delete-routeg`, para eliminar essa informação. Neste ponto ficam completos os circuitos nó-ligação-nó, ficando a rede pronta para a simulação.

A.5 *Link State Routing*

A.5.1 Agente rtProtoLS

O protocolo LS não é distribuído conjuntamente com o simulador. A versão existente deste protocolo foi criada a partir do agente do protocolo DV, por Mingzhou Sun[11].

Como muitos dos objectos existentes na arquitectura, os agentes do protocolo LS têm uma existência dual, sendo uma parte definida em C++ e outra em OTcl.

Em OTcl, a classe que define estes agentes chama-se Agent/rtProto/LS, como tivemos já ocasião de referir diversas vezes. Em C++, o nome da classe é `rtProtoLS`.

Para começar, interessa-nos conhecer a composição em termos de estruturas de dados destes agentes, pelo que apresentamos a definição parcial da classe em C++, no fragmento de código A.4. Note-se que esta classe descende da classe `Agent`. As variáveis mais importantes são:

- `peerAddrMap`: lista dos endereços dos nós vizinhos;
- `nodeId`: identificação (m.q. endereço) do nó em que está instalado o agente;
- `linkStateList`: estado (inclui o custo) das ligações do nó;
- `peerIdList`: lista das identificações dos nós vizinhos;
- `delayMap`: lista dos atrasos estimados das ligações físicas para os vizinhos;
- `routing`: esta variável é o coração do agente. Inclui, entre outros, os seguintes dados: topologia completa da rede, lista das mensagens protocolares recebidas, tampão para mensagens a enviar, além duma tabela de encaminhamento. Como facilmente se depreende é também este objeto o responsável pela gestão de mensagens e pelo cálculo dos caminhos mais curtos para todos os nós.

[11] À data em que este documento foi escrito o seu endereço de correio eletrônico era `sunmin@networks.ecse.rpi.edu`

Aapresentação completa da definição da classe `LsRouting` seria de todo o interesse para a correta compreensão do funcionamento do agente, mas é de tal forma extensa que não pode ser incluída neste texto.

A.5.2 Inundação

Como já dissemos, a recepção e o envio de mensagens relativas ao protocolo LS são da responsabilidade do objeto da classe LsRouting — que é uma variável de nome routing incluída no agente rtProtoLS.

Porém, tem que ser compreendido que quem recebe e envia mensagens são os agentes pelo que a variável routing recebe as mensagens por via do agente (função `receiveMessage()` do agente) e envia-as invocando a função `sendMessage()` do agente (inclui para esse efeito um ponteiro para o agente). Fora estas funções, incluídas no agente da classe `rtProtoLS`, quase todo o trabalho relevante no que diz respeito à inundação é feito na classe `LsRoutin`g.

Um agente do protocolo LS pode receber três tipos de mensagens: atualizações dos estados das ligações, topologias completas e confirmações. Neste texto vamos ignorar os detalhes relacionados com estas últimas.

Atualizações dos Estados das Ligações

Primeiro, quando arranca e depois periodicamente[12] cada nó envia a todos os seus vizinhos o estado das suas ligações físicas. Cada vizinho, ao receber uma destas mensagens, começa por verificar se já tinha tomado conhecimento dela (para isso as mensagens têm um número de sequência). Em caso afirmativo, descarta-a. Caso ainda não conheça a mensagem, propaga-a a todos os vizinhos excepto aquele de onde ela provém e excetuando, também, o nó que a criou, se for caso disso. Note-se que este mecanismo permite difundir a mensagem por toda a rede, não existindo, apesar disso, a possibilidade de a mensagem andar em círculos.

[12] Este período é, por norma, bastante elevado, na ordem dos 30 minutos, por exemplo.

É interessante referir que as mensagens são propagadas para os vizinhos todas ao mesmo tempo. Isto significa que sempre que uma destas mensagens tem que ser enviada a, por exemplo, três vizinhos, as diversas cópias vão sendo colocadas num tampão sendo este despejado no fim.

Obviamente que os nós têm que ir coleccionando estas mensagens, de forma a construírem uma base de dados (BD) que lhes permita conhecer toda a topologia da rede.

Em termos de funções temos: `LsRouting::receiveLSA()` — chamada quando o agente recebe uma atualização; `LsRouting::sendLinkState()` — enviar uma atualização.

Topologias Completas

A diferença desta mensagem, em relação à anterior é que, nesta, são transmitidos todos os estados de todas as ligações conhecidas pelo nó que gera a mensagem.

Esta mensagem pode ser gerada quando uma ligação física que estava em baixo ressurge. Cada um dos nós nos extremos dessa ligação envia uma destas mensagens para o outro numa situação destas (`LsRouting::sendTopo()`). Ao receber uma destas mensagens (função `LsRouting::receiveTopo()`), o nó começa por verificar se é a primeira vez que a recebe. Depois disto, percorre toda a mensagem que é composta por uma lista de estados das ligações de cada nó, verifica o número de sequência presente em cada uma destas listas e atualiza a sua BD sempre que alguma desta informação for mais recente que a informação conhecida até ao momento.

No fim de fazer a atualização da sua BD, o nó propaga a mensagem pelos seus vizinhos, de acordo com uma regra idêntica à que é usada na inundação das mensagens de atualização, isto é, envia para todos menos para o vizinho de quem a recebeu e para o nó que a gerou, se for caso disso (função `LsRouting::regenAndSend()`). Neste caso, as mensagens não são previamente armazenadas num tampão, sendo imediatamente enviadas.

Em resumo, para gerir estes tipos de mensagens são usadas as seguintes funções:

- `sendTopo()`: gerar mensagens com topologia;

- `receiveTopo()`: receber mensagens com topologia;

- `regenAndSend()`: propagar mensagens com topologia.

A.5.3 Inicialização e Reacção às Alterações

Inicialização

Como já descrevemos, quando falámos da inicialização dos protocolos de encaminhamento em geral, o LS, quando se inicia não tem conhecimento de qualquer caminho, começando com todas as estruturas vazias. Isso aliás é explicitamente feito no construtor do objecto `Agente/rtProto/LS`, onde os vectores `rtpref_`, `nextHop_` e `metric_` são inicializados, como está representado no fragmento de código A.5. Na criação destes agentes, o último passo consiste em preparar o envio da primeira mensagem com o estado das ligações do nó. Esse envio que é feito no procedimento `send-periodic- update {}` do OTcl é protelado para um instante escolhido aleatoriamente entre os 0 e os 0,5 segundos. Não é feito imediatamente para evitar a enchente de mensagens que resultaria de todos os nós serem ativados em simultâneo.

Depois de o agente ser criado e anexado a um nó é iniciado um vetor com a lista de todos os vizinhos do nó (os elementos desse vector são objectos da classe `rtPeer`).

Alteração nas Interfaces Locais

Quando há uma alteração nas interfaces são chamadas as funções/procedimentos enumerados na tabela A.3. Em parêntesis indicamos se se trata de um método do C++ ou do OTcl.

Recepção duma Mensagem

Outra situação que pode resultar em alterações às decisões de encaminhamento é a recepção duma mensagem com informações de alterações dos estados das ligações.

`rtObject instproc intf-changed`	• Chama `intf-changed` de todos os protocolos; • chama `$self compute-route`s; • Envia mensagens que ficaram no tampão, no protocolo LS.
`Agent/rtProto/LS instproc intf-changed`	• Atualiza estado das interfaces armazenado no agente (necessário para construir um mapa interno da topologia); • chama função `command()` do C++ para chamar `intf-Changed()`.
`rtProtoLS::-command()`	Chama `intfChanged()`.
`rtProtoLS::-intfChanged()`	• Obtém o estado das interfaces; • envia o estado da interface para os vizinhos com `rou-ting.linkStateChanged();` • chama `route-change` do OTcl.
`Agent/rtProto/LS instproc route-changed`	• `$self install-routes` (ver a seguir); • `$rtObject compute-routes` ´ (ja visto: vai calcular primeiro e preencher, depois, nos classificadores as alterações produzidas se for caso disso).
`Agent/rtProto/LS instproc install-routes`	• Chama Preenche os vetores `nextHop` , `metric` e `rt-pref` ; • para determinar o `nextHop` usa o procedimento `loo-kup{}`

Tabela A.3: Sequência de procedimentos após alteração nas interfaces locais.

`rtProtoLS::recv()`	Chama `receiveMessage()`.
`rtProtoLS::-receiveMessage()`	• `routing.receiveMessage();` • se há novidades a registar chama a função `install-Routes()`.
`routing.-receiveMessage()`	Se há alterações são calculados os caminhos.
`rtProtoLS::-installRoutes()`	Chama `route-changed()` do OTcl.

Tabela A.4: Sequência de procedimentos após recepção duma mensagem com informação acerca do estado da rede

algures na rede. Nesse caso são chamadas as funções e procedimentos, na sequência presente na tabela A.4. Da última função em diante a sequência é totalmente idêntica à do caso anterior. Note-se porém que, neste caso, não é despejado o tampão no fim, porque as mensagens são imediatamente propagadas para os vizinhos, como tivemos ocasião de explicar.

Fragmento de Código A.4 Classe `rtProtoLS`

```
class rtProtoLS : public Agent , public LsNode /* LS */ {
   ...
protected:
   ...
  void sendBufferedMessages () { routing.sendBufferedMessages();
}
  void computeRoutes() { routing.computeRoutes(); }
  void intfChanged ();
  void sendUpdates() { routing.sendLinkStates(); }
  void lookup(ls_node_id_t destinationNodeId) ;
  // ------------ Implements LsNode virtual methods ----------
-----
public:
   bool sendMessage (ls_node_id_t destId, ls_message_id_t messa-
geId,
          ls_message_size_t size );
  void receiveMessage (ls_node_id_t sender, ls_message_id_t ms-
gId);
   ...
  void installRoutes () {Tcl::instance().evalf("%s route-
changed", name());}
protected: // data
  // addr for peer Id
  typedef LsMap<ls_node_id_t, nsaddr_t> PeerAddrMap;
  PeerAddrMap peerAddrMap;
  int LS_ready; // to differentiate fake and real LS , de-
bug,0 == no
  // needed in recv and sendMessage;
protected: //method
  ls_node_id_t findPeerNodeId( nsaddr_t agentAddr);
  protected: // data for LsNode methods
  ls_node_id_t nodeId;
  LsLinkStateList linkStateList;
  LsNodeIdList peerIdList;
  LsDelayMap delayMap;
  LsSourceRouting routing;
};
```

Fragmento de Código A.5 Construtor do ojbecto `Agente/rtProto/LS`

```
foreach dest [$ns all-nodes-list] {
   set rtpref_($dest) $preference_
   set nextHop_($dest) ""
   set nextHopPeer_($dest) ""
   set metric_($dest) $UNREACHABLE
}
```

Anexo B

Network Simulator-2

Orientado às ligações

Como tivemos já ocasião de dizer, o ns-2 é um simulador para redes sem ligações. No entanto, este tipo de redes sem mais alterações não se adequa ao teste de algoritmos capazes de fazer encaminhamento com base em parâmetros de qualidade de serviço (QoS — *Quality of Servic*e). Para resolver o problema, julgou-se apropriado adicionar um módulo ao ns-2 de forma a transformá-lo num simulador capaz de suportar redes orientadas às ligações.

Neste capítulo vamos descrever os elementos que adicionamos ao ns-2, de forma a conseguir a transformação desejada.

B.1 Modelo do Nó

O modelo atual do nó, representado na figura A.2, não é suficiente numa rede que pretenda oferecer garantias de QoS. O problema essencial deste tipo de nó é que não permite estabelecer uma reserva de recursos, com todos os problemas que essa situação pode acarretar.

Para obviar a esse problema, apresentamos, nesta solução um modelo alternativo composto por vários elementos que adicionamos à arquitetura tradicional do ns-2:

- `ConnectedClassifie`r: este é um novo tipo especial de classificador que faz uma separação de pacotes conforme o seu tipo;

- agente `Setupe`r: incluído em cada nó, este agente é responsável por estabelecer as ligações. Para o fazer utiliza um protocolo que inclui mensagens próprias;

- CAC: o CAC — Controlo de Admissão de Ligações — é um módulo incluído, para já, no agente `Setuper`, responsável por admitir ou rejeitar novos pedidos de reservas. Utiliza, na realização dos seus cálculos informação proveniente de objetos chamados `SimpleLink`'s, mas com informação adicional de QoS (`QoSLink`'s);

- `QoSLink`:não fazendo parte do nó, este objeto é uma extensão aos `Simple-Link`'s normais. Numa fase posterior do trabalho deveria ser transformado numa classe autônoma, derivada da classe base `SimpleLink`. Contém informação de QoS relativa a uma ligação que, para já, é apenas a largura de banda reservada nessa ligação (além da largura de banda disponível)

- agente `rtProtoLsSource`: este agente é um objecto duma classe derivada da classe `rtProtoLS` normal. A nova classe adiciona duas características ao protocolo LS normal: encaminhamento pela fonte *(source routing*) e encaminhamento com QoS. A primeira das duas características é importante por uma razão: o estado das reservas efetuadas por um dado nó não é propagado de imediato para todos os nós vizinhos. Isto significa que os vários nós têm visões diferentes da rede, ou seja, mesmo usando todos o mesmo algoritmo, não seria possível garantir a inexistência de ciclos. Usando encaminhamento pela fonte, esse problema fica solucionado, embora o preço a pagar por isso seja a obtenção de soluções eventualmente subótimas.

A interação de todos estes componentes está ilustrada na figura B.1. À entrada, o nó tem um pré-classificador que faz uma separação prévia dos pacotes. Em função dos tipos de pacotes que chegam, três situações são possíveis:

- se o pacote foi enviado por agentes `Setuper`, entrega-o ao agente `Setuper` local. Caso tenha sido o agente local a gerar o pacote, entrega-o ao classificador de endereços tradicional (`classifier`). Mais adiante será apresentado o formato dos pacotes usados pelo agente `Setuper`. Para j á, interessa saber que há uma *flag* no pacote que indica se ele foi ou não gerado pelo agente local;

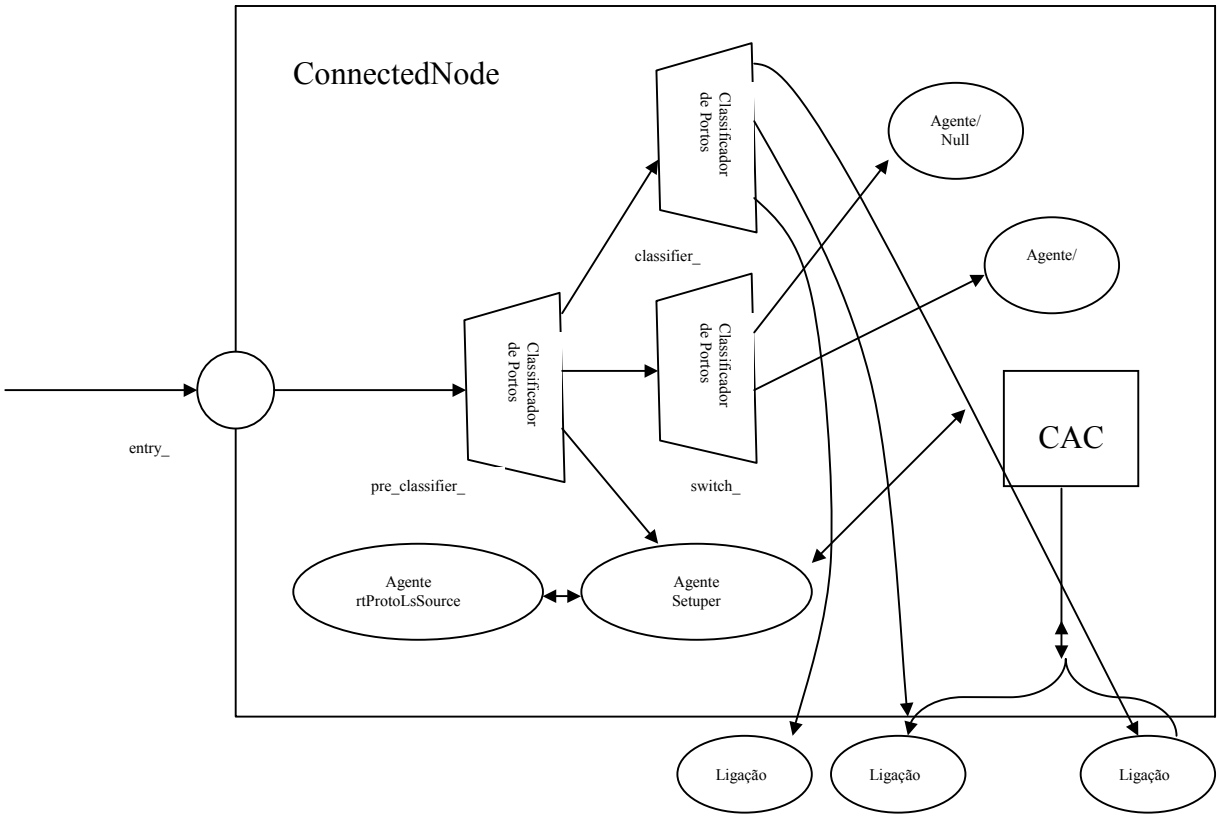

Figura B.1: Estrutura interna dum nó ligado (`ConnectedNode`)

- os pacotes gerados pelos protocolos de encaminhamento tradicionais (DV ou LS) são encaminhados para o classificador de endereços `classifier_`. Este, que é inicializado e mantido duma forma em tudo idêntica ao classificador de endereços dos nós comuns, mantém informação relativa à conectividade com toda a rede. Note-se que bastava manter informação de conectividade para os vizinhos do nó, mas o código existente não foi alterado por uma questão de simplicidade nesta fase de desenvolvimento. A ideia é que a configuração da rede se mantenha a funcionar precisamente da mesma forma, uma vez que continuam a existir os mesmo elementos. Simplesmente, agora, os agentes são ligados a um classificador diferente, de forma que este classificador nem sequer necessita de um classificador de portos na saída que corresponde ao nó local onde o classificador está instalado;

- se for recebido um pacote de um quarto tipo (isto é, cujo tipo não seja uma das constantes PT_SETUPER, PT_RTPROTO_DV ou PT_RTPROTO_LS) é encaminhado para o classificador `switch_` .`O switch_` é um classificador de *hash* baseado na origem, destino e *fi*d, como explicámos atrás. Este classificador mantém apenas registos relativos às ligações já estabelecidas. Sempre que surge um pacote cuja combinação {*origem, destino, fid*} não exista no classificador, é descartado ou, mais precisamente, é enviado pela saída `default` deste classificador que é um agente nulo, que a única coisa que faz é descartar o pacote.

Como facilmente se compreende só é possível enviar pacotes pertencentes ou a protocolos de controlo ou a ligações existentes. Neste modelo assumimos, para já, que estas ligações são bem comportadas e que o débito que enviam para a rede está de acordo com o que foi contratado. Aqueles fatos, em conjunto com esta assunção permitem concluir que uma rede assim construída pode oferecer garantias de QoS.

B.2 Estabelecimento das Ligações

O estabelecimento duma ligação com origem no agente `X` no nó (`ConnectedNod`e) `A`, com destino ao agente Y, no nó (`ConnectedNod`e) B, para o fluxo com identificação `f` e com uma largura de banda reservada `b` seria iniciada com a seguinte instrução (admitindo que A, B, X, Y, `f` e `b` são variáveis):

```
$A start-connection $B $X $Y $f $b
```

Nesta instrução interessa salientar dois aspectos:

- é invocado o procedimento `start-connection` do objeto A, que pertence à classe `ConnectedNod`e;

- a instrução tem que ser executada já durante a execução do simulador, isto é, não pode ser executada antes do procedimento `run` do objeto da classe `Simulato`r. Para o fazer utiliza-se o escalonador, pondo-a na lista de acontecimentos a despoletar num determinado instante de tempo. Teríamos, então, por exemplo:

```
$ns at 2 "$A start-connection $B $X $Y $f $b"
```

que executaria a chamada ao procedimento no instante de tempo $t = 2s$, já com o simulador em execução. Inclusivamente, é conveniente que as ligações só sejam iniciadas depois de o protocolo de encaminhamento LS/*Source* já ter atingido a estabilidade na rede, o que pode levar ainda alguns instantes.

Depois de iniciada a ordem de estabelecer uma ligação, a sequência de acontecimentos é a seguinte:

- é pedido ao agente `Setuper` local que estabeleça a ligação (comando `setup` invocado através de `Setuper::command()`;
- agente `Setuper` invoca o agente `rtProtoLsSource` para obter um caminho até ao destino (isso é feito dentro do procedimento `Agent/Setuper get-source-route{}`);
- agente `Setuper`, conhecendo já a ligação física de saída para o primeiro salto, invoca o CAC (procedimento `Agent/Setuper CA`g) para saber se é possível ou não (mais exactamente, se há recursos ou não) para estabelecer a ligação pretendida;
- se for possível, envia um pacote para o próximo salto propagando o pedido de estabelecimento da ligação. Note-se que neste pacote está incluída informação de todo o percurso ao longo do qual deve ser estabelecida a ligação;
- instala num classificador de *hash* existente no `Setuper` (chamado `forward`) a ligação de saída *(lin*k) para esta ligação[1]. Este classificador é um repositório onde são guardados dados temporários de ligações em curso;

- de volta ao procedimento `start-connection` é instalado no `switch` do nó de destino o agente de destino. Note-se que isto não é uma boa medida, uma vez que a ligação pode ser rejeitada ao longo do seu percurso. Porém, era o momento mais simples de o fazer. Este problema será corrigido em alterações futuras a efetuar no código.

Terminados estes passos, a ligação está pronta a ser iniciada no nó de origem e foi já enviado um pacote para o próximo salto, por onde deve ser estabelecida. A ligação só

[1] Aqui poderíamos ter usado a palavra “conexão”

será dada por concluída quando o agente receber um pacote de confirmação ou de rejeição. É importante referir que a ligação s ó pode ser confirmada pelo último nó, mas pode ser rejeitada por qualquer um ao longo do percurso através do qual deverá ser estabelecida.

Outro detalhe importante diz respeito ao caminho de volta a seguir pela confirmação ou pela rejeição: este tem que ser precisamente igual, nó a nó, ao caminho de ida. Para isso à medida que a mensagem vai avançando vai sendo construída uma lista com o caminho de volta.

Quando um nó intermédio (que não o primeiro nem o último, portanto), recebe um pedido de estabelecimento duma ligação realiza a seguinte sequência de ações:

- extrai a informação do próximo salto da ligação;
- verifica se a ligação é admitida pelo CAC e, em caso afirmativo, envia o pacote para o nó seguinte. Daqui em diante o procedimento é igual ao que já foi descrito para o nó inicial.

No entanto, se a ligação for rejeitada é seguida outra sequência:

- é retirado o registo da ligação do classificador `forward_`;
- é retirado da lista de retorno o nó anterior e é gerada uma mensagem de rejeição para esse nó.

Ao ser recebida uma rejeição, o comportamento dos outros nós a montante (note-se que esta mensagem circula no sentido inverso ao do estabelecimento da ligação) é semelhante ao que acabámos de descrever.

Se, pelo contrário, tudo correr bem, quando o nó de destino recebe um pedido de estabelecimento de ligação, aceita-a sempre (admitimos que não tem limites de recursos de computação) e gera uma mensagem de confirmação a enviar no sentido inverso.

Não tem que atualizar quaisquer estruturas internas, uma vez que o agente receptor j á está instalado.

Em todos os nós do percurso com excepção do último, se a ligação for aceite é calculada a razão entre a largura de banda reservada e a largura de banda anteriormente

disponível. Se esta razão ultrapassar os 10% é imediatamente gerada uma atualização do estado da ligação física a que se refere a reserva. No entanto, para não gerar uma sobrecarga na rede com informação deste tipo, não é possível a um agente `rtProtoLsSource` enviar dois anúncios com alterações dos estados das ligações separados por menos de cinco segundos.

Ainda por realizar neste momento estão as seguintes caracerísticas:

- envio dum anúncio (dentro do limite inferior de tempo que referimos), caso o CAC num determinado nó rejeite uma ligação, pois isto significa que há nós com informação desatualizada;
- envio dum anúncio quando é terminada uma ligação que liberta uma quantidade de recursos acima dum determinado limiar (10%, por exemplo, como no caso do estabelecimento das ligações);
- acumulação de recursos consumidos pelo estabelecimento de ligações, quando sucessivamente inferiores a 10%, para que seja gerado um anúncio logo que esse limiar seja ultrapassado.

B.3 Pacotes `SETUP`

Os pacotes trocados pelos agentes `Setuper` têm o formato listado de seguida. Na figura B.2 o formato destes pacotes está representado de uma forma alternativa sendo depois explicados os significados dos vários campos.

```
struct {
connection_phase_t phase;
nsaddr_t source;
nsaddr_t dest;
int fid;
int bw_required;
int catch_pack;.
}
```

`phase`	`source`	`dest`
`fid`	`bw-required`	`catch-pack`

Figura B.2: Formato do pacote de *setup*

- O primeiro campo indica em que fase se encontra a ligação: 1 — em estabelecimento; 2 — em terminação; 3 — em aceitação; 4 — em rejeição. Note-se que `connection phase t` é um tipo enumerado;

- `source`, `dest` e `fid` são, como facilmente se deduz, endereço de origem, endereço de destino e identificação do fluxo dos pacotes a usar durante a existência da ligação;

- `bw required` é a largura de banda pedida na ligação;

- `catch pack` é uma *flag* que indica se o pacote deve seguir para o próximo nó ou se deve ser entregue ao agente local. A sua necessidade prende-se com detalhes relativos à implementação. Numa próxima versão deste *software* deverá ser eliminada.

B.4 Programação do Módulo Adicional

Na apresentação do código produzido optamos por fazer a apresentação classe a classe, uma vez que a interacção entre estas j á foi devidamente abordada. Algumas das classes são definidas exclusivamente em OTcl e outras em C++, mas a maior parte delas tem um definição dual.

Classe `ConnectedNode`

A classe `ConnectedNode` é totalmente definida em OTcl e é derivada da classe base `Node`.

`ConnectedNode instproc get-default-classifier`	Devolve o classificador de endereços comum: `return $classifier_`.
`C.N. instproc get-lsrouting-agent`	Devolve o agente `LS/Source` alojado no nó, ou nada, se ele não existir (embora este último caso não seja possível em situações normais).
`C.N. instproc mk-default-classifier`	Cria todos os classificadores e inicia o pré-classificador. Inicia também a saída por omissão do classificador `switch` .
`C.N. instproc entry`	Devolve a interface de entrada do nó: `return $pre classifier .`
`C.N. instproc add-route`	Adiciona um destino no classificador de endereços.
`C.N. instproc start-connection`	Inicia uma ligação, chamando o procedimento `setup` do agente `Setuper`.
`C.N. inst-proc install-connection`	Instala uma ligação concluída no `switch.`

Tabela B.1: Procedimentos da classe `ConnectedNode`

Variáveis relevantes:

- `classifier_`, `pre_classifier` e `switch_`são os classificadores que aparecem na figura B.1;
- `address_` é o endereço do nó;
- `setuper_agent_` é o agente `Setuper` alojado no nó.

Os procedimentos da classe são apresentados na tabela B.1.

Classe `ConnectedClassifier`

Esta classe é derivada da classe `Classifier` (escrita em C++) que já foi descrita neste texto.

Contém, apenas, duas funções com alterações dignas de registo, descritas na tabela B.2. Quase todas as outras funções estão definidas na classe base.

`command()`	Admite três comandos, invocados do OTcl, que permitem configurar as saídas em classificadores deste tipo (var `pre_classifier` da figura B.1). São eles: • `set-connless-classifier`; • `set-switch-classifier`; • `set-setuper-agent`.
`classify()`	A função `classify()` devolve o *slot* de uma das suas três saídas, de acordo com a regra j á definida quando apresentámos os nós com ligação.

Tabela B.2: Funções da classe `ConnectedClassifier`

`admited call()`	Chama o método `CAC` que está definido neste agente (em OTcl) para saber se a ligação é, ou não, admitida.
`pack and send()`	Prepara um pacote de `SETUP` e preenche todos os seus dados, inclusive as listas de nós de ida e de retorno. Envia o pacote para o nó seguinte.
`generate confirm()` `generate reject()`	Geram pacotes de aviso de confirmação/rejeição da ligação pedida.
`recv()`	A função `recv()` é invocada quando um agente tem que receber um pacote. Naturalmente que os pacotes recebidos têm que ser do tipo `PT_SETUPER`.
`command()`	A função `command()` admite um único comando invocado a partir do OTcl: `setup`. Este comando, que j á foi explicado atrás, permite iniciar uma ligação. É invocado sempre pelo nó de origem.

Tabela B.3: Métodos definidos em C++ do Agente `Setuper`

Agente `Setuper`

O agente `Setuper` é simultaneamente um dos elementos chave e um dos elementos mais complexos da arquitetura que temos vindo a descrever.

Este agente tem métodos e variáveis definidas quer em C++, quer em OTcl, dosquais os mais importantes, serão apresentados em separado. Os procedimentos em C++ estão descritos na tabela B.3 e os procedimentos em OTcl estão descritos na tabela B.4.

Variáveis	• `my_node` :nó onde está alojado o agente; • `my_node_addr_`: endereço do nó onde está alojado o agente.
`Agent/Setuper instproc init`	Construtor. Cria o classificador de *hash* `forward_`, onde são armazenados dados de ligações em curso.
`Agent/Setuper instproc set-node- info`	Inicializa as informações relativas ao nó onde está alojado o Agente (`my_node_` e `my_node_addr`).
`Agent/Setuper instproc get-source-route`	Invoca o procedimento `get-compl-pat{}` do agente responsável pelo encaminhamento que existe no nó. Em resposta recebe o caminho completo em forma de *string* com os endereços dos nós.
`Agent/Setuper instproc CAC`	O procedimento `CAC{}`, como já foi dito, está incluído no agente `Setuper`. Devolve 0, caso não seja possível fazer a reserva pedida, 1, caso contrário.
`Agent/Setuper instproc connect-this- node`	Instala no classificador de *hash* temporário, `forward_`,os dados da ligação que está em curso.
`Agent/Setuper instproc discon-nect`	Permite terminar uma ligação.
`Agent/Setuper instproc over-threshold`	Este procedimento verifica se os valores de largura de banda duma ligação recém estabelecida ultrapassam um limite, acima do qual, os vizinhos devem ser imediatamente informados.
`Agent/Setuper instproc re-ject/confirm`	Instala em definitivo ou elimina os vestígios duma ligação, conforme se trate do procedimento `confirm` ou `reject`, respectivamente.

Tabela B.4: Métodos definidos em OTcl do Agente `Setuper.`

QoSLink e CAC

Como já foi dito, a classe `QoSLink` não tem, pelo menos para já, existência própria, sendo apenas um nome alternativo para a classe `SimpleLink` normal, com informação adicional sobre a largura de banda já reservada para um nó.

Quando se inicia um pedido de ligação, o CAC limita-se a ir à ligação física em questão verificar se há largura de banda disponível. Caso haja, essa largura de banda fica, imediatamente, reservada, sendo apenas desbloqueada se a ligação não for concluída com sucesso ou quando a ligação terminar.

Agente `rtProtoLsSource` (C++); `Agent/rtProto/LS/Source` (OTcl)

O agente `rtProtoLsSource` é constituído por duas partes, sendo uma delas es-crita em C++ (onde o agente é conhecido por este nome) e a outra em OTcl (onde o agente é designado por `Agent/rtProto/LS/Source`).

A classe definida em C++ é derivada da classe base `rtProtoLS`, já brevemente descrita, que permitia definir um agente capaz de realizar o protocolo LS. Esta nova classe introduz apenas algumas alterações, que passamos a descrever na tabela B.5. Antes de avançarmos para a explicação dos métodos escritos em OTcl é importante re-ferir um aspecto. Ao contrário do que acontece numa rede sem ligações vulgar, numa rede com ligações como a que queremos desenvolver, tem que ser trocada informação com o estado das ligações[2]. Isto porque poderão aparecer novas ligações ou desapa-recer algumas existentes que impliquem alterações na informação de largura de banda disponível. Isto representa um grau adicional de complexidade na decisão de quando e que informação enviar aos vizinhos.

Para já, a ideia é enviar informação sempre que uma reserva que ultrapassa certa percentagem da largura de banda de uma ligação física *(lin*k)é iniciada/terminada. Além desta ocasião, a informação continua a ser trocada periodicamente.

[2] Neste contexto “ligação” significa “conexão”.

`rtProtoLsSource-:: intfChanged()`	Esta função vai atualizar as estruturas internas do agente necessárias para fazer o encaminhamento. Já existia no agente base, mas agora há mais informação nas ligações físicas para recolher (largura de banda reservada e disponível). Esta função, que é chamada quando ocorre uma alteração ao estado das ligações físicas envia mensagens aos vizinhos com dados acerca dessas alterações (como aliás j á tínhamos visto).
`rtProtoLsSource-::-intfInfoChanged()`	Esta função é invocada quando há alterações no estado das ligações produzidas por uma nova reserva ou por uma reserva que seja eliminada. Neste caso, pode ser propagada informação para os vizinhos, mas a decisão de o fazer não depende da função (ver procedimento `over-threshol{}` do agente `Setuper`). É esta função que permite manter as estruturas internas sempre atualizadas apesar das ligações que vão surgindo/terminando.
`rtProtoLsSource-:: initialize()`	Diferente da função homônima da classe base, porque inclui informação completa do nó, contando com a largura de banda total e reservada.
`rtProtoLsSource-::command()`	A função `command()` adiciona à definição de métodos em OTcl os seguintes procedimentos: • `get-compl-path`: devolve uma lista completa de nós, desde uma origem A,até um destino B, para uma dada largura de banda; • `intf-info-changed`,`intfChanged` e `initialize`: permitem invocar as funções homónimas do C++.

Tabela B.5: Métodos definidos em C++ do Agente `rtProtoLsSource`

`Agent/rtProto/-LS/Source proc init-all`	Já explicada atrás, quando falámos de encaminhamento. Note-se que este procedimento é estático.
`A./rt./LS/S. instproc init`	Construtor do agente.
`A./rt./LS/S. instproc send--intf-info-update`	Quando possível envia informação atualizada dos estado das ligações, quando aparece ou termina uma nova ligação *(conexã*o). Eventualmente, o envio tem que ser adiado. Ver procedimento `send-periodic-update{}`
`A./rt./LS/S. instproc send--periodic-update`	Envia atualizações periódicas do estado das ligações físicas. Acontece que não podem ser enviadas duas actualizações separadas por menos do que um determinado limite de tempo, pelo que tem que agir em conjunto com o procedimento `send-intf-info-update{}`
`A./rt./LS/S. instproc get--link-status`	Obtém o estado de uma ligação física, incluindo a largura de banda disponível e reservada.

Tabela B.6: Métodos definidos em OTcl do Agente `Agent/rtProto/LS/Source`

Para evitar uma enchente de mensagens de atualização, não podem haver duas atualizações separadas por menos do que um dado intervalo limite de tempo.

Na tabela B.6 apresentamos os métodos do agente `Agent/rtProto/LS/Source` definido em OTcl.

Anexo C

Rede Neuronal de Ali e Kamoun

C.1 Método de Runge-Kutta de quarta ordem

Consideremos o sistema de equações diferenciais de primeira ordem e a condição inicial associada, representados em C.1.

$$\dot{x} = \mathbf{f}(x,t)$$
$$x(t_0) = \mathbf{a} \qquad \text{(C.1)}$$

x e a são vetores n-dimensionais e **f**é uma função de $\mathbb{R} \times \mathbb{R}^n \rightarrow \mathbb{R}^n$, possivelmente não linear. Estes vetores estão representados em C.2. Não vamos, neste texto, enumerar as condições necessárias para garantir a existência e unicidade da solução para este sistema de equações diferenciais.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}, \mathbf{f} = \begin{bmatrix} f_1(t; x_1; x_2; \ldots; x_3) \\ f_2(t; x_1; x_2; \ldots; x_3) \\ \vdots \\ f_n(t; x_1; x_2; \ldots; x_3) \end{bmatrix} \qquad \text{(C.2)}$$

O método de Runge-Kutta é iterativo e gera um conjunto de valores aproximados para x (t), x_1; x_2; x_3, . . ., para os instantes de tempo t_1; t_2; t_3,. . . O intervalo, *h*, entre os instantes de tempo considerados é fixo e é designado por *passo*. A ordem de grandeza deste passo depende do erro de aproximação pretendido. Quanto menor for o passo, menor será o erro da aproximação.

Então, dado x_n no instante t_n, x_{n+1} e t_{n+1} são calculados de acordo com a expressão C.3.

$$\begin{aligned} x_{n+1} &= x_n + k \\ t_{n+1} &= t_n + h \end{aligned} \qquad \text{(C.3)}$$

O vetor **k** é uma combinação linear de quatro vetores $\mathbf{k_1}$, $\mathbf{k_2}$, $\mathbf{k_3}$ e $\mathbf{k_4}$, calculados em C.4.

$$\begin{aligned} k_1 &= h \cdot \mathbf{f}(t_n; x_n) \\ k_2 &= h \cdot f\left(t_n + \frac{h}{2}, x_n + \frac{k1}{2}\right) \\ k_3 &= h \cdot f\left(t_n + \frac{h}{2}, x_n + \frac{k2}{2}\right) \\ k_4 &= h \cdot f\ (t_n + h, x_n + k3) \\ k &= \frac{1}{6}(k_1 + 2 \cdot k_2 + 2 \cdot k_3 + k_4) \end{aligned} \qquad \text{(C.4)}$$

C.2 Código para simulação em *software*

O método que acabámos de descrever permite simular por *software* o comportamento das redes de Hopfield em geral e, portanto, da arquitectura de Ali e Kamoun em particular. Na figura C.1 está ilustrado um programa, escrito em C, capaz de fazer essa simulação. Na figura C.2 é apresentado o ficheiro que contém a definição para uma rede de dados com cinco nós usada no artigo de (Ali & Kamoun, 1993). Neste ficheiro são igualmente definidas as constantes para as parcelas da função de energia, bem como o passo a usar no método de Runge-Kutta.

Depois de compilado este programa simulador, e admitindo que o nome do executável fosse AK, a forma correta de o utilizar seria:

```
AK entradaAK
```

Figura C.1: Código para simulação da RN de Ali e Kamoun

AliKamoun. c *Page 1*

```
/* Programa que simula uma rede de Hopfield, capaz de calcular
Caminhos Mais Curtos segundo o metodo de Ali e Kamoun */
#include <math. h>
#include <stdio. h>
#include <stdlib. h>
#include <unistd. h>
#include <fcntl. h>


#define POS( lin, col, tamanho) \
                      (( lin) * (tamanho) + (col))
#define POS2( X, i, Y, j, tamanho) \
                      (( X) * (tamanho) * (tamanho) * (tamanho) + \
                      (i) * (tamanho) * (tamanho) + \
                      (Y) * (tamanho) + \
                      (j))
#define DELTA( a, b)    (( a) == (b) ? 1 : 0)

#define LIMIAR 1e-5

typedef float t_ float;



void erro( char * frase) {
    fprintf( stderr, "% s\ n", frase);
    exit( 1);
}




t_ float calc_ actividade( t_ float * pesos, t_ float * vies,
                           t_ float * saida, int X, int i, int dim) {
     int Y, j;
     t_ float soma = 0;
    for (Y = 0; Y < dim; Y++) {
        for (j = 0; j < dim; j++) {
          if (Y == j)
              continue;
          soma += (*( pesos + POS2( X, i, Y, j, dim))) *
                  (*( saida + POS( Y, j, dim)));
      }
   }
   soma += *( vies + POS( X, i, dim));
   return soma;
}



t_ float f( t_ float t, t_ float Uxi, t_ float act) {
    return (Uxi + act);
}




t_ float transferencia( t_ float act) {
    const t_ float ganho = 1;

    return ( 1.0 / (1.0 + exp( -act * ganho)));
}
```

AliKamoun. c *Page 2*

```
void main( int argc, char ** argv) {
   t_ float * saida, * actividade, *p;
   t_ float Uxi, h, res;
   long iteracao;
   int continua = 1;

   t_ float * pesos, * vies, * custo;
   t_ float valor, C_ xi, mu_ 1, mu_ 2, mu_ 3, mu_ 4, mu_ 5;
   int * ro, ro_ xi;
   int dim, d, s, i, j, X, Y;

   if (argc == 2) {
     int d;

     if (( d = open( argv[ 1], O_ RDONLY)) < 0)
         erro(" Nao se consegue abrir o ficheiro\ n");
     dup2( d, 0);
     close( d);
}

   printf(" Introduza a dimensao do problema: ");
   scanf(" %d", &dim);
   if (dim <= 2)
     erro(" Dimensao invalida\ n");
   printf(" Introduza a origem: ");
   scanf(" %d", &s);
   if (s < 1 || s > dim)
     erro(" Origem incorrecta\ n");
   printf(" Introduza o destino: ");
   scanf(" %d", &d);
   if (d < 1 || d > dim)
    erro(" Destino incorrecto\ n");
   d--; s--;
   printf(" Introduza mu_ 1: ");
   scanf(" %f", &mu_ 1);
   printf(" Introduza mu_ 2: ");
   scanf(" %f", &mu_ 2);
   printf(" Introduza mu_ 3: ");
   scanf(" %f", &mu_ 3);
   printf(" Introduza mu_ 4: ");
   scanf(" %f", &mu_ 4);
   printf(" Introduza mu_ 5: ");
   scanf(" %f", &mu_ 5);

   pesos = (t_ float *) malloc( dim * dim * dim * dim * sizeof( t_ float));
   vies = (t_ float *) malloc( dim * dim * sizeof( t_ float));
   custo = (t_ float *) malloc( dim * dim * sizeof( t_ float));
   ro = (int *) malloc( dim * dim * sizeof( t_ float));
   if (! ro || !custo || !vies || !pesos)
    erro(" Falta memoria\ n");
   printf(" Introduza os custos (0 - custo nao existente): \n");
   for (i = 0; i < dim; i++) {
     for (j = 0; j < dim; j++) {
        if (j == i)
          continue;
        printf(" No %d para no %d: ", i + 1, j + 1);
        scanf(" %f", &valor);
        if (valor == 0)
           *( ro + POS( i, j, dim)) = 1;
        else
           *( ro + POS( i, j, dim)) = 0;
       *( custo + POS( i, j, dim)) = valor;
```

AliKamoun. c **Page 3**

```
    }
}
printf(" Determinacao dos viezes...");
for (X = 0; X < dim; X++) {
    for (i = 0; i < dim; i++) {
       if (X == d && i == s)
          *( vies + POS( X, i, dim)) = mu_ 5 / 2 - mu_ 4 / 2;
    else {
         C_ xi = *( custo + POS( X, i, dim));
         ro_ xi = *( ro + POS( X, i, dim));
         *( vies + POS( X, i, dim)) =
                  - mu_ 1 / 2 * C_ xi - mu_ 2 / 2 * ro_ xi - mu_ 4 / 2;
      }
    }
}
printf(" Feito!\ n");

printf(" Determinacao dos pesos...");
for (X = 0; X < dim; X++) {
   for (i = 0; i < dim; i++) {
      for (Y = 0; Y < dim; Y++) {
        for (j = 0; j < dim; j++) {
          *( pesos + POS2( X, i, Y, j, dim)) =
            mu_ 4 * DELTA( X, Y) * DELTA( i, j)
            - mu_ 3 * DELTA( X, Y)
            - mu_ 3 * DELTA( i, j)
            + mu_ 3 * DELTA( j, X)
            + mu_ 3 * DELTA( i, Y);
        }
      }
    }
}
printf(" Feito!\ n");

/* Metodo de Runge Kutta */
free( custo);
free( ro);
saida = (t_ float *) malloc( dim * dim * sizeof( t_ float));
actividade = (t_ float *) malloc( dim * dim * sizeof( t_ float));
if (! saida || !actividade)
   erro(" Falta memoria\ n");
for (X = 0; X < dim; X++) {
for (i = 0; i < dim; i++) {
res = *( actividade + POS( X, i, dim)) = 0.0;
*( saida + POS( X, i, dim)) = transferencia( res);
}
}
printf(" Introduza o passo (delta t): ");
scanf(" %f", &h);
printf("\ n\ n");

for (iteracao = 0; continua; iteracao++) {
   t_ float act, k1, k2, k3, k4;
   for (X = 0; X < dim; X++) {
     for (i = 0; i < dim; i++) {
          if (X == i)
            continue;
       act = calc_ actividade( pesos, vies, saida, X, i, dim);
       p = actividade + POS( X, i, dim);
       Uxi = *p;
       k1 = h * f( 0, Uxi, act);
       k2 = h * f( 0, Uxi + k1 / 2, act);
       k3 = h * f( 0, Uxi + k2 / 2, act);
       k4 = h * f( 0, Uxi + k3, act);
       *p += (k1 + 2 * k2 + 2 * k3 + k4) / 6;
```

AliKamoun. ***C*** *Page 4*

```
      }
    }
    continua = 0;
    for (X = 0; X < dim; X++) {
      for (i = 0; i < dim; i++) {
         if (X == i)
            continue;
         p = saida + POS( X, i, dim);
         valor = *p;
        *p = transferencia(*( actividade + POS( X, i, dim)));
        if (valor - *p > LIMIAR || valor - *p < - LIMIAR)
           continua = 1;
      }
   }
    /* Mostra o resultado todas as 500 iteracoes */
    if (iteracao % 500 == 0 || !continua) {
   system(" clear"); /* limpa o monitor */
    printf(" Iteracao = %d\ n", iteracao);
    for (X = 0; X < dim; X++) {
       for (i = 0; i < dim; i++) {
         if (X == i) {
            printf(" ");
            continue;
         }
         res = *( actividade + POS( X, i, dim));
          printf("% 15f", res);
     }
     printf("\ n\ n");
    }
    printf("\ n\ n");
    for (X = 0; X < dim; X++) {
      for (i = 0; i < dim; i++) {
        if (X == i) {
           printf(" ");
           continue;
        }
        res = *( saida + POS( X, i, dim));
        printf(" %. 1f ", res);
      }
      printf("\ n\ n");
    }
     printf("\ n");
   }
  }
}
```

Figura C.1 (continuação).

entradaAK **Page 1**

```
5
1
5

950
2500
1500
475
2500

0.377483
0.766675855
0
0

0
0.987736821
0.728571891
0.415129035

0
0
0.455640485
0.0369326319

0
0
0
0.876102889

0
0
0
0
1e-5
```

Figura C.2: Definição da rede de dados e das constantes da rede neuronal

Referencias Bibliográficas:

AGLET, Aglet.org a Resourse for the Aglet Java Communit.Disponível na Internet via WWW. URL http://www.aglets.org 2001.

ALI ,Mustafa k. Mehmet & KAMOUN,Faouzi. neural Networks for Shotest Path Computation and Routing Computer Networks. IEEE Transaction on neural networks, 4(5),941-953. 1993.

ARGON, E. & ARROWPOINT, R. & RAJAGOPALAN, b. A framework for Qos-Based Routing in Internet. Relatorio Tecnico. Nec, H. sandick, Bay Networks. 1998.

BARRETO, Jorge M., Introdução às redes neurais Artificiais. Anais V Escola Regional de Informática da SBC Regional Sul, 5 a 10 de maio de 1997. Páginas 41 - 71.

BELTRÃO, José; SAUVÉ, Jackes Philippe; GIOZZA,William F. et al. redes Locais de Computadores: Protocolos de Alto Nível e Avaliação de Desempenho. São Paulo: McGraw Hill, 1986.

BOSACK, L; HEDRICK, C. Bridges and Routers Observations, Comparisons and Choosing Problems in Large LANs. IEEE Network, New York, v.2, n.1, p.49-56, Jan. 1988.

BRISA. Gerênciamento de redes- Uma abordagem de Sistemas Abertos. Makron Books do Brasil, 1993.

CARVALHO T. Cristina. RMON permite gerênciar redes remotas. LANTIMES Brasil, vol. 3, edição 1, pp. 24-25, Abril 1997.

CASE J., Fedor M., Schoffstall M. e Davin J., A Simple Network Management Protocol (SNMP). RFC 1157. Network Working Group. Maio 1990

CISCO Systems Inc. Simple Network Management Protocol (SNMP). In Cisco Systems Inc., Cisco Connection Documentation - Enterprises Series, San Jose, 1996.

COCKBURN, D., McDonald, J. Burt, G. Brailsford, J. Beaton, J., and Lo, K., Expert Systems for On-line Fault Diagnosis in Electrical Power Networks", Proc. Int. Conf. on Electricity Distribution, Liege,Belgium, 1991.

COHEN, P. R. et al. An Open Agent Architecture. Working Notes of the AAAI Spring Symp.: Software Agents. AAAI Press. Cambridge, Mass., 1994, pp. 1-8.

COMER, Douglas E. Internetworking with TCP/IP: Principles, Protocols, and Architecture. 2nd ed. Englewood Cliffs: Prentice Hall, 1991. v.1.

CRIVELLI M. Piccoli, Trabalho de Graduação : Análise dos requisitos necessários para implementação de um protótipo de uma bancada para simulação de ambientes e agentes heterogêneos.Orientador: Nilson Montinho dos Santos. 1996.

CRONK, Robert; CALLAHAN, Paul H; BERNSTEIN, Lawrence. Rule Based Expert System for Network Management and Operations: An Introduction. IEEE Network, New York, v.2, n.5, p.7-21, 1988.

CYBENKO, G., Gray, R., Wu, Y., Khrabrov, A., Information Architeture and Agents, Technical Report, Tharey School of Engeneering, 1994.

DALE, J., A Mobile Agent Architecture to Support Distribuited Resource Information Management, Thesis of Doctor of Philosophy; Faculty of Engineering Department of Eletronics and Computer Science, University of Southampton, 1997.

DAVIDSSON, P. - On the Concept of Concept in the Context of Autonomous Agents. In Second World Conference on the Fundamentals of Artificial Intelligence, 1995.

DAYHOFF, Judithneural Network Architectures – An introduction. International Thomson Computer Press. . 1996.

DE CARLO, R. & Santos, N.M., Estudo e Aplicação de Agentes Inteligentes no Processo de Engenharia de Software, Trabalho de Graduação, Universidade Estadual de Maringá, Paraná, Brasil, 1995.

DENT, L., Boticario, J. Mc Dermott, J. Mitchell, T., A Personal Learning Apprentice. in Proceedings of the National conference on Artificial Intelligence. MIT Press.; Cambridge, Mass., 1992.

DOUGLAS W. Stevenson. Network Management - What is it and what it isn't. Disponível na Internet via WWW. URL : http://www.nettech.com/Netman.htm 1995.

ERLINGER, Michael A. RMON from Concept to Specification. In: International Symposium On Integrated Network Management. 3.. ISINM'93. Proceedings...[S.l.:s.n.], 1993.

ETZIONI, O.; Weld, D. S. A Softbot-Based Interface to the Internet. Comm. ACM. 37(7). pp. 72-76. July. 1994.

ETZIONI, O.; Weld, D. S. Intelligent Agents on the Internet: Fact, Fiction, and Forecast. IEEE Expert. August 1995.

FANG, K.; . LEIWARD, A.; Network Mangement - A Practical Perspective. Addison-Wesley Publishing Company, 1993

FEYNMAN, C. Nearest Neighbor and Maximum Likelihood Methods form Social Information Filtering.Internal Report, MIT, 1993.

FILHO, Edson Costa de Barros Carvalho., Modelagem, Aplicações e Implementações de redes neurais. Anais da IV Escola Regional de Informática da SBC Regional Sul, 21 a 27 de abril de 1996. Páginas 36 - 53.

FILIPE João Boavida de Mendonça Machado de Araújo, em, Aplicação das redes de Hopfield no caminhamento das redes de dados, dissertação de mestrado,Outubro de 1999, Universiade de Coimbra Portugal.

FREEMAN, James A./Skapura, David M., neural Networks - Algorithms, Aplications and Programming Techniques. Loral Space Information Systems and Adjunct Faculty, School of Natural and Applied Sciences University of Houston at Clear Lake.

GIESE, L. F., Estrutura de Agentes para os Processos de Compra e Venda utilizando Tomada de Decisão Difusa, Dissertação de Mestrado, Universidade Federal de Santa Catarina, 1998.

GILBERT, D., et. al.,Intelligent Agent Strategy, Technical Report, IBM Coorporation, Research Triangule Park, NC, USA, 1995.

GIORGI U., Ponticelli. Tutorial Proxy. Disponível na Internet via WWW. URL: http://penta.ufrgs.br/redes296/proxy/proxy.html 1996.

GORNI, Antônio Augusto, redes neurais Artificiais - Uma Abordagem revolucionária em Inteligência Artificial. Revista MicroSistemas edição 133 páginas 14 a 25 e edição 134 páginas 14 a 17, Ano XII

GUIFOYLE, C. and Warner, E., Intelligent Agents: The New Revolution in Software, Ovum, 1994.

GUPTA, Madan M.e RAO, Dandina H. - Neuro-Control Systems. Um volume selecionado reeditado. IEEE neural Networks Council, Sponsor.

HARRISON, C., Chess, D., Kershenbaum, A., Mobile Agents, Are they good idea?, Technical Report, IBM Research Division, T. J. Research Center, Yorktown, New York, 1995.

HAYES-ROTH, Frederick. Rule Based Systems. Communications of the ACM, New York, v.28, n.9, Sept. 1985.

HECHT-NIELSEN, Robert-Neurocomputing. HNC, Inc. and University of California, San Diego.

HEDBERG, S. R. Intelligent Agents: The First Harvest of Softbots Looks Promising. IEEE Expert. August 1995.

HEDRICK Charles L., Introduction to the Internet Protocols. Computer Science Facilities Group. Laboratory for Computer Science Research - Center for Computers and Information Services, The State University of New Jersey. 1988.

HOPFIELD, J.J. & TANK, D.W. neural computation of decisions in Optimization Problems. Biological Cybernetics. 1986.

HOPFIELD, J.J.neural Networks and Physical Systems with Emergente collective Computational abilites. Proccedings of the National Academy of Sciences od the U.S.A. 1982.

HUITEMA, Chistian. 1995. Routing in Internet. New Jersey: Prentice Hall PTR.
ISO International Standard Organization ISO/IEC. International Standard. ANSI/IEEE std. 802.3. Information Techology -Local and Metropolitan Area Networks-. Part 3: Carrier Sense Multiple Access Method and Phyxical Layer Specifications. 8802-3. [S.l.]. 1992.

JANDER, Mary. A Net Management System That Knows When to Worry. Data Communication, New York, v.21,n.4,5, p.41-42, Mar.1992.

JANDER, Mary. Proactive LAN Management: Tools that look for trouble to keep LANs out of danger. Data Communications, New York, v.22, n.4,5, p.49-56, Mar. 1993.

JENNINGS, N.R., and Witting, T., ARCHON: Theory and Practice, in Distributed Artificial Intlligence: Theory and Praxis (eds. N.M. Avouris and L.Gasser), Kluwer Academic Press 179 - 195.

JENNINGS, N.R., Cockburn, D., ARCHON: a Distributed Artificial Intelligence System for Industrial Applications.

JENNINGS, N.R., Corera, J. M., Laresgoiti, I., Developing Industrial Multi-Agent Systems.

JR., L.R. FORD & FULKERSON, D. R. Flows in Networks. Princeton , N.J.: Princeton University Press. 1962.

KLERER S. Mark. The OSI Managements Architecture: an Overview. IEEE Network, New York,v.2, n.2, p.20-29, Mar.1988.

KNOBLOCK, C. A.; Levy, A. Efficient Query Processing for Information Gathering Agents. CIKM Workshop on Intelligent Information Agents. 3 International Conference on Information and Knowledge Management. National Institute of Standards and Technology. Gaithersburg. Maryland. 1994.

KNOBLOCK, C.A.; Arens, Y. An Architecture for Information-Retrieval Agents. Working Notes of the AAAI Spring Symp.: Software Agents. AAAI Press. Cambridge, Mass., 1994, pp. 49-56.

KOVÁCS, Zsolt L. - redes neurais Artificias. Segunda edição, editora Collegium Cognitio, 1996

LEINWAND, Allan; FANG, Karen. Network Management: A Practical Perspective. Menlo Park. CA: Addison-Wesley, 1993.

LEVY, A. Y.; Yehoshua, S.; Srivastava, D. Towards efficient information gathering agents. Working Notes of the AAAI Spring Symp.: Software Agents. AAAI Press. Cambridge, Mass., 1994, pp. 64-70.

LIEBOWITZ, J. Expert System Applications to telecommunications. [S.l.]:Willey Interscience, 1988.

LINGNAU, A., Drobnik, O., An Infrastructure for Mobile Agents: Requirements and Architecture, Proceedings 13th DIS Workshop, Orlando, Flórida, 1995.

LOGG, Connie. The principal LAN performance problems. Stanford: SLAC -Stanford Linear Acelerator Center. 18 Oct. 1994.

LUCK, M., Griffiths, N., d'Inverno, M. - From Agent Theory to Agent Construction: A Case Study. Springer-Verlag, 1997.

MADRUGA, Ewerton L, Ferramentas de Apóio à Gerência de Falhas e Desempenho em Contexto Distribuído. Porto Alegre: CPGCC da UFRGS, 1994. Dissertação de Mestrado.

MAES, Pattie., Agents that Reduce Work and Information Overload. Communications of the ACM, vol.37 no.07 July, 1994.

MAGNO L. C. Silva, Trabalho de Graduação -: Utilização de Técnicas de IA para implementação de um Agente Operador de Sistema de acesso remoto. Orientador: Nilson Moutinho dos Santos. 1996.

McCLOGHRIE, K.; ROSE, Marshall. Management Information Base for Network Management of TCP/IP-based Internets: MIB-II. [S.l.]:DDN Network Information Center.

SRI International. 70p. Disponível por FTP Anônimo de “caracol.inf.ufrgs.br”. diretório “./pub/net/docs/rfc”. (Request for Comment 1213). . March 1991

MCCULLOCH, W.S., PITTS, W.H. a logical calculus of ideas immanent in nervous activity, Bull. Of Mathematicqal Biophysiscs, 1943

MILLER, Mark A. LAN Troubleshooting Handbook. 2nd ed. New York: M&T Books, 1993.

MILLER, Mark A. Troubleshooting Internetworks: Tools, Techniques and Protocols. San Mateo, California: M&T Books, 1991.

MINSKY, M.L., PAPERT, S. A. Perceptrons: an introduction to computacional geometry, MIT Press,1988

MOTOROLA Codex. The Basics Book of OSI and Network Manegement. Massachusetts: Addison-Wesley, 1993.

NASSER, Dan. Network Optimization and TroubleShooting: Achieve Maximum Network Performance. Indianapolis: NRP,1994.

NEMZOW, Martin A.W. The Ethernet Management Guide: Keeping the Link. 2nd ed. Local: McGraw Hill, 1992.

NETSCOUT Systems Inc. The Integration of RMON and SNMP Technologies for Managing Enterprise Networks. NetScout Documentation. 1996.

ODA C., S., Introdução à Gerência de redes.Disponível na Internet via WWW. URL: http://www.gt-er.cg.org.br/operacoes/gerência-redes 1994.

OTTEN, Hans. The Beholder Cookbook. Delft: Delft University, Netherlands, Dec.1993.

PARK, Dong-Chul & CHOI, Seung-Eok. A neural Network based Multidestination Routing Algorithm for Communication Network. IEEE, 1673-1678. 1998.

PERKINS, David. McGINNIS, Evan. Understanding SNMP MIBs. Chapter 5. Prentice Hall PTR. Upper Saddle River, New Jersey, 1997.

Process Software Corporation. Web Proxy Servers. In Purveyors Administrator's Guide. Disponível na Internet via WWW. URL: http://vms.process.com/~help/help.html 1997.

RAUCH, Herbert E. & WINARSKE, Theo. 1988. neural Networks for Routing Communication Traffic. Ieee Control System Magazine, Abril, 26-31.

REID, Peter. A Data Model for Network Monitoring and Management. Telecommunications, Norwood, MA, v.25, n.8, p.85-88, Aug. 1991.

RICH, Elaine; KNIGHT, Kevin. Inteligência Artificial. 2.ed. São Paulo: Makron Books, 1994.

ROCHOL, Jürgen. Projeto de redes. Universidade da Amazônia. Belém, 1998.

ROGÉRIO de Carlo, Trabalho de Graduação - ano 1995: Estudo e Aplicação de Agentes Inteligentes no processo de Engenharia de Software.Orientador: Nilson Moutinho dos Santos

ROSE, M.,e McCloghrie K., Management Information Base for Network Management of TCP/IP-based Internets. RFC 1156. Network Working Group. 1990.

ROSE, Marshall T. The Simple Book: An Introduction to Internet Management. 2nd ed. Englewood Cliffs: Prentice-Hall, 1993.

ROSE, Marshall. The Simple Book: An Introduction to Management of TCP-Ip based Internets. Englewood Cliffs, 1995.

SÁENZ A. Esmilda. Avaliação da Degradação dos Serviços para Realizar uma Gerência Pró-ativa em rede. Porto Alegre: CPGCC da UFRGS, 1994.(Trabalho Individual).

SOARES, L. F., Guido L., S., e Sérgio C.. redes de computadores: das LANs, MANs e WANs às redes ATM. Campus, 2º edição, 1997.

SPURGEON, Charles. Broadcasts Storm and Packets Avalanches on Campus Internets. University of Texas at Austin. July. 1989.

SPURGEON, Charles. Guide to 10-Mbps Ethernet Configuration. Networking Services. Austin:University of Texas at Austin, Jan. 1995.

STALLINGS, William. A practical guide to Queuing Analysis. BYTE, Peterborough, N.H., v.16, n.1, Feb. 1991.

SUN MICROSYSTEMS. SunOs 4.1.1 Rev. B Release and Instalation Manual. [S.l.:s.n.], Jan.1991.

TANENBAUM, Andrew S. Computer Networks. 2nd ed. Englewood Cliff: Prentice-Hall, 1991.

TANENBAUM, Andrews S. redes de Computadores. Rio de janeiro. Campus, 1997.

TAROUCO, L. M. Evolução do gerênciamento de redes. In Sociedade Brasileira para Interconexão de Sistemas Abertos, Ed., Gerênciamento de redes - Uma abordagem de sistemas abertos, pp. 1-12. Makron Books do Brasil, São Paulo, 1993.

TAROUCO, Liane M.R. Inteligência Artificial Aplicada ao Gerênciamento de redes de Computadores. São Paulo: USP-Escola Politécnica, 1990. Tese de Doutorado.

TAROUCO, Liane R.. www.penta.ufrgs.br

WALDBUSSER, S. Remote Network Monitoring Management Information Base. RFC 1271. Carnegie Mellon University. 1991.

WALDBUSSER.S. Remote Network Monitoring Management Information Base: RMON MIB. [S.l.]: Cornegie-Mellon University. 1995.

WALDBUSSER.S. Token Ring Extensions to the Remote Network Monitoring MIB. [S.l.]: Cornegie-Mellon University. 1993.

WHITE, J., Mobile Agents White Paper , URL: http://www.genmanic.com/agents/Whitepaper/whitepaper.html,1996.

WITTIG, T., ARCHON: An architecture for Multi-agent Systems, Ellis Horwood Chichester Ed.,1992.

WOOD, A. & Dr. BEALE R., Desktop Agents, School of Computer Science , The University of Birmingham, 1993

WOOD, A. , Towards a Medium for Agent-Based Interaction, School of Computer Science , The University of Birmingham, 1994

WOOLDRIDGE, M.; Jennings, N. R. Intelligent Agents: Theory and Practice. Knowledge Engineering Review. October 1994. Revised January 1995.

YOSHIDA, Keila M. - redes neurais e suas aplicações em Inteligência Artificial. Trabalho de graduação de 1996

ZURADA,J.M. Introduction to a Artificial neural System, West Publishing Co., USA, 1992