Aplicação das Redes de Hopfield no Encaminhamento em Redes de Dados

Filipe João Boavida de Mendonça Machado de Araújo

(Licenciado em Engenharia Electrotécnica)

Dissertação para obtenção do grau de Mestre em

Engenharia Informática

Outubro de 1999

Aplicação das Redes de Hopfield no Encaminhamento em Redes de Dados

Filipe João Boavida de Mendonça Machado de Araújo

Tese submetida para provas
de Mestrado em
Engenharia Informática

Departamento de Engenharia Informática

Faculdade de Ciências e Tecnologia

Universidade de Coimbra

Coimbra

Outubro de 1999

Tese realizada sob a orientação da

Professora Doutora Bernardete Martins Ribeiro

Professora Auxiliar do Departamento de Eng. Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra

Resumo

Esta tese estuda a possibilidade de se utilizarem Redes Neuronais, mais concretamente, Redes de Hopfield, no encaminhamento em redes de dados com Qualidade de Serviço. Para isso, é inicialmente abordado um conjunto de problemas que se colocam, quando o encaminhamento deve satisfazer requisitos de Qualidade de Serviço. Um destes problemas é a dificuldade em manter tabelas de encaminhamento previamente calculadas nos encaminhadores, como acontece nas redes de dados tradicionais.

Desta situação advém a necessidade de encontrar um método que seja mais rápido que o algoritmo de Dijkstra (que permite encontrar o óptimo) e que obtenha resultados tão próximos do resultado óptimo quanto possível.

As Redes Neuronais e, dentro destas, as Redes de Hopfield, são estudadas para o efeito. Em particular, nesta tese, tentamos determinar em que medida é que as Redes de Hopfield se configuram como uma alternativa ao algoritmo de caminho mais curto de Dijkstra. Para esse efeito são avaliadas, recorrendo a simulação, por um lado, a qualidade das soluções obtidas, face aos resultados óptimos de Dijkstra e, por outro, o tempo que as Redes de Hopfield necessitam para convergir para uma solução válida.

A tese propõe uma solução baseada em Redes de Hopfield que, tanto quanto sabemos, é original. Nesta solução, em vez de uma única camada de neurónios, como é tradicional, são usadas duas camadas. Esta solução vai ser comparada com outras duas Redes de Hopfield bem conhecidas.

A comparação é efectuada num simulador de redes de dados, tendo sido, para isso, necessário construir uma arquitectura com garantias de Qualidade de Serviço. Esta arquitectura também é apresentada nesta tese.

Abstract

This thesis studies the feasibility of using Neural Networks, specifically, Hopfield Networks, to routing in Quality of Service data networks. To do that, we must start to address a set of problems that arise whenever routing must satisfy Quality of Service requirements. One of such problems is the difficulty of keeping previously calculated routing tables on routers, the way they are kept now.

It follows from this situation that it is necessary to find a method faster than Dijkstra's (which can always find the optimum) and capable of getting as close to the optimum as possible.

Neural Networks and, among these, Hopfield Networks are considered for that purpose. In particular, in this thesis we try to determine if Hopfield Networks are an alternative to Dijkstra's shortest path algorithm. To evaluate this, a simulation is run and two different aspects are analyzed: firstly, the Hopfield Networks solutions quality against Dijkstra's; and secondly, the time needed for these neural solutions to converge to a valid result.

The thesis presents an Hopfield Network-based solution that is original, as far as we know. In this solution, instead of a traditional single neuron layer, two layers are used. This will be compared with two other well known Hopfield Network solutions.

The comparisons are made with the aid of a network simulator, for which an additional module was built to enable a Quality of Service architecture. This module, too, is presented here.

Palavras Chave

Redes Neuronais, Redes de Hopfield, Redes de Comunicação, Qualidade de Serviço (QoS), Encaminhamento, Protocolos de Encaminhamento, Algoritmos de Caminho Mais Curto.

Keywords

Neural Networks, Hopfield Networks, Communication Networks, Quality of Service (QoS), Routing, Routing Protocols, Shortest Path Algorithms.

Agradecimentos

Gostaria, em primeiro lugar, de agradecer à Professora Doutora Bernardete Martins Ribeiro, do Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra, na qualidade de orientadora científica, pela oportunidade concedida para a realização do presente trabalho, assim como pela motivação e disponibilidade que sempre demonstrou.

Queria também agradecer ao Professor Doutor Luís Eduardo Teixeira Rodrigues do Departamento de Informática da Faculdade de Ciências da Universidade de Lisboa (DI-FCUL), pelo papel de colaborador que desempenhou. Sem a sua orientação e sem o seu incentivo não teria sido possível concluir a tese nesta data.

Agradeço ao DI-FCUL por me ter proporcionado os meios e as condições para desenvolver esta tese.

Ao Centro de Informática e Sistemas da Universidade de Coimbra (CISUC) agradeço, também, os meios informáticos que me permitiu utilizar.

Quero ainda agradecer aos meus colegas do DI-FCUL pelas opiniões que sempre se prontificaram a dar. Pelo mesmo motivo, agradeço aos colegas e aos professores do Grupo de Comunicações e Serviços Telemáticos do CISUC.

Finalmente, gostaria de agradecer de forma especial à minha família, à Carla e aos meus amigos pelo apoio e paciência.

Aos meus pais e
à avó.

Índice

Lista de Figuras

Lista de Tabelas

Símbolos Importantes

A_i	padrão i, de dígitos binários
a_{pi}	i-ésimo dígito binário do padrão p
$b_l\,(t)$	quantidade de dados em trânsito na ligação l (em dígitos binários)
C	capacidade da rede (em dígitos binários)
C_{ur}	capacidade utilizada da rede (em dígitos binários)
C_{ij}	custo do nó i para o nó j
C_l	débito da ligação física l (em dígitos binários por segundo)
CT_{ij}	custo total do nó i para o nó j
d_l	atraso na ligação física l (em segundos)
d_{ij}	distância (ou custo) do nó i para o nó j
E	função de energia duma rede neuronal
$E(t)$	esperança do tempo de atraso extremo a extremo duma rede de dados (em segundos)
$f\,(\cdot)$	função de energia dum neurónio numa Rede Neuronal de Variáveis Dependentes
f_l	fluxo de dados na ligação física l (em dígitos binários por segundo)
I_i	viés aplicado ao neurónio i
L_{na}	número de ligações não aceites
L_t	número de ligações tentadas
M	número de arcos num grafo
N	número de nós num grafo

$O(\cdot)$	ordem de um algoritmo
$\mathbf{o}$	(o a negrito) vector das entradas aplicadas às variáveis dependentes numa Rede Neuronal de Variáveis Dependentes
o_i	entrada aplicada à variável dependente i
P	percentagem de perdas na rede de dados
P_{ij}	percurso do nó i para o nó j
R	percentagem de ligações rejeitadas
R_u	rácio médio de utilização das ligações físicas
T_i	tamanho do pacote i
v_i	nível interno de actividade do neurónio i
v_{Xi}	nível interno de actividade do neurónio na linha X coluna i
$\mathbf{v}$	(v a negrito) vector com os níveis internos de actividade duma rede neuronal
w_{ij}	peso do neurónio j para o neurónio i
$w_{Xi,Yj}$	peso da ligação do neurónio na linha Y coluna j para o neurónio na linha X coluna i
$\mathbf{x}$	vector x representado pela utilização de negrito
x_i	entrada correspondente à saída do neurónio i ($x_i = y_i$, se o atraso for ignorado)
x_{Xi}	entrada correspondente à saída do neurónio na linha X coluna i
y_i	saída do neurónio i
α_{ij}	coeficiente que exprime a relação entre a variável independente j e a variável dependente i numa Rede Neuronal de Variáveis Dependentes
δ	função *delta* de Kronecker
$\varphi_i(\cdot)$	função de activação do neurónio i
θ_i	viés ou limite aplicado ao neurónio i
ρ_{ij}	função que representa a existência ou ausência duma ligação física entre o nó i e o nó j
τ	$\tau = RC$ — produto da resistência pela capacidade dum neurónio aditivo

1

Introdução

Esta tese aborda a aplicação das Redes Neuronais (RN), mais especificamente as Redes de Hopfield, às Redes de Dados com capacidade de oferecer garantias de Qualidade de Serviço (QoS — "Quality of Service" [1]). Em particular, estudamos a exequibilidade de aplicar RN numa rede de dados, como meio de aumentar o seu desempenho.

De modo a motivar o trabalho, começamos no capítulo 2 por estudar os problemas que se levantam nas redes de dados que visam oferecer garantias de QoS. Neste estudo é dado ênfase ao problema do encaminhamento, uma vez que esta é uma das áreas — e aquela que nos interessa estudar — em que as RN podem ser aplicadas. Após uma síntese superficial do funcionamento de vários protocolos de encaminhamento utilizados actualmente, apresentamos com mais pormenor o protocolo "Open Shortest Path First" (OSPF), que se prefigura como o protocolo dominante para utilização em redes IP internas às organizações, i.e., redes que normalmente são geridas por uma única entidade e em que todos os encaminhadores (*routers*) utilizam o mesmo protocolo.

As redes IP em geral e o protocolo OSPF em particular servirão como referência para ilustrar quais as alterações e adições a efectuar a uma rede de dados, com vista a que seja possível definir e garantir alguns parâmetros de QoS. Note-se que este assunto é extenso e extravasa o âmbito desta tese, pelo que, aqui, não fazemos mais do que uma resenha às questões mais pertinentes. A panorâmica das redes de dados termina com o estudo dum protocolo baseado no OSPF — o QoSPF ("Quality of Service Extensions to OSPF"), que considera apenas a largura de banda como parâmetro de QoS.

[1] QoS, cuja definição será apresentada no capítulo 2 deste texto, é uma expressão utilizada para classificar as redes de dados capazes de oferecer garantias ao transporte do tráfego gerado pelas aplicações, em oposição às que não oferecem quaisquer garantias, como é o caso das redes IP ("Internet Protocol") tradicionais.

Desse capítulo, resultam duas conclusões fundamentais que enquadram a tese:

- só é viável oferecer garantias de QoS, ao longo dum dado percurso, se for estabelecida uma ligação / reserva para os fluxos de dados, antes destes se iniciarem;
- a determinação do percurso para um fluxo de dados não pode ser feita com recurso a uma tabela pré-calculada, porque o percurso escolhido não depende apenas do destino (que é uma variável discreta) e passa a depender também de, pelo menos, um parâmetro de QoS (que normalmente será uma variável contínua — largura de banda, por exemplo).

Isto significa que, sempre que alguma aplicação pretenda estabelecer uma ligação / reserva, é necessário calcular todo o percurso até ao(s) destino(s) [2], o que representa um esforço computacional consideravelmente maior do que pesquisar uma saída numa tabela previamente calculada, tal como é feito nos encaminhadores actuais. Esta tarefa pode constituir, portanto, um ponto de estrangulamento no desempenho do estabelecimento das ligações / reservas.

É importante compreender que a selecção dum percurso a utilizar por um fluxo de dados, quando são considerados parâmetros de QoS, pode não corresponder exactamente à determinação pura e simples do caminho mais curto (CMC), segundo um único critério. Com efeito, o percurso poderá ser seleccionado com base em múltiplos critérios e atendendo a algumas restrições específicas. Como veremos, no momento actual e para o tipo de problemas que iremos considerar, a determinação do percurso acaba por se obter através do cálculo de um CMC, onde é considerado apenas um único critério sujeito a algumas restrições que simplificam a rede de dados a considerar.

Deverá resultar deste capítulo um conjunto de conclusões que nos permitam construir para fins de simulação uma arquitectura com suporte básico de QoS.

Seguidamente, no capítulo 3, apresentamos uma arquitectura de RN geralmente utilizada em problemas de optimização: as Redes de Hopfield. Depois duma abordagem teórica, apresentamos um conjunto de soluções para o problema do CMC basea-

[2] Note-se que o destino pode ser mais do que um único nó.

das neste tipo de arquitectura. Estas soluções são apresentadas por ordem cronológica, o que se justifica por algumas das soluções serem evoluções de outras anteriores.

No que diz respeito às RN são importantes os seguintes aspectos:

- a arquitectura proposta para a RN deve ser passível de se concretizar em *hardware*, por um lado, e de se adaptar a alterações na rede de dados em que eventualmente tenha que operar, por outro;
- se uma RN utilizada para determinar um percurso for construída em *hardware*, assumimos que o cálculo é mais rápido do que o efectuado por qualquer algoritmo, devido à natureza maciçamente paralela das RN;
- interessa saber qual a qualidade das soluções obtidas, sobretudo para problemas de grande dimensão, que são aqueles em que estamos mais interessados (caso em que algoritmos como o de Dijkstra levariam relativamente mais tempo para calcular o CMC).

Do estudo deste capítulo deve resultar um conjunto de soluções baseadas em RN capazes de serem utilizadas para fazer encaminhamento. Nesta tese, estas RN não serão realizadas em *hardware*, mas apenas simuladas por *software*.

No capítulo 4 é apresentada uma RN que, tanto quanto sabemos, inclui alguns aspectos originais e que procura resolver alguns dos problemas exibidos pelas soluções clássicas apresentadas no capítulo anterior.

Dadas estas considerações, os objectivos desta tese são os seguintes: saber se é possível construir uma RN capaz de determinar percursos para os fluxos de dados, considerando parâmetros de QoS, por um lado e qual a qualidade das soluções obtidas, por outro. Caso os resultados sejam positivos, poderá justificar-se a utilização duma RN com vista a reduzir o tempo de estabelecimento duma ligação / reserva. Para podermos chegar a estas conclusões, vamos concretizar uma arquitectura com QoS num simulador de rede.

O capítulo 5 diz respeito à avaliação das soluções utilizadas. Esta avaliação será efectuada sob duas perspectivas diferentes. A primeira será pela comparação da quali-

dade dos percursos calculados pelas RN *versus* percursos calculados por um algoritmo, como o de Dijkstra, que determine o CMC. Outro factor de avaliação será relativo ao desempenho. Aqui, tentaremos verificar até que ponto é que as RN são, de facto, mais rápidas que a execução dum algoritmo sequencial.

No capítulo 6 apresentamos as conclusões e indicaremos algumas notas para trabalho futuro.

2

Encaminhamento em Redes de Dados

2.1 Encaminhamento em Redes IP Internas às Organizações

2.1.1 Problema do Encaminhamento

Consideremos uma rede de comunicação como a que está ilustrada na figura 2.1, composta por vários nós e respectivas ligações físicas. A cada ligação é associado um valor que pode representar grandezas várias, tais como, distância, atraso na propagação da informação, preço a pagar pela utilização da ligação, fiabilidade, etc.. Ao valor atribuído à ligação chamamos genericamente custo ou distância. Como entre

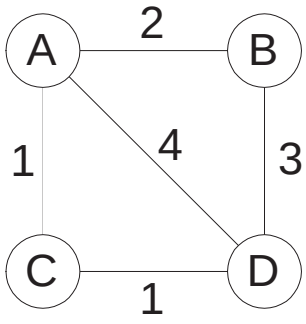


Figura 2.1: Rede de comunicação

dois nós distintos pode haver mais do que um único percurso possível, levanta-se a questão de saber que percursos são esses e a qual deles corresponde o menor custo total.

Em termos formais, definimos uma rede de comunicação como sendo um grafo

dirigido $G = (N, A)$, com N nós e M arcos. Associado a cada arco (i, j) está um número não negativo C_{ij} designado por custo do nó i para o nó j. Definimos também um percurso dirigido P_{od} como sendo uma sequência ordenada de nós, desde uma origem o até a um destino d:

$$P_{od} = (o, p, q, r, ..., v, d) \tag{2.1}$$

O custo total do percurso P_{od} será dado por $CT_{od} = C_{op} + C_{pq} + C_{qr} + ... + C_{vd}$. O problema de determinar o percurso P_{od} de custo mínimo é conhecido pelo problema do Caminho Mais Curto (CMC). É precisamente o problema de determinar o CMC que se põe, por exemplo, quando numa rede de comunicação de dados se pretende fazer o encaminhamento de pacotes, desde um determinado nó de origem até um nó de destino, minimizando a distância percorrida segundo uma dada métrica.

Os algoritmos mais vulgarmente utilizados para fazer o encaminhamento podem ser classificados como pertencendo a um de três grupos distintos: algoritmos centralizados, distribuídos ou localizados.

Num algoritmo centralizado, um determinado nó central mantém uma representação de toda a rede onde inclui as ligações que existem entre todos os nós e respectivos custos. Com base nesta representação, cabe ao nó central seleccionar o caminho a seguir para cada pacote que circula na rede. Este tipo de algoritmo sofre de dois problemas:

- sobrecarga provocada pelas comunicações com o nó central;
- falta de fiabilidade — uma falha no nó central deixa a rede inoperante.

Nos algoritmos de encaminhamento distribuídos e nos algoritmos de encaminhamento localizados a filosofia é diferente: as decisões relativas ao caminho a seguir por um pacote são tomadas ao nível dos nós por onde aquele vai passando. Para isso cada nó tem que manter um conjunto de informações que lhe permitam tomar decisões de encaminhamento. A natureza da informação mantida pelos nós distingue os algoritmos distribuídos dos algoritmos localizados.

Em termos breves, num algoritmo distribuído, cada nó conhece apenas o próximo passo a dar em direcção a um qualquer destino, desde que acessível. Por outro lado, num algoritmo localizado, cada nó mantém uma representação da topologia da rede e com base nesta representação determina o próximo passo a seguir por um pacote. Esta última ideia aplica-se apenas a redes de dimensão reduzida, precisamente por manter uma representação completa da rede — solução não escalável.

O algoritmo de Bellman-Ford (Bellman, 1957)(Jr. & Fulkerson, 1962) serve como exemplo de um algoritmo distribuído. O algoritmo de Dijkstra serve como exemplo de um algoritmo localizado. Nas secções seguintes descreveremos estes dois algoritmos e apresentaremos dois protocolos neles baseados, amplamente utilizados na Internet, para encaminhamento ao nível das redes internas às organizações.

2.1.2 Algoritmo de Bellman-Ford

Segundo (Huitema, 1995) o algoritmo de encaminhamento distribuído mais utilizado é o algoritmo de Bellman-Ford. A versão que apresentaremos deste algoritmo não é distribuída, mas centralizada. Veremos depois a forma de determinar o CMC distribuindo a informação e o cálculo pelos vários nós que compõem a rede.

Segue-se a descrição do algoritmo, de acordo com (Huitema, 1995):

- seja N o número de nós, M o número de ligações físicas;
- seja L a tabela das ligações de dimensão M, onde $L[l].m$ é a métrica da ligação, $L[l].o$ a origem da ligação e $L[l].d$ o destino;
- seja D a matriz de dimensão $N \times N$, onde $D[i,j]$ é a distância de i para j;
- seja H a matriz de dimensão $N \times N$, onde $H[i,j]$ é a ligação em que i encaminha pacotes destinados a j.

O algoritmo consiste em:

1. Inicializar todos os $D[i,j]$ para 0 se $i = j$, para ∞ caso contrário. Inicializar todos os $H[i,j]$ para -1;

2. Para todas as ligações l, para todos os destinos k: fazer $i = L[l].o$, $j = L[l].d$, calcular $d = L[l].m + D[j,k]$;

3. Se $d < D[i,k]$, actualizar $D[i,k] = d$; $H[i,k] = l$;

4. Se a matriz D sofreu alterações nesta iteração, regressar ao passo 2. Caso contrário, os cálculos estão terminados.

Este algoritmo é, na pior das hipóteses, de ordem $O(M \cdot N^2)$.

Na versão distribuída, a informação das matrizes H e D não está reunida num único ponto, mas fragmentada. Nesta versão, cada nó armazena uma das linhas das matrizes. O nó i guardará as linhas i das matrizes D e H, respectivamente, a distância que o separa de cada um dos outros nós e o próximo passo em direcção a cada um dos destinos possíveis.

O protocolo *Routing Information Protocol* — RIP — especifica a forma de construir em cada nó uma destas linhas (ou vector) com as distâncias e com o próximo passo. É este protocolo, que passamos a descrever.

2.1.3 *Routing Information Protocol* — RIP

O protocolo RIP pertence à família dos protocolos de vectores de distância. No RIP, cada nó possui uma tabela de encaminhamento que reúne informação relativa à conectividade com os outros nós: distância e ligação física a usar para despachar um pacote com destino a esses nós (tal como foi descrito atrás). A cada nó acessível corresponde uma linha da tabela de encaminhamento local.

No início as tabelas de encaminhamento contêm apenas uma entrada com informação de encaminhamento relativa ao próprio nó (custo 0, ligação física local), uma vez que este não tem conhecimento da topologia da rede. A tabela de um dado nó A está representada na tabela 2.1. Como é evidente, não é possível fazer encaminhamento com uma tabela no estado inicial, pelo que se impõe uma troca de mensagens na rede. Essa troca pode começar por qualquer nó, que difundirá a sua tabela através de todas as suas ligações físicas.

Desde A para	Ligação	Custo
A	local	0

Tabela 2.1: Tabela de encaminhamento inicial do nó A

Outro nó, digamos, o nó B, ao receber uma tabela de encaminhamento proveniente do nó A começa por adicionar a todas as distâncias presentes na tabela que recebeu o custo da ligação existente entre A e B, actualizando depois a sua própria tabela nas seguintes situações:

- se existir um novo caminho em direcção a um novo nó desconhecido até ao momento (um terceiro nó C, ou mesmo, o próprio A), insere-o na tabela local marcando como próximo passo o nó que enviou a tabela — A — com o custo actualizado respectivo.
- se existir um caminho mais curto do que aquele que era conhecido até agora em direcção a um destino, actualiza a entrada respectiva na tabela, segundo a mesma regra.

Se, da chegada duma tabela de encaminhamento, resultar alguma alteração na tabela local, então esta deve ser difundida através de todas as ligações físicas do nó (i.e., para toda a vizinhança). Quando outro nó recebe uma destas tabelas de encaminhamento age de acordo com a descrição antes efectuada.

Uma descrição completa do protocolo vai muito para além do objectivo deste texto mas, naturalmente, estão previstos mecanismos que o tornam adaptável a novas situações na rede resultantes, nomeadamente, de falhas, quer nos nós, quer nas ligações físicas.

A especificação do RIP pode ser encontrada em (Hedrick, 1988).

2.1.4 Algoritmo *Shortest Path First* de Dijkstra

Passamos, agora, à descrição do algoritmo de Dijkstra, de acordo com o que se encontra em (Huitema, 1995):

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 restantes nós, R. Consideremos também um conjunto de caminhos, O, ordenado por ordem crescente de custos.

O algoritmo consiste em:

1. Inicializar o conjunto A para conter apenas o nó 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 na 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 $O(M \cdot \log N)$, onde M é o número de ligações físicas existentes entre os nós e N o número de nós (se $M > N$ o que, em geral, é verdade). O CMC de Dijkstra será útil ao protocolo *Open Shortest Path First* — OSPF — que introduzimos de seguida.

2.1.5 Protocolo de Estado das Ligações — OSPF

Ao contrário do que sucede no RIP, onde cada nó mantém apenas a informação que lhe diz respeito, no protocolo OSPF, cada nó mantém informação completa acerca

da topologia da rede. Para representar esta topologia basta armazenar os custos fixados entre as diversas ligações físicas existentes. Numa rede muito simples, como a da figura 2.1, por exemplo, onde os números representados junto às ligações físicas representam os custos da ligação respectiva, cada nó terá de manter uma base de dados semelhante à que está representada na tabela 2.2. Na tabela 2.3 está a numeração usada para designar as ligações físicas. Este protocolo diz-se um protocolo de estado das

De	Para	Ligação	Distância
A	B	1	2
A	C	2	1
A	D	3	4
B	A	1	2
B	D	4	3
C	A	2	1
C	D	5	1

Tabela 2.2: Base de dados com a representação da rede

Ligação	Número
A — B	1
A — C	2
A — D	3
B — D	4
C — D	5

Tabela 2.3: Numeração das ligações da rede

ligações (*link state routing protocol*) precisamente por manter um mapa, que se encontra distribuído por todos os nós, com informação completa do estado da rede visível a partir de cada ponto.

A construção e manutenção adequada da base de dados consegue-se através dum protocolo que propaga a informação do estado das ligações através de toda a rede — "Disseminação por Inundação" (*Flooding Protocol*). Mais adiante, na secção 2.2, teremos oportunidade de estudar o OSPF com algum pormenor.

A partir da informação disponível, cada nó pode, por si só, calcular o CMC que o separa de qualquer outro, sendo, por isso, capaz de fazer o encaminhamento.

No entanto, o facto de o caminho a percorrer por um pacote ser apenas parcialmente determinado em cada nó, potencia o aparecimento de ciclos no percurso do pacote. Para garantir que isso não aconteça têm que ser asseguradas as seguintes condições:

- a métrica tem que ser a mesma para todos os nós, ou seja, todos os nós têm que ter a mesma vista dos custos, para as mesmas ligações físicas;
- na mesma situação, a computação do CMC efectuada por nós diferentes tem que resultar sempre no mesmo percurso.

A primeira condição deve ser assegurada pelo protocolo de inundação [1]. A segunda condição passa por utilizar um algoritmo que determine o óptimo. O algoritmo de Bellman-Ford ou o algoritmo SPF de Dijkstra apresentados neste texto satisfazem esse requisito, embora o segundo seja preferível por uma questão de rapidez. O nome *Open Shortest Path First* advém do facto de o protocolo OSPF não impor a utilização de um algoritmo em particular para o cálculo do CMC.

2.1.6 Comparação entre os protocolos RIP e o OSPF

Numa primeira análise, quando comparamos os protocolos de encaminhamento de que falámos — RIP e OSPF — a vantagem, em termos de escalabilidade, parece tender para o RIP, por duas razões essenciais: primeiro porque a quantidade de informação armazenada em cada nó é inferior; depois, porque a determinação do CMC no OSPF é excessivamente pesada em redes de grande dimensão. De facto, esta análise não é exacta por duas características que não transparecem da breve apresentação que fizemos do protocolo RIP:

- convergência muito lenta, no caso de haver alterações na configuração da rede;
- métrica pouco precisa.

[1] Esta condição pode ser difícil de assegurar em certos períodos transitórios, nomeadamente quando ocorre uma falha nalgum ponto da rede.

A primeira das desvantagens é, por si só, suficiente para justificar a substituição do RIP por outro protocolo. Numa rede de grande dimensão as alterações ocorrem com elevada probabilidade e provocam longos processos de reajuste das tabelas, durante os quais podem ocorrer ciclos no encaminhamento de pacotes, acarretando congestionamento e eventuais perdas.

A segunda desvantagem deve-se a alguns pormenores do funcionamento do RIP, que caem fora do âmbito desta tese e que podem ser encontradas em (Huitema, 1995). Pelo contrário, o OSPF permite utilizar uma métrica muito detalhada e permite, inclusivamente, usar múltiplas métricas, desde que não haja incoerências entre os nós quanto à métrica a utilizar — caso que poderia resultar no aparecimento de ciclos na circulação de pacotes.

Foram precisamente as limitações dos protocolos baseados em vectores de distância — RIP, neste caso — que motivaram o desenvolvimento dum protocolo baseado na informação do estado das ligações — OSPF —, capaz de ultrapassar essas limitações, mesmo que à custa duma maior complexidade.

2.2 *Open Shortest Path First* — OSPF

2.2.1 Elementos da Topologia Considerados no OSPF

Tipos de Redes Suportados pelo OSPF

O protocolo OSPF, que como vimos é executado nos encaminhadores, suporta três tipos de redes físicas, o que é o mesmo que dizer que os encaminhadores podem ter interfaces de ligação a três tipos de redes diferentes:

- redes ponto a ponto;
- redes com difusão;
- redes sem difusão;

No primeiro caso, em termos lógicos, existe uma ligação permanente e exclusiva entre dois encaminhadores. Concretamente, ao nível das camadas inferiores, o suporte deste tipo de ligação pode assumir múltiplas formas, mas as interfaces dos encaminhadores para este tipo de rede não precisam sequer de ter um endereço IP, caso em que a rede é designada por *unnumbered* (sem número).

Dos dois outros casos, a situação mais vulgar será provavelmente a de se encontrar um encaminhador ligado a uma (ou mais) redes que suportem difusão de datagramas IP. Entre estas redes contam-se as *ethernet*, por exemplo [2].

A grande diferença entre estes dois tipos de redes é que numa rede com difusão é possível aos encaminhadores descobrirem os seus vizinhos dinamicamente, através do protocolo "Hello", que iremos abordar mais adiante. Pelo contrário, em redes sem difusão a localização dos vizinhos tem que ser configurada manualmente [3].

Os encaminhadores constituem, então, um dos elementos da topologia da rede [4], onde são representados como vértices, sempre que a rede é representada por um grafo. Também representadas como vértices são as redes IP propriamente ditas.

Os vértices do grafo, são interligados por arestas, embora a existência destas arestas tenha que cumprir regras bem claras, que se prendem com o funcionamento da própria totalidade da rede (composta por redes IP e encaminhadores):

- dois vértices que representem encaminhadores podem ser ligados directamente por uma aresta, que representa neste caso, uma ligação ponto a ponto;
- um vértice que represente um encaminhador pode ser ligado por uma aresta com outro vértice, que represente uma rede. Neste caso, a aresta representa a interface

[2] É interessante constatar que o facto de ser ou não possível a um encaminhador enviar um datagrama endereçado a vários outros encaminhadores, não depende apenas do facto de a rede física que serve de base ao IP suportar, ou não, difusão. Enquanto que uma rede baseada num *bus*, como a *ethernet* permite obter esse efeito duma forma simples, outro tipo de redes, como a ATM, onde são utilizadas ligações virtuais entre nós, podem permitir o mesmo efeito, embora à custa duma organização hierárquica mais complexa. Tudo depende da configuração da arquitectura usada como suporte ao IP (ver (Laubach, 1994) por exemplo).

[3] Dois encaminhadores dizem-se vizinhos se tiverem interfaces numa rede comum. Por sua vez, uma rede que suporte vários encaminhadores (mais do que dois) designa-se por "rede multi-acesso".

[4] Note-se que há aqui algum abuso na utilização da palavra *rede*, que também é um dos elementos da própria topologia.

que o encaminhador possui para essa rede;

- dois vértices que representem duas redes não podem ser ligados entre si por uma aresta. Este facto é óbvio, se pensarmos que é necessário um encaminhador para interligar duas redes distintas.

Além destes dois tipos de vértices há ainda um terceiro que representa redes designadas por "stubs". A diferença entre as redes *stubs* e as redes normais é que aquelas são terminais, porque não fazem trânsito entre vértices, limitando-se a receber tráfego ou a gerá-lo. As restantes redes e os encaminhadores podem servir como meio de transmissão para tráfego que não geraram e de que não são o destino.

Subredes IP

O OSPF permite que haja uma divisão arbitrária das redes IP em múltiplas subredes até ao nível de detalhe máximo em que os computadores são representados individualmente. Naturalmente que, quanto maior for o detalhe, maior a quantidade de tráfego, de informação a armazenar e de processamento a afectar ao protocolo.

A forma de descrever as subredes IP é muito simples e é especificada em (Moy, 1994b): a cada endereço anunciado é adicionado uma máscara que especifica toda a gama de endereços IP que são, de facto, anunciados nesse endereço. Se o endereço for, por exemplo, `194.136.208.0` e a máscara for `0xffffff00` então está a ser representada toda a gama de endereços desde `194.136.208.0` até `194.136.208.255`. Facilmente se compreende então, que as redes representadas pelos vértices na topologia que já descrevemos não têm todas a mesma dimensão, em termos do número de computadores que incluem.

Exemplo

A figura 2.2 retirada de (Moy, 1994b) é um exemplo de um sistema autónomo [5] onde estão contemplados todos os tipos de redes e de interligação entre redes e encaminhadores, que já descrevemos.

O número anexado a cada ligação é o custo dessa mesma ligação, que normalmente será configurado manualmente. Se o administrador do SA decidir que uma dada ligação deve ser mais utilizada, atribui-lhe um número baixo, caso contrário, atribui-lhe um número mais alto. Um SA como o da figura 2.2 para efeitos computacionais é representado como um grafo ilustrado na figura 2.3, de acordo com as regras descritas atrás. Note-se que os custos são sempre direccionados dos encaminhadores para as redes, porque os custos das redes para os encaminhadores são nulos. Em termos de estrutura de dados, uma forma possível de armazenar um grafo é com uma tabela bidimensional, onde as colunas representam origens e as linhas destinos (ou vice-versa). Nas células de intersecção de origens com destinos são armazenados os custos da utilização atribuídos às interfaces em causa. As células que representam ligações inexistentes são marcadas como inválidas. Desta rede resulta a tabela 2.4, que também foi retirada de (Moy, 1994b). Para garantir a coerência da informação nos encaminhadores dum determinado SA, a configuração desse SA tem que ser correctamente propagada entre eles. Além disso, sempre que se verifica alguma alteração nessa configuração, por exemplo, ao nível das interfaces dum encaminhador (alteração no custo da interface ou uma interface que deixou de estar disponível devido a uma falha, por exemplo) a nova informação tem também que ser imediatamente propagada. A forma utilizada para o fazer será resumida mais adiante mas, para já, interessa saber que o resultado é que cada um dos encaminhadores existentes no SA tem a cada instante (excepto, porventura, em momentos transitórios) uma versão actualizada dum grafo como o que está ilustrado na figura 2.3.

[5] Sistema Autónomo: grupo de encaminhadores que trocam informação de encaminhamento através de um protocolo de encaminhamento comum, que no caso será o OSPF. Abreviado como *SA* (em inglês, Autonomous System —AS).

	RT1	RT2	RT3	RT4	RT5	RT6	RT7	RT8	RT9	RT10	RT11	RT12	N3	N6	N8	N9
RT1 RT2													0 0			
RT3 RT4					8	6							0 0			
RT5 RT6			8	8	7	6	6			5						
RT7 RT8					6									0 0		
RT9 RT10						7								0	0	0
RT11 RT12															0	0 0
N1 N2	3	3														
N3 N4	1	1	1 2	1												
N6 N7							1	1 4		1						
N8 N9									1	3	2 1	1				
N10 N11									3			2				
N12 N13					8 8		2									
N14 N15					8		9									
H1												10				

Tabela 2.4: Grafo resultante representado numa tabela (as colunas representam origens e as linhas destinos)

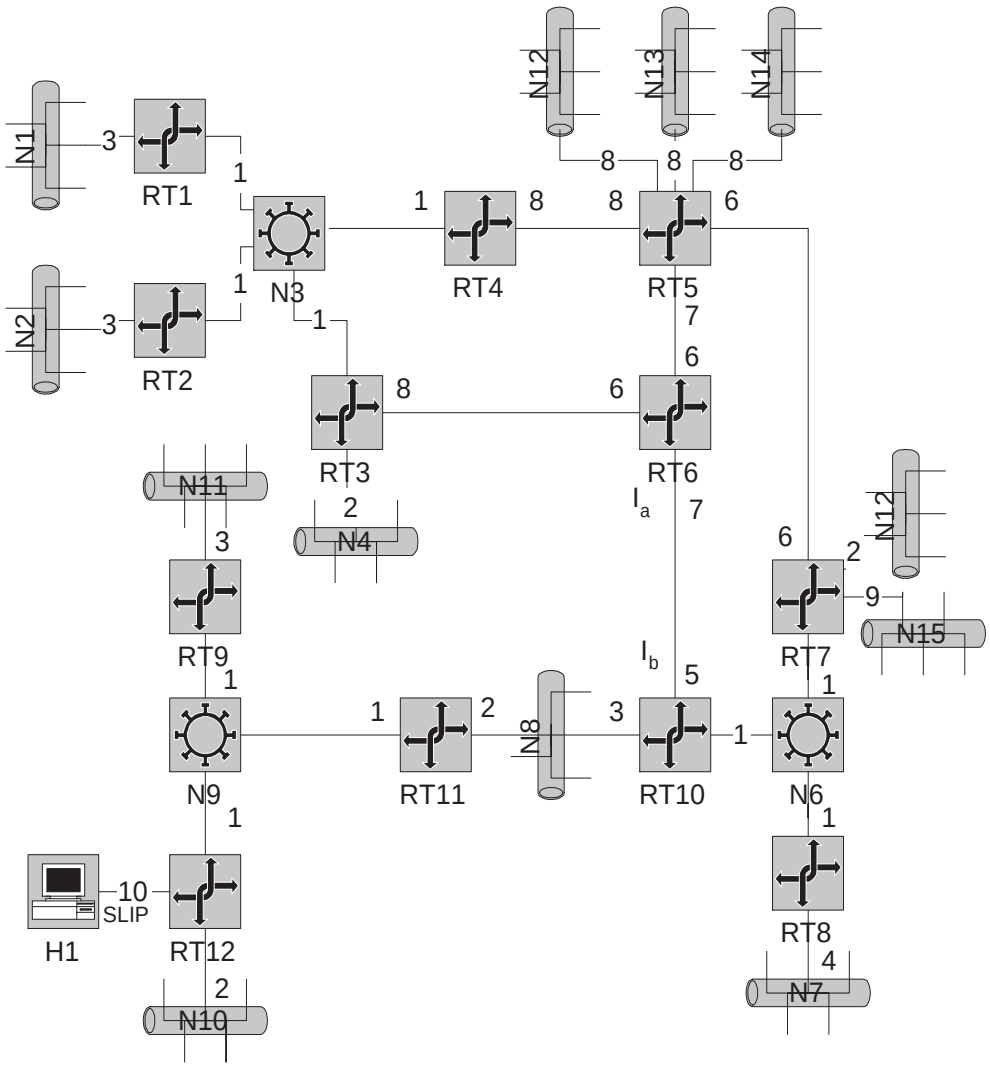


Figura 2.2: Exemplo de um Sistema Autónomo

Áreas

Para reduzir o tráfego gerado pelo OSPF, o espaço de armazenamento e o processamento, dentro dum SA, é possível dividir esse SA em grupos distintos de encaminhadores e redes. Esses grupos são conhecidos por áreas.

A ideia é que a informação relativa à topologia da rede (i.e., a informação que diz respeito à disposição dos vários componentes da rede) só é inundada dentro de cada área. Assim, deixa de ser verdade que todos os encaminhadores dentro dum SA tenham a mesma vista do SA, porque cada encaminhador passa apenas a ter conhecimento da topologia da área em que está incluído. Caso o encaminhador pertença a

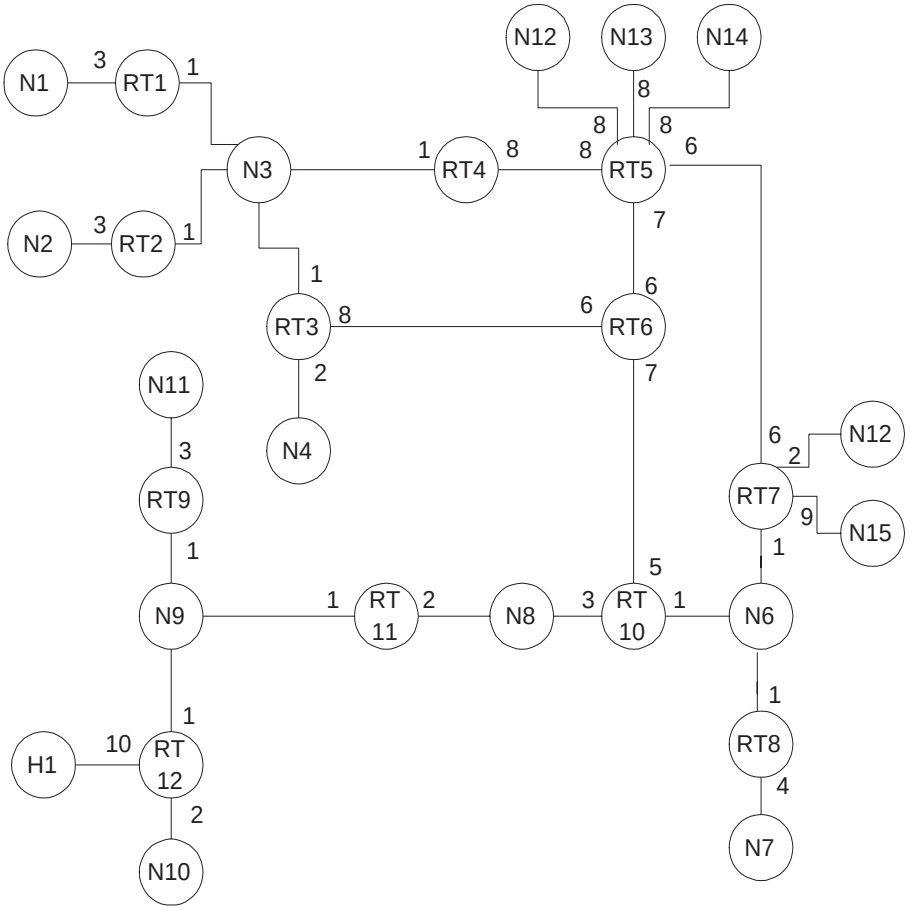


Figura 2.3: Grafo direccionado, que representa um SA

mais do que uma área terá de conhecer a topologia de todas as áreas a que pertence.

De entre as várias áreas dum SA existe uma área com características específicas que é conhecida por espinha dorsal ("backbone"). Esta área é composta pelas redes que não pertencem a nenhuma outra área e respectivos encaminhadores [6]. A espinha dorsal é ainda composta por todos os encaminhadores que pertençam a mais do que uma área. O conjunto dos elementos que formam a espinha dorsal tem que obedecer ele próprio à definição de área, mesmo que para o efeito seja necessário configurar manualmente ligações virtuais entre alguns dos encaminhadores. Isto significa que a espinha dorsal está ligada a todas as outras áreas do SA.

A espinha dorsal é responsável por fazer a distribuição da informação de encaminhamento e de fazer a distribuição do próprio tráfego entre as diversas áreas (tráfego inter-áreas). Esta característica confere ao SA uma disposição em estrela, sendo a espinha dorsal o ponto central responsável por despachar todas as comunicações.

A divisão adicional do SA em áreas, permite atribuir classificações para os encaminhadores que enumeramos de seguida:

- encaminhadores internos: todas as suas interfaces pertencem à mesma área. Correm uma única versão do algoritmo de encaminhamento pois conhecem apenas uma topologia que é a da sua área;

- *Area Border Router* — ABR: encaminhador que pertence a mais do que uma área (e que pertence à espinha dorsal, portanto). Tem que conter informação topológica de todas as áreas a que pertence, além da espinha dorsal, e tem que ser capaz de fazer o encaminhamento em todas elas. Envia informação (de encaminhamento e não de topologia [7]) para a espinha dorsal, que por sua vez a distribui para as outras áreas.

- encaminhadores da espinha dorsal: encaminhador que tem uma interface na espinha dorsal. Note-se que este pode ser um encaminhador interno ou um ABR,

[6] Note-se que os encaminhadores podem já pertencer a outras áreas.

[7] A diferença é que a informação de encaminhamento não permite determinar a topologia, porque inclui apenas para cada destino qual o próximo salto do encaminhamento e o custo respectivo.

conforme tenha todas as suas interfaces na espinha dorsal ou tenha alguma interface noutra rede, respectivamente.

A definição de *AS Boundary Router* — ASBR — não será incluída neste documento e pode ser encontrada, juntamente com uma explicação mais detalhada do funcionamento das áreas em (Moy, 1994b).

2.2.2 Configuração das Adjacências — Protocolo *Hello*

Para evitar que todos os encaminhadores vizinhos, i.e., encaminhadores que pertencem à mesma rede, troquem informação entre si, foi criado o papel de *Designated Router* (DR — *Encaminhador Seleccionado*) para uma rede. Este papel só faz sentido nas redes multi-acesso, onde é possível instalar mais do que um par de encaminhadores, em oposição às ligações ponto a ponto entre dois encaminhadores.

O DR é um encaminhador que pertence a uma dada rede (i.e., que tem uma interface nessa rede) e que é eleito e reconhecido pelos restantes encaminhadores vizinhos para o desempenho do cargo. Uma das funções do DR é possibilitar a definição do conceito de "adjacência". Uma adjacência é uma relação formada entre dois vizinhos e usada para efeitos de trocas de informação de encaminhamento. Este conceito faz sentido porque existem menos adjacências que vizinhos e, portanto, é reduzida a quantidade de informação que circula dentro de cada rede (além do tamanho das BD, como poderá ser percebido pela leitura de (Moy, 1994b)).

O papel do DR é o de ser adjacente a todos os encaminhadores vizinhos. Constrói-se assim, com raiz no DR, um grafo de adjacências que inclui todos os encaminhadores que têm interfaces numa rede. Este grafo assume a forma duma árvore de distribuição, que permite inundar a informação topológica, por todos os encaminhadores que têm uma interface na rede.

Convém não esquecer que os encaminhadores têm interfaces em mais do que uma rede e que, para cada rede, mantêm um conjunto de adjacências. Pode, por isso, acontecer, que dois encaminhadores tenham mais do que uma adjacência entre eles se tiverem mais do que uma rede em comum e se os grafos de adjacência assim o determinarem.

Além de criar adjacências, outro dos papeis do DR é gerar informação dos estados das interfaces em "nome da rede" — esta informação inclui o conjunto de todos os encaminhadores que pertencem à rede. A identificação da informação enviada — o *Link State ID* — é o da interface do DR. Em conjunto com a máscara (de que já tivemos oportunidade de falar), o *Link State ID* permite determinar a gama de endereços IP da rede. Porém, interessa referir que se o DR mudar, a informação das BD de todos os encaminhadores da área vai ter de ser alterada, o que é o mesmo que dizer que há uma alteração na vista da rede. Este facto implica recalcular caminhos ao nível de toda a área, o que deve ser evitado sempre que possível. De qualquer forma, a existência dum DR na rede é essencial, pelo que existe um DR de reserva ("Backup DR"), que também mantém adjacências com todos os outros encaminhadores da mesma rede e que a todo o momento pode assumir o papel de DR, embora com as desvantagens que acabámos de ver.

A eleição do DR é feita com o protocolo *Hello*. Este protocolo, que utiliza pacotes com o mesmo nome (pacotes *Hello* — ver adiante) é utilizado pelos encaminhadores para determinarem a localização dos seus vizinhos e para assegurarem comunicações bidireccionais com eles. Isto, no caso de a rede em questão permitir difusão; caso contrário, essa informação tem que ser configurada *a priori*. Terminada esta fase, os encaminhadores podem eleger o DR para essa rede.

O estabelecimento duma adjacência atravessa várias fases e só é dada por completa (diz-se então que os encaminhadores são *completamente adjacentes*) quando as BD dos encaminhadores ficarem totalmente sincronizadas. A sincronização é feita em duas fases:

- *Database Exchange Process*: nesta fase os encaminhadores trocam informação numa relação mestre / escravo sobre o estado das suas BD. Usam para isso *Database Description Packets* (ver adiante);
- envio de *Link State Requests*: terminada a fase anterior, cada encaminhador construiu uma lista com a informação que tem menos actualizada que o adjacente e começa a enviar *Link State Requests* (ver adiante). Quando todos os pedidos forem satisfeitos, as duas BD consideram-se sincronizadas.

Note-se que a adjacência pode ser usada para a inundação assim que começa a primeira destas duas fases e antes de ser considerada completa.

2.2.3 Inundação

No mecanismo de inundação são enviados pacotes *Link State Update*, que agrupam *Link State Advertisements* entre nós adjacentes. Estes *Link State Advertisements*, a que doravante chamaremos "anúncios" contêm, basicamente, informação relativa ao estado das interfaces dos encaminhadores e têm, portanto, que ser inundados por toda a área.

Em cada nó, o mecanismo de inundação tem início com a recepção dum pacote *Link State Update* durante a fase de trocas de BD (*Database Exchange Process*). Resumidamente, as ideias básicas do mecanismo de inundação são:

- só as adjacências é que fazem inundação;
- um encaminhador que não o DR envia anúncios só ao seu DR (tê-las-á recebido de outras redes onde tem interface);
- o DR propaga as informações para todas as suas adjacências na interface da rede em que é DR e, eventualmente, pelas outras interfaces, onde possivelmente não é DR;
- a informação só é propagada a uma adjacência se não houver a certeza de que esse vizinho já tem uma versão pelo menos tão recente quanto a que foi recebida.

Para enviar um ou mais anúncios, se assim o entender, o encaminhador utiliza pacotes *Link State Update*.

2.2.4 Pacotes do protocolo OSPF

A tabela 2.5 representa o tipo de pacotes existentes no OSPF e inclui um resumo das funções desempenhadas pelo tipo de pacote em causa. O formato dos pacotes utilizados pelo protocolo OSPF está totalmente especificado em (Moy, 1994b). No entanto,

Tipo	Nome	Descrição
1	Hello	Descobrir vizinhos e construir adjacências
2	DB description	Indica o estado das actualizações do emissor
3	Link State Request	Pede informação ao encaminhador adjacente
4	Link State Update	Contém informação sobre o estado das ligações físicas
5	Link State Acknowledgement	Confirma os anúncios (*advertisements*), que são enviados em *Link State Updates*

Tabela 2.5: Pacotes existentes no OSPF

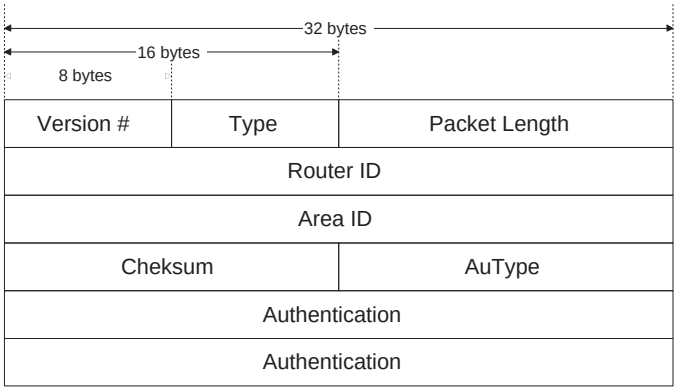


Figura 2.4: Cabeçalho dum pacote OSPF

pensamos que a inclusão do formato dos pacotes neste documento pode ser esclarecedora nalguns aspectos que tenham ficado excessivamente resumidos, pelo que se segue uma breve apresentação.

Cabeçalho OSPF

Todos os pacotes OSPF começam com um cabeçalho de 24 octetos (*bytes*), que está representado na figura 2.4. A maior parte dos campos são auto-explicativos. *Type* é um dos tipos da tabela 2.5, *Router ID* é a identificação do encaminhador que envia o pacote e *Area ID* é a identificação da área a que o pacote se circunscreve.

Dos vários tipos de pacote vamos ignorar o primeiro (*Hello*) e o último (*Link State Acknowledgment*) que pouco contribuem para um aprofundamento da compreensão do protocolo.

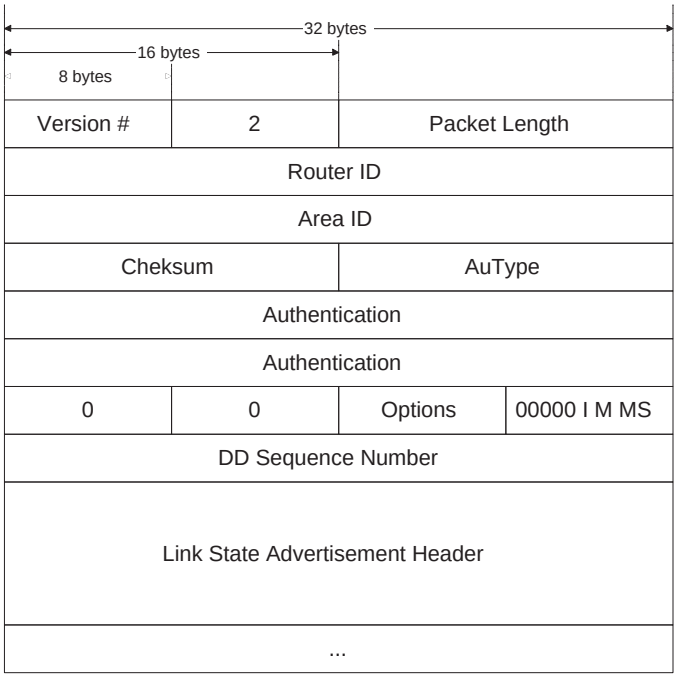


Figura 2.5: Pacote *Database Description Packet*

Database Description Packet

O formato destes pacotes (tipo 2) está representado na figura 2.5. Estes pacotes são trocados quando uma adjacência estiver a ser iniciada (antes de atingir o estado completamente adjacente). Chamamos a atenção para o seguintes campos:

- *DD sequence number*, que atribui um número de ordem aos pedidos. As respostas são relacionadas com os pedidos através deste número;
- os *Link State Advertisement Headers* descrevem os *Link State Advertisements* existentes na BD e a instância actual (i.e., a versão) desses anúncios.

Link State Request Packets

Estes pacotes (tipo 3) são enviados por um encaminhador para pedir *Link State Advertisements* sempre que, depois da fase *Database Exchange Process*, concluir que tem informação ultrapassada, ou em falta, na sua BD. O formato dos *Link State Request Packets* está representado na figura 2.6. Os pedidos de anúncios são identificados pelo

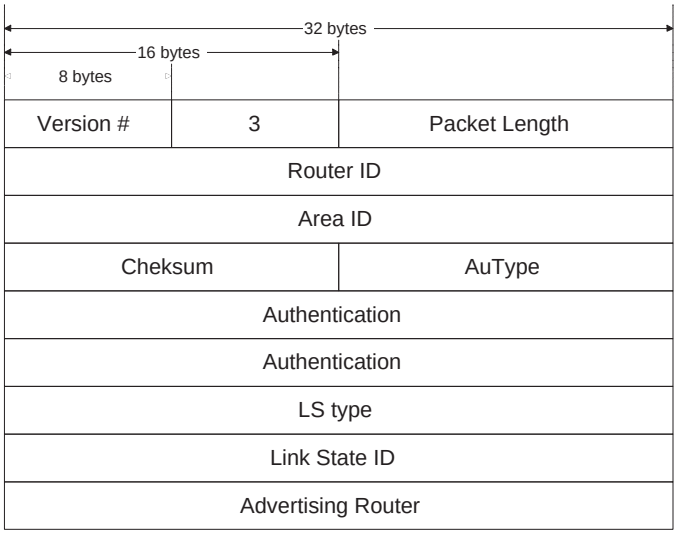


Figura 2.6: Pacote *Link State Request Packet*

Link State ID e *advertising router*. Note-se que não incluem um número de versão para o anúncio porque é sempre enviada a versão mais recente.

Link State Update Packets

Os pacotes de *Link State Update*, identificados pelo tipo 4, são utilizados no processo de inundação.

Cada um destes pacotes transporta um número variável de anúncios. Esse número é indicado no campo *#advertisements*, a que se seguem os anúncios propriamente ditos. Veja-se a propósito a figura 2.7

Formatos de *Link State Advertisements* (anúncios)

Há cinco tipos distintos de anúncios, que estão descritos na tabela 2.6. Os tipos de anúncios distinguem-se pelo campo *LSType* presente em todos os cabeçalhos. O formato dos cabeçalhos está representado na figura 2.8.

O anúncio em questão é sempre identificado pelos campos *Link State ID* e *advertising router* e a instância do anúncio através do campo *LS sequence number*.

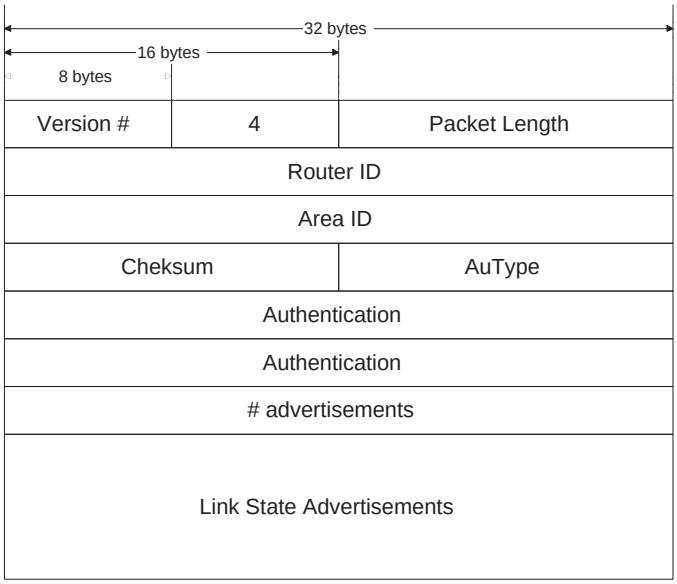


Figura 2.7: Pacote *Link State Update Packet*

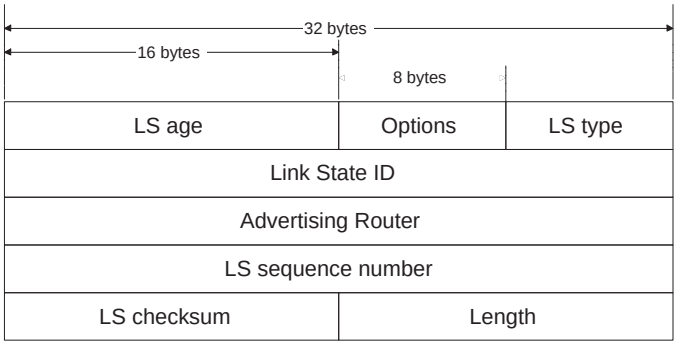


Figura 2.8: Cabeçalho dos *Link State Advertisements*

LSType	Descrição
1	Ligações físicas dum encaminhador
2	Ligações duma rede
3	Sumário de ligações (rede IP)
4	Sumário de ligações (ASBR)
5	Ligação externa ao SA

Tabela 2.6: Tipos de *Link State Advertisements*

2.2.5 Cálculo do Caminho Mais Curto

A partir da BD que representa a topologia da rede, obtida pelo processo de inundação, cada nó calcula todos os CMC, desde si próprio, até todos os outros nós da sua área. O algoritmo de Dijkstra, usado neste cálculo, já foi descrito atrás. Podemos ver o resultado da aplicação do algoritmo, em termos gráficos, como uma árvore cuja raiz é o próprio nó, sendo os outros nós dessa árvore os vértices do grafo que representa a topologia da rede original (redes e encaminhadores).

Calculada a árvore, são adicionados outros nós provenientes de duas fontes diferentes:

- de outras áreas; esta informação é fornecida em forma de sumários de configuração das áreas, que os respectivos ABR enviam para a espinha dorsal. Note-se que, com esta informação, não é possível determinar a topologia das outras áreas, mas apenas a distância que separa cada uma das redes existentes nessas áreas, dos encaminhadores da espinha dorsal, que são também ABRs da área local (note-se mais uma vez que o tráfego inter-áreas passa sempre pela espinha dorsal). Isto permite seleccionar correctamente o encaminhador da espinha dorsal para onde deve ser enviado o tráfego;

- de outros SA; um pouco à semelhança do que acontece no caso anterior, também aqui só é conhecida informação relativa às distâncias até às redes externas, de forma a calcular o ASBR mais económico de saída do SA.

Voltando ao exemplo da figura 2.2, a árvore calculada no encaminhador RT6 seria algo como o que está representado na figura 2.9. Na árvore estão representados todos os

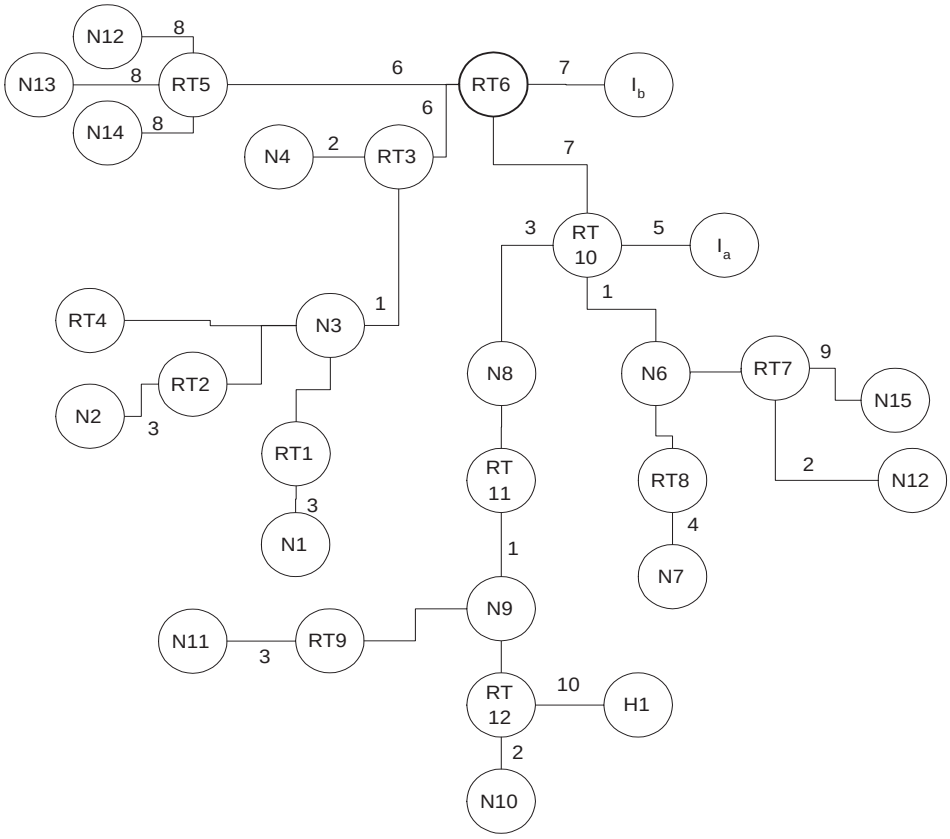


Figura 2.9: Árvore de encaminhamento do encaminhador RT6

caminhos e respectivos custos para todos os destinos. A partir desta árvore pode ser construída uma tabela que indique para cada destino existente, qual o próximo salto para fazer o encaminhamento e qual o custo dessa decisão.

Repare-se que, em termos computacionais, esta tabela representa uma significativa poupança de processamento, pois é construída uma única vez e recalculada apenas quando houver alterações nas informações relativas à topologia.

Utilização de Múltiplas Métricas

O OSPF permite a utilização de múltiplas métricas. Utiliza para esse efeito o campo *Tipo de Serviço* (TOS — *Type of Service*) existente em todo os anúncios. Isto significa que a cada custo que é inundado pela rede é adicionada uma etiqueta que especifica o TOS a que o custo se refere.

A utilização da informação de TOS propagada pelo OSPF é feita pelo IP, que inclui um campo homónimo nos cabeçalhos dos seus pacotes. As aplicações que utilizam o IP podem, assim, utilizar serviços com diferentes parâmetros de qualidade, especificando um valor diferente de 0 (serviço normal) para o TOS.

Os TOS possíveis estão representados na tabela 2.7. Para poder suportar um TOS específico, um encaminhador tem que manter informação topológica da área a que pertence e construir uma árvore como a que está representada na figura 2.9. Isto representa um esforço adicional considerável pois exige, basicamente, a duplicação de informação armazenada e de processamento por cada TOS.

É este mecanismo que acabámos de descrever que permite a utilização de várias métricas diferentes, que referimos atrás.

Codificação OSPF	Valor de TOS	Descrição
0	0000	Serviço normal
2	0001	Minimizar custo monetário
4	0010	Maximizar fiabilidade
6	0011	
8	0100	Maximizar *throughput*
10	0101	
12	0110	
14	0111	
16	1000	Minimizar atraso
18	1001	
20	1010	
22	1011	
24	1100	
26	1101	
28	1110	
30	1111	

Tabela 2.7: TOS suportados pelo OSPF

2.3 Enquadramento para Serviços com Garantias

2.3.1 Definição de Qualidade de Serviço — QoS

Qualidade de Serviço (QoS — "Quality of Service"), numa rede de comunicação de dados, é um conceito que exprime a capacidade que a rede tem de oferecer e garantir diversos tipos de contratos de utilização da sua infra-estrutura.

Normalmente a expressão "QoS" é utilizada para classificar redes que oferecem e garantem determinados serviços, como as redes de comutação de circuitos, por exemplo, em oposição ao que sucede noutro tipo de redes, como são normalmente as redes de comutação de pacotes, onde o serviço é designado por "melhor esforço" ("best-effort") ou ASAP ("As Soon As Possible", i.e., "tão cedo quanto possível"), termos estes que são usados como antónimos de "QoS".

A título de exemplo, e como referências, podemos apresentar dois casos diametralmente opostos no que toca às garantias de QoS oferecidas: a rede telefónica (que usa comutação de circuitos) e as redes IP (que usam comutação de pacotes).

Enquanto que nas redes telefónicas é reservado um canal com uma largura de banda fixa entre dois pontos extremos, na rede IP não existe qualquer reserva de largura de banda, sendo usada toda a que estiver disponível, que poderá eventualmente ser insuficiente para as necessidades duma qualquer aplicação num determinado momento.

No que diz respeito às redes telefónicas e em particular aos percursos adoptados por um canal telefónico, não existem tampões (*buffers*) nos nós de comutação, pelo que os dados não sofrem atrasos para além dos que são impostos pelo próprio meio físico de transmissão. Pelo contrário, nas redes IP, os pacotes podem ficar retidos por tempo indeterminado nos tampões, pelo que podem sofrer atrasos eventualmente elevados, que difiram, inclusivamente, de uns pacotes para os seguintes. Os pacotes podem mesmo ser descartados se num determinado encaminhador não houver mais espaço disponível nos tampões.

O preço a pagar pelas garantias de QoS está no sub-aproveitamento das infra-estruturas instaladas, que será potencialmente maior numa rede que não ofereça quaisquer garantias, com é o caso das redes IP. Enquanto que na rede telefónica cada chamada efectua a reserva dum circuito de utilização exclusiva, seja ou não transmitida informação (voz, geralmente), numa rede IP, toda a largura de banda é passível de ser utilizada até ao limite da congestão da rede.

Algures entre estes dois extremos situam-se as necessidades das aplicações distribuídas. Cabe, para isso, à rede de comunicação, definir os serviços e respectivas características postos à disposição das aplicações. Numa tentativa de sintetização, podemos separar o problema da realização duma rede capaz de oferecer garantias de QoS em três partes:

- definição dos serviços oferecidos, estabelecimento e definição dos parâmetros dos contratos a estabelecer com as aplicações;
- infra-estrutura de suporte aos serviços oferecidos;
- fiscalização e imposição dos contratos estabelecidos com as aplicações;

Naturalmente que estas três partes não são independentes entre si, mas a sua separação reduz consideravelmente a complexidade de construir uma rede com QoS.

Neste documento é dada especial ênfase à segunda destas três partes, i.e., à concepção duma infra-estrutura que permita garantir QoS.

2.3.2 Parâmetros de QoS

Quando as aplicações estabelecem contratos com a rede ou, dito de outra forma, quando estabelecem reservas de recursos, têm que especificar um conjunto de métricas cujo significado seja perfeitamente conhecido por todas as partes envolvidas na comunicação. A esse conjunto de métricas chamamos "parâmetros de QoS". Idealmente, o conjunto de parâmetros de QoS suportados por uma rede deve constituir um sistema ortogonal e completo, i.e., os parâmetros devem ser totalmente independentes entre si, mas quando combinados de forma arbitrária devem cobrir todo o espectro de especificações de serviços que façam sentido.

Na tabela 2.8 estão representados alguns dos parâmetros de QoS mais comuns. Estes parâmetros não obedecem necessariamente aos critérios que acabámos de referir. Cada aplicação distribuída ao iniciar um contrato com a rede (se este existir, evidente-

Parâmetro de QoS	Descrição
Largura de banda disponível / máxima	Capacidade disponível / máxima duma ligação física
Atraso médio / máximo	Tempo médio / máximo que leva a atravessar uma ligação física
Jiiter	Variação no atraso sentido pelas células ou pacotes dentro do mesmo fluxo de dados
Taxa de Perdas	Percentagem da informação (células, pacotes, dígitos binários, ...) que se perde na transmissão

Tabela 2.8: Parâmetros de QoS mais comuns

mente) especifica, ou não, um ou mais destes parâmetros. A tabela 2.9 mostra, para um conjunto de aplicações, quais os parâmetros que faria sentido considerar. Note-se que a escolha dos parâmetros de QoS que se decide considerar numa arquitectura é bastante importante, porque um algoritmo de encaminhamento não pode suportar requisitos

Aplicação	Parâmetros de QoS
Ftp	Largura de banda
Telnet	Atraso, *Jiiter*
Http	Largura de banda
Telefone	Largura de banda, Atraso, *Jiiter*, Taxa de perdas
Vídeo a pedido	Largura de banda, Atraso, *Jiiter*, Taxa de perdas

Tabela 2.9: Parâmetros de QoS a considerar pelas aplicações

de QoS que não possam ser representados convenientemente pelos parâmetros de QoS existentes. Veja-se (Argon *et al.*, 1998).

2.3.3 Avaliação dos Serviços Oferecidos

A partir do momento em que uma aplicação estabelece um contrato de prestação de serviços com uma rede é suposto o contrato ser cumprido por ambas as partes. A aplicação compromete-se a não gerar tráfego que viole esse contrato, enquanto que a rede tem que garantir que, nessas condições, o tráfego da aplicação é entregue dentro dos limites impostos para os parâmetros de QoS considerados.

Naturalmente que, para conseguir cumprir os contratos que estabelece, a rede tem que fazer uma gestão criteriosa dos seus recursos, tendo que, eventualmente, negar a sua utilização a certas aplicações. Por outro lado é preciso optimizar a utilização dos recursos disponíveis, por motivos de viabilidade económica. Cabe, então, ao conjunto de mecanismos que garantem a QoS determinar o ponto de operação da rede, tendo sempre em consideração estes dois objectivos antagónicos (cumprimento de contratos *vs.* optimização da utilização dos recursos).

A tarefa de avaliar estes mecanismos pode ser feita de acordo com determinados critérios, dos quais, enumeramos os seguintes:

- rácio de utilização das ligações físicas (ou taxa de ocupação): em cada instante, qual é a capacidade das ligações físicas que está a ser usada em relação à capacidade máxima instalada;
- débito da rede: número de dígitos binários (*bits*) por segundo que a rede consegue entregar às aplicações;

- atraso médio dos dados: quanto tempo, em média, leva um octeto dentro dum pacote a chegar ao destino;
- percentagem de rejeição de ligações / reservas: número total de ligações / reservas admitidas pela rede em relação ao número solicitado;
- percentagem de perdas: número médio de octetos perdidos (em pacotes descartados nos tampões, por exemplo) em relação ao número de octetos que entram na rede.

Note-se que, como é evidente, é possível definir muitos outros critérios, alguns dos quais consideravelmente mais complexos.

É de salientar, neste ponto, que o desempenho da rede segundo estes critérios está fortemente dependente do comportamento das aplicações, como aliás teremos ocasião de ver nos resultados experimentais desta tese.

2.3.4 Estrutura duma Rede com QoS

É importante constatar que, na sua totalidade, a infra-estrutura que permite garantir serviços com qualidade é bastante complexa. Entre outras funcionalidades tem que garantir as seguintes:

- armazenamento e propagação da informação relativa às reservas efectuadas pelas aplicações e ao estado das ligações físicas;
- estabelecimento de novas reservas;
- determinação dos caminhos a seguir pelos pacotes, de acordo com os vários parâmetros de QoS;
- gestão das filas de espera à entrada das ligações físicas;
- controlo de admissão de ligações / reservas (“Connection Admission Control — CAC).

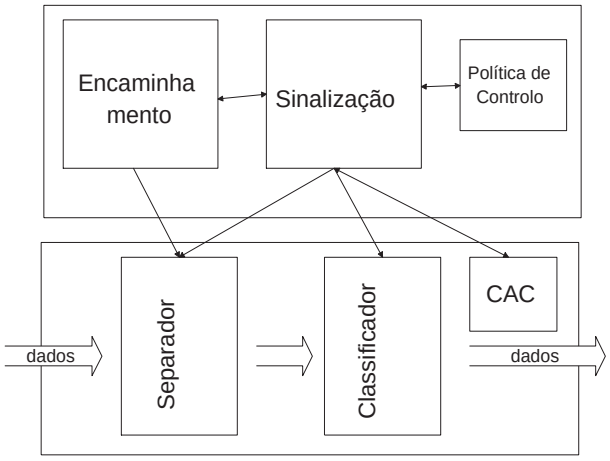


Figura 2.10: Divisão em módulos dum encaminhador com QoS

Por este motivo convém que a infra-estrutura não seja monolítica, mas sim modular, de forma a também aqui se reduzir a complexidade da tarefa à custa da divisão de um todo em partes bem delimitadas, com interacções claras.

Na figura 2.10 apresentamos uma divisão possível em módulos para esta infra-estrutura, semelhante à que é apresentada em (Braden *et al.*, 1997). A utilidade de cada um destes módulos será estudada mais adiante. Interessa aqui compreender, que os princípios subjacentes a esta divisão fazem sentido qualquer que seja o tipo de rede que consideremos. Neste caso, estamos a falar de redes IP, mas facilmente podemos aplicar o mesmo modelo às redes "Asynchronous Transfer Mode" — ATM, por exemplo.

O problema de definir estes módulos é mais complexo nas redes IP, que foram pensadas na sua génese, apenas para serviços melhor esforço. Isto porque, em primeiro lugar, não é aceitável retirar ou alterar partes que já existam; em segundo lugar quaisquer novos elementos a introduzir terão que coexistir com toda a infra-estrutura actual durante um período de tempo que poderá vir a ser longo.

A solução radical e diametralmente oposta de pura e simplesmente substituir o IP por outra tecnologia orientada de raiz para oferecer garantias de QoS não parece ser razoável, dada a forte implantação do IP [8].

[8]Têm sido dedicados esforços muito grandes ao desenvolvimento do ATM, que se propôs a partir de certa altura ser uma tecnologia de suporte universal. Porém, no momento em que é escrito este documento, o sucesso do ATM é algo incerto, sobretudo no que diz respeito às redes locais (LANs).

Nas secções seguintes, vamos abordar o funcionamento de cada um dos módulos da divisão proposta.

2.3.5 Sinalização

O papel do módulo de sinalização pode variar um pouco com o tipo de solução adoptada.

Tentando encontrar uma definição comum, podemos dizer que este módulo é responsável por codificar e transmitir toda a informação relativa ao estabelecimento de reservas.

Se a rede for orientada às ligações, como o ATM, por exemplo, o papel dum módulo de sinalização será o de estabelecer novas ligações sempre que tal for solicitado. Estabelecer uma nova ligação implicará escolher um caminho desde o ponto de origem até ao ponto de destino e encaminhar um pedido, para o efeito, entre esses dois pontos [9]. É importante referir que a selecção do caminho é uma tarefa que cabe ao módulo de encaminhamento e não ao módulo de sinalização. Ao longo do caminho seleccionado tem que ser armazenada a informação relativa à nova ligação já que, no ATM, as células seguem sempre o mesmo percurso, depois daquela ser estabelecida.

Nas redes IP, tradicionalmente, por não haver ligações, não está previsto qualquer módulo de sinalização. No entanto, para ser possível garantir QoS, tem que existir algum mecanismo de sinalização, razão pela qual têm sido desenvolvidos alguns esforços no sentido de criar protocolos capazes de desempenhar esse papel.

Um desses protocolos é o Protocolo de Reserva de Recursos (RSVP — *Resource ReSerVation Protocol*). Este protocolo é orientado de raiz para uma rede IP *multicast* e opera directamente sobre o protocolo IP. No RSVP há uma diferença explícita entre emissores e receptores, cabendo a estes últimos efectuar as reservas. Esta situação, que se apresenta contra-intuitiva numa primeira análise, apresenta duas vantagens consideráveis:

[9] O caso das ligações ponto a multiponto é um pouco mais complexo.

- poupança de recursos, uma vez que cada receptor só reserva as quantidades de que precisa, podendo haver vários receptores na mesma sessão (i.e., com o mesmo endereço *multicast*, protocolo de transporte e porto de destino) a receber os dados com qualidades distintas;

- escalabilidade, porque as reservas não são propagadas para montante (i.e., na direcção do emissor) mais do que aquilo que é estritamente necessário. Os encaminhadores mais próximos do emissor poderão nem sequer tomar conhecimento de que há reservas a ser efectuadas ou canceladas.

Aos emissores cabe fazer o aviso do tráfego que vão gerar. A propagação destes avisos é feita na direcção dos receptores e os percursos seleccionados têm que ter em consideração a QoS anunciada. Tal como seria de esperar num protocolo de sinalização, também no RSVP, é este um dos pontos de interacção entre a sinalização e o encaminhamento. Os avisos de tráfego são feitos em mensagens designadas por *PATH*. A resposta dos receptores a estas mensagens é feita com mensagens RESV, que são enviadas na direcção dos emissores. À medida que as mensagens RESV com as reservas efectuadas pelos receptores sobem para montante na árvore de distribuição *multicast*, podem ser fundidas em novas mensagens com requisitos iguais ou superiores aos das mensagens que lhes deram origem. Esta constitui uma das ideias base na lógica do RSVP.

Note-se que, neste protocolo, a decisão de encaminhamento com QoS e a reserva propriamente dita ocorrem em momentos distintos, pelo que há a possibilidade de ser seleccionado um percurso subóptimo para as reservas. Este problema é um dos preços a pagar pelo tipo de solução adoptada, mas pode ser minorado, obrigando os emissores a renovarem periodicamente o envio de mensagens PATH, que resultem num eventual reajuste dos percursos [10].

A especificação do RSVP pode ser encontrada em (Braden *et al.*, 1997).

Já vimos que uma das interacções do módulo de sinalização se faz com o módulo de encaminhamento. Inclusivamente, o módulo de sinalização não precisa sequer de

[10]Embora isso possa não acontecer se o protocolo de encaminhamento fizer *route pinning*, que é um conceito que será abordado mais adiante.

ter conhecimento dos parâmetros de QoS, podendo tratá-los como objectos opacos compreendidos apenas pelas aplicações (ou por entidades que oferecem os serviços às aplicações, pertencentes ao sistema operativo, por exemplo) e pelo módulo de encaminhamento. Neste caso, o módulo de sinalização limita-se a transferir esses objectos duns locais para outros, mas sempre sem inspeccionar directamente o seu conteúdo. É assim que se passa no RSVP. Eventualmente, se o RSVP precisar de manipular informação de QoS (*flowspecs*) para fundir duas reservas diferentes, por exemplo, terá de recorrer a uma funcionalidade oferecida por alguma entidade do sistema.

Outra das interacções do módulo de sinalização será feita com o módulo de CAC, que descreveremos adiante. No ATM, por exemplo, quando uma ligação está em fase de estabelecimento, em todos os nós que ela atravessa, o CAC tem que ser consultado, para determinar se é ou não possível estabelecer a ligação. Se não for, a solução a adoptar pode passar pela pura e simples desistência ou pela tentativa de um caminho alternativo, dependendo da complexidade do mecanismo.

As interacções deste módulo com os restantes estão representados na figura 2.10, por intermédio de setas.

2.3.6 Algoritmo de Encaminhamento

Vimos já que existem alguns pontos de contacto entre o módulo de sinalização e o módulo de encaminhamento. Em particular, vimos que é ao módulo de encaminhamento que compete determinar um caminho, ao longo do qual será estabelecida uma reserva, tendo em conta os parâmetros de QoS solicitados. A determinação do caminho é feita normalmente, uma única vez ao longo da vida da reserva ou, quando muito, periodicamente [11]. Uma excepção a esta regra ocorre quando surge uma falha na rede que interrompe o percurso da reserva. Neste caso, se a sinalização for suficientemente evoluída é possível, e desejável, tentar seleccionar um caminho alternativo.

Note-se que, aparentemente, o módulo de sinalização deveria tomar a iniciativa

[11] Embora, quando se esteja a usar o RSVP, por exemplo, a determinação do caminho e a reserva sejam decisões tomadas em momentos diferentes. Neste caso o caminho já se encontra determinado no momento da reserva.

de alterar um determinado percurso quando surgisse outro que apresentasse melhores características de QoS. Esta capacidade é relativamente simples de conseguir num protocolo de sinalização como o RSVP, em que o estado das reservas é volátil e estas têm que ser refrescadas continuamente, mas sensivelmente mais complicado num protocolo que utilize ligações estáticas, como acontece numa rede ATM.

No entanto, alterar os caminhos para reservas já estabelecidas pode introduzir um novo problema de instabilidade no encaminhamento. Com efeito, numa rede onde haja uma taxa elevada de reservas e de libertação de reservas, os resultados poderão ser alterações constantes às decisões de encaminhamento.

Além disso, há que garantir que o protocolo de encaminhamento, quando toma decisões acerca duma sessão [12], considera sempre as reservas estabelecidas para essa mesma sessão numa próxima reavaliação do percurso seleccionado. Caso isso não seja feito, pode acontecer o seguinte cenário: suponhamos que na primeira vez, o algoritmo seleccionava o percurso *A*; quando voltasse a reavaliar a sua decisão, se não descontasse a reserva feita pela sessão nesse percurso, poderia preferir desta vez o percurso *B*. Na terceira vez voltaria a seleccionar o percurso *A* e assim sucessivamente, alternando indefinidamente entre os dois percursos. Esta situação está exemplificada em (Goto *et al.*, 1997).

Posto isto, concluímos ser essencial que o encaminhamento com QoS tome decisões estáveis, e que não dependam das reservas já estabelecidas pelas sessões para as quais se pretende determinar um novo percurso.

Note-se que este é um tipo de problema que podemos considerar como pertencendo ao módulo de encaminhamento. No entanto, pode ser tratado em conjunto com o módulo de sinalização, que pode dar algumas indicações ao módulo de encaminhamento. Essas indicações podem consistir em fixar partes do percurso seleccionado para uma determinada reserva (enquanto tal for possível), por exemplo.

Um algoritmo de encaminhamento com QoS, que apresentaremos mais adiante, o

[12] Uma sessão nas redes IP, como já vimos, define-se como um trio da forma {protocolo de transporte, endereço IP de destino, porto de destino}. Neste contexto, podemos alargar o significado da palavra sessão, de forma a incluir o conceito de ligação noutras tecnologias diferentes, como o ATM.

QoSPF — *Quality of Service Path First* (cujo nome resulta da concatenação de "QoS" com "OSPF") — utiliza uma técnica designada por *route pinning*, que basicamente só altera a decisão acerca dum percurso se este deixar de ser utilizável.

Actualização do Estado das Ligações

Algo que não será difícil de deduzir, a partir não só do que já dissemos, mas também do funcionamento dos protocolos de encaminhamento actuais — como o OSPF, por exemplo — é que a informação sobre o estado das ligações físicas é uma tarefa que continua a ser da competência do protocolo de encaminhamento.

No entanto, têm que ser propagados dois tipos distintos de informação:

- sobre o estado das ligações, que já existia nos protocolos tradicionais, como o OSPF;
- sobre o estado das reservas, que propaga por todos os encaminhadores os dados relativos às reservas efectuadas pelas aplicações.

É importante constatar que só a conjugação destes dois tipos de informação permite tomar decisões de encaminhamento com QoS. O conhecimento do estado das ligações permite a um encaminhador determinar a topologia da rede, com conhecimento das disponibilidades de largura de banda ou de outras grandezas. O segundo tipo de informação permite conhecer a ocupação que as reservas fazem da rede, de forma a evitar situações de instabilidade como a que descrevemos atrás, quando forem tomadas novas decisões de encaminhamento.

Um dos problemas que se levanta é o da quantidade de informação relativa ao estado das ligações e reservas, que circulam na rede para construir BD adequadas nos encaminhadores. Duma correcta abordagem ao problema poderá surgir uma solução escalável a redes de grandes dimensões ou, pelo contrário, uma solução que dificilmente poderá ser adoptada na prática.

Voltaremos a este assunto mais adiante, quando falarmos no QoSPF.

Encaminhamento pela Fonte vs. Encaminhamento Salto a Salto

Tradicionalmente, no encaminhamento de pacotes, numa rede sem ligações, ou na escolha de um percurso para um circuito virtual, numa rede com ligações, existem duas possibilidades distintas quanto à forma de calcular um percurso.

O percurso pode ser calculado apenas uma vez pelo nó de entrada na rede ou, pelo contrário, pode ser calculado em todos os nós de encaminhamento por onde vai passando. No primeiro caso, o percurso já está determinado à saída do primeiro nó [13], e diz-se que o encaminhamento é feito pela origem (*source routing*). No segundo caso, cada nó de encaminhamento limita-se a escolher o próximo salto do percurso (*next hop*) e o encaminhamento diz-se feito *salto a salto* (*next hop routing*). Note-se que, quando o encaminhamento é feito pela origem, os pacotes têm que levar a informação relativa ao percurso seleccionado.

Não podemos afirmar peremptoriamente que um dos métodos seja superior ou inferior ao outro. No caso do IP, onde não havia exigências de QoS, talvez fizesse mais sentido usar o encaminhamento salto a salto, por ser possível calcular previamente todas as saídas, para todos os destinos possíveis, e armazenar essa informação em tabelas.

Numa rede com QoS, onde é mais complicado sincronizar nos encaminhadores a informação de estado relativa às ligações físicas e reservas e onde é muito mais complicado construir tabelas com pares {destino, saída} — voltaremos a este assunto mais adiante — parece fazer mais sentido utilizar encaminhamento pela origem, por dois motivos essenciais:

- para eliminar a possibilidade de haver ciclos;
- para poupar processamento.

Outra vantagem de fazer encaminhamento pela origem está na possibilidade de se usarem algoritmos diferentes nos encaminhadores (estamos a admitir que só estes é que

[13] Isto não tem que ser estritamente assim. O percurso pode ser apenas especificado duma forma vaga, ficando a especificação detalhada a cargo de outros encaminhadores das zonas em que a especificação foi apenas parcial.

determinam o percurso). Este poderá ser um factor de diferenciação entre fabricantes e é, também, menos um problema de normalização a resolver.

Granularidade da Decisão de Encaminhamento

Nas redes IP actuais, a decisão de encaminhamento é baseada, exclusivamente, no endereço de destino dos pacotes. Esta é uma solução que nitidamente não serve numa rede que pretende oferecer garantias de QoS. Com efeito, esta situação implica que todos os pacotes dirigidos a um determinado destino são tratados de forma indiscriminada, independentemente da sua origem e, também, independentemente dos requisitos de QoS que lhe estiverem associados.

No entanto, esta é a solução mais económica em termos de necessidades de armazenamento dos encaminhadores. Como já sabemos, estes necessitam apenas de uma tabela para fazer o encaminhamento, sendo esta tabela indexada pelos destinos e permitindo obter uma saída.

Uma solução ligeiramente melhor consiste em utilizar o par {origem, destino} para fazer o encaminhamento. Aqui, o problema da determinação dos requisitos de QoS põe-se ao nível dos fluxos que tenham a mesma origem e o mesmo destino, que terão forçosamente que utilizar o mesmo percurso.

Uma terceira hipótese consiste em atribuir uma identificação a cada fluxo — *fid* (identificador de fluxo). Nesta alternativa, o encaminhamento é feito com base no trio {origem, destino, *fid*}. A vantagem óbvia é que os percursos dos fluxos individuais de dados podem ser totalmente independentes. Desta forma, é possível oferecer garantias de QoS às aplicações individualmente (ou mesmo a ligações diferentes das mesmas aplicações) em vez de serem oferecidas a um grupo de aplicações com reservas que partilham a mesma origem e o mesmo destino. As desvantagens também existem e prendem-se, por um lado, com a quantidade de informação a armazenar nos encaminhadores, por outro, com a quantidade de informação de actualização, do estado das ligações e reservas, distribuídas na rede, pese embora o facto de que esta quantidade dependa do protocolo de encaminhamento a adoptar. Por estes motivos não é claro

se esta, que aparenta ser a melhor solução, é escalável em termos de aplicação a uma situação real.

É interessante neste ponto estabelecer uma comparação com o esquema adoptado no ATM, que tem evidentemente de suportar a comutação das ligações individuais. Esta comutação é feita com base em dois identificadores presentes em todas as células ATM: VPI e VCI, respectivamente *Virtual Path Indicator* e *Virtual Channel Indicator*. Isto significa que cada ligação terá, à partida, informação relativa ao seu encaminhamento em todos os comutadores da rede que atravessa. À semelhança do que vínhamos a descrever para as redes IP, também no ATM, este facto representa um enorme esforço do ponto de vista da quantidade de informação a armazenar nos comutadores.

Para obviar este problema, é possível fazer a comutação com base apenas no VPI, agrupando-se, para o efeito, conjuntos de células de circuitos virtuais (VC — *Virtual Circuit*) relacionados. Estes VCs terão naturalmente, identificadores diferentes, porque os VCIs serão diferentes, mas os seus VPIs serão iguais. Esta possibilidade permite poupar recursos consideráveis nos comutadores. Pressupõe-se evidentemente que do ponto de vista da QoS faz sentido agrupar um conjunto de ligações — seria o caso de um conjunto de chamadas telefónicas, por exemplo.

Para outra abordagem a esta questão da granularidade veja-se (Argon *et al.*, 1998).

Cálculo Prévio dos Caminhos

Numa primeira análise, quando são considerados parâmetros de QoS num algoritmo de encaminhamento, não parece possível armazenar em memória um conjunto de decisões previamente calculadas. Isto porque, simplesmente, essas decisões dependem dos parâmetros de QoS considerados. Dito de outra forma, o percurso seleccionado pelo encaminhador depende dos parâmetros de QoS solicitados na reserva.

Ainda assim, (G.Apostolopoulos *et al.*, 1998) propõe um método para obter decisões pré-calculadas, quando é considerado apenas um parâmetros de QoS: largura de banda, sendo utilizado como critério de selecção o número de saltos (*hop count*) dum percurso (quanto menor o número de saltos mais favorável será o percurso).

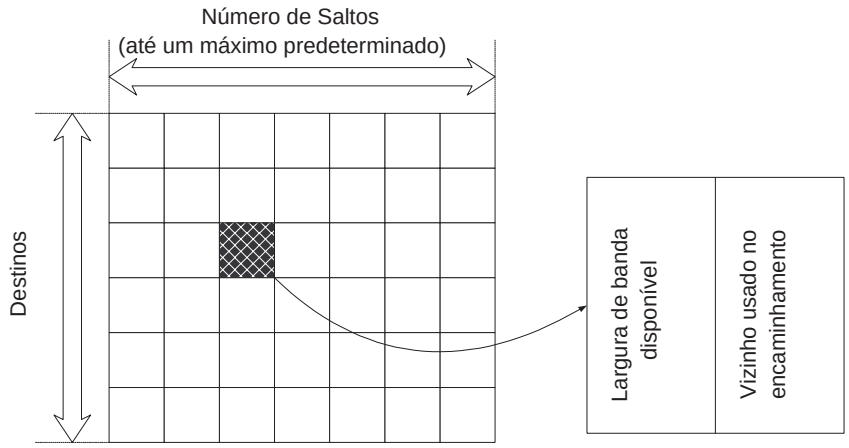


Figura 2.11: Tabela de encaminhamento pré-calculada, usando um algoritmo de Bellman-Ford

Para isso no algoritmo de Bellman-Ford é feita uma adaptação de forma a calcular a largura de banda máxima disponível em vez do CMC para um destino. Inerente ao próprio algoritmo está o método usado para determinar o número de saltos.

No final do algoritmo a tabela de encaminhamento terá a forma representada na figura 2.11. Vemos, então, que a tabela permite, para cada destino, armazenar todos os percursos possíveis em função do número de saltos. Para um número de saltos fixo é armazenado o vizinho imediato de saída e a largura de banda máxima admissível.

Além deste, é proposto outro algoritmo, baseado num dos algoritmos de Dijkstra, para compor uma tabela de encaminhamento. Neste caso, não é possível discriminar a largura de banda em termos quantitativos, mas apenas qualitativos, num número limitado de níveis. Se usássemos, digamos, três níveis, poderíamos ter largura de banda baixa, média e alta. Para esta situação concreta, a tabela de encaminhamento teria o aspecto apresentado na figura 2.12.

2.3.7 CAC e Acesso ao Meio

Há ainda alguns módulos na divisão proposta na figura 2.10, que desempenham um papel importante, mas aos quais não vamos dedicar muita atenção por excederem o âmbito deste documento.

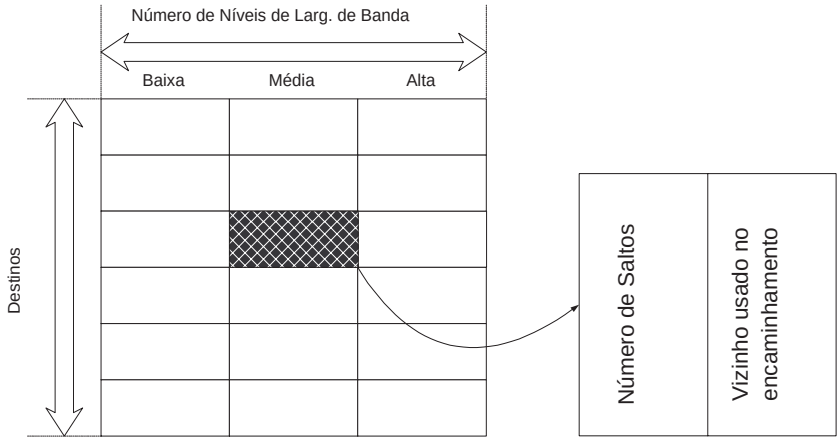


Figura 2.12: Tabela de encaminhamento pré-calculada, usando um algoritmo de Dijkstra

O Controlo de Admissão de Ligações (CAC — *Connection Admission Control*) — é responsável por aceitar ou rejeitar reservas solicitadas pelas aplicações. Este módulo ocupa, portanto, uma posição chave no que diz respeito às garantias de QoS. Do facto deste módulo ser mais ou menos permissivo depende a qualidade sentida pelas aplicações com reservas já estabelecidas (embora o CAC, só por si, não seja garante suficiente dessa qualidade, como teremos ocasião de ver) e também a taxa de ocupação das ligações físicas (ou posto de outra forma, a rentabilização do investimento em infra-estrutura).

O CAC constitui também mais um ponto de divergência entre fabricantes de encaminhadores, uma vez que, também aqui, é possível a utilização de algoritmos com características diferentes.

Note-se que o CAC pode constituir uma dificuldade adicional para os algoritmos de encaminhamento, que não conseguem determinar à partida se uma dada reserva vai ou não ser admitida. Mesmo que o algoritmo de CAC fosse conhecido [14], esta consideração continuaria a ser válida no caso em que a informação quanto ao estado das reservas não seja imediatamente propagada por toda a rede. Nesta situação, um encaminhador que selecciona um percurso, pode não ter a informação actualizada, que

[14] O ATM utiliza um algoritmo genérico que pretende aproximar o resultado do CAC utilizado nos vários nós. Este algoritmo é conhecido por GCAC (*Generic CAC*).

lhe permita avaliar correctamente a resposta do CAC de outro encaminhador.

Outro problema se levanta quando termina esta fase e assim que uma aplicação estabelece uma reserva, pois torna-se estritamente necessário controlar o envio de dados que essa aplicação faz para a rede. As razões para tal são as seguintes:

- a aplicação não deve poder enviar mais dados do que o que estava previsto no contrato. Esta regra é mais importante no caso de o contrato estabelecer valores monetários;
- uma aplicação mal comportada não deve prejudicar a QoS sentida pelas outras aplicações.

Para resolver estes assuntos é conveniente utilizar filas de acesso às ligações físicas separadas por classes de serviço, por exemplo (idealmente a separação seria feita ao nível dos fluxos individuais).

A separação dos dados (pacotes ou células) por filas é trabalho para um separador de pacotes e, a repartição da largura de banda pelas filas, cabe a um escalonador. Estas duas entidades, que aparecem na figura 2.10, são consideradas no modelo do RSVP. Veja-se (Braden *et al.*, 1997).

A abordagem deste tema está para além do âmbito desta tese, pelo que não vamos prosseguir o nosso raciocínio.

Daquela figura resta-nos por descrever a política de controlo que, em termos breves, determina que entidades podem ou não fazer reservas e que tipos de reservas.

2.4 *Quality of Service Extensions to OSPF* — QoSPF

2.4.1 Características Gerais

O QoSPF — “Quality of Service Extensions to OSPF” ou “Quality of Service Path First Routing” é um conjunto de extensões aos protocolos OSPF (ver (Moy, 1994b)) e

MOSPF. O *Multicast Extensions to OSPF* — MOSPF — é um conjunto de extensões ao protocolo OSPF, que visa dotar as redes IP de capacidade de encaminhamento *multicast*. O encaminhamento *multicast* é feito com base na origem e no destino dos datagramas. Veja-se (Moy, 1994a).

Na figura 2.10 o QoSPF desempenha o papel correspondente ao protocolo de encaminhamento. As extensões introduzidas pelo QoSPF, em conjunto com outros módulos, como sejam o de sinalização (lugar que poderá ser ocupado pelo RSVP) permitem que a rede ofereça garantias de QoS.

O QoSPF parte duma base solidamente instalada (o OSPF) e é uma proposta concreta cuja especificação está ainda em fase de desenvolvimento e pode ser encontrada em (Zhang *et al.*, 1997).

No QoSPF, a oferta de QoS baseia-se em dois tipos de mensagens novas, que não existiam na versão original do OSPF. O primeiro tipo de mensagens propaga informação do estado das ligações físicas adicionando, agora, descrições dos parâmetros de QoS considerados relevantes, enquanto que o outro tipo de mensagens propaga informação acerca do estado das reservas efectuadas.

A partir da informação assim obtida, as decisões de encaminhamento são tomadas em função dos endereços de origem e destino. Esta é uma alternativa intermédia, em termos de granularidade da decisão de encaminhamento como já tivemos ocasião de ver na subsecção 2.3.6, página 43.

Outra das características mais importantes deste protocolo é a capacidade de fazer *route pinning*, embora à custa de interacções com o módulo de sinalização, que decide quais os troços do percurso que são fixos.

Finalmente, no documento que descreve o QoSPF ((Zhang *et al.*, 1997)) estão previstas soluções para o problema da escalabilidade, onde o desempenho do protocolo enfrenta algumas dificuldades.

2.4.2 Propagação da Informação Relativa a Recursos

Em relação ao que acontecia no OSPF, podemos considerar que existem dois novos tipos de *Link State Advertisements* (avisos) que são particularmente relevantes no funcionamento do OSPF:

- *Link Resource Advertisements* (RES-LSA);
- *Resource Reservation Advertisements* (RRA).

Estes novos avisos são resumidamente descritos de seguida.

Link Resource Advertisements **(RES-LSA)**

Estes avisos são parecidos com os que descrevem as interfaces dos encaminhadores no protocolo original (*Router-LSA*). Descrevem as mesmas interfaces, mas em termos dos parâmetros de QoS relevantes. Desta forma, permitem aos encaminhadores construir uma topologia baseada nesses parâmetros em vez duma topologia baseada numa ou mais métricas simples (que se obtinham com TOS $= 0$ ou TOS $\neq 0$).

Os parâmetros de QoS são representados por parâmetros de "balde de testemunhos" ("token bucket") e em termos de disponibilidade de largura de banda.

O balde de testemunhos é um mecanismo de formatação de tráfego, que se descreve à custa de dois parâmetros: débito, ρ, e capacidade do balde, β. Uma representação gráfica do modelo está ilustrada na figura 2.13. O débito é, como o nome indica, a quantidade média de tráfego que, neste caso, a rede tem capacidade de suportar, embora também possa representar a quantidade média de tráfego a gerar por uma aplicação. Podemos pensar no débito, ρ, como sendo resultante da frequência com que são colocados testemunhos de um determinado tamanho fixo dentro do balde. Para que a aplicação envie um pacote de um determinado tamanho, digamos, P, o total de testemunhos no balde, N, tem que perfazer pelo menos esse tamanho (i.e., $N \geq P$). Após o envio do pacote, o número de testemunhos correspondente ao seu tamanho é subtraído do balde. Caso o tamanho dos testemunhos seja inferior ao do pacote, este fica retido

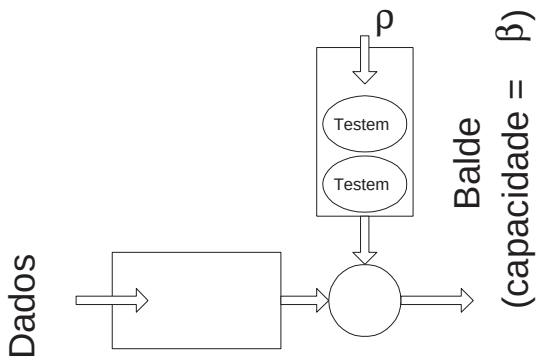


Figura 2.13: Modelo “balde de testemunhos”

até que se acumulem testemunhos em quantidade suficiente. O tamanho máximo do balde, β, é, portanto, o tamanho máximo dos testemunhos que se podem acumular em qualquer instante.

Este modelo garante que em termos médios o débito duma aplicação não excede ρ e que o tamanho das suas rajadas não excede $\beta + \tau/\rho$, no intervalo de tempo τ. Em termos de descrição da capacidade da rede, os parâmetros do balde de testemunhos terão um significado diferente: largura de banda média disponível e variação máxima do fluxo transportado.

Em conjunto com os parâmetros de balde de testemunhos, que descrevem a capacidade remanescente da ligação, é distribuído também o atraso em termos temporais introduzido por essa mesma ligação.

Na figura 2.14 está representado um aviso RES-LSA.

Como se compreende, tem que ser encontrada uma solução de compromisso para a política de propagação de RES-LSA. Por um lado, convém que os avisos sejam propagados sempre que há alterações, para que possam ser tomadas decisões tão próximas do óptimo quanto possível, ao nível de toda a rede. Por outro, a adopção desta política resultará num aumento descontrolado de avisos, com o aumento das reservas efectuadas.

Assim, o protocolo define que, além das situações anteriormente previstas pelo OSPF para propagar avisos, os RES-LSA devem ser propagados nas seguintes

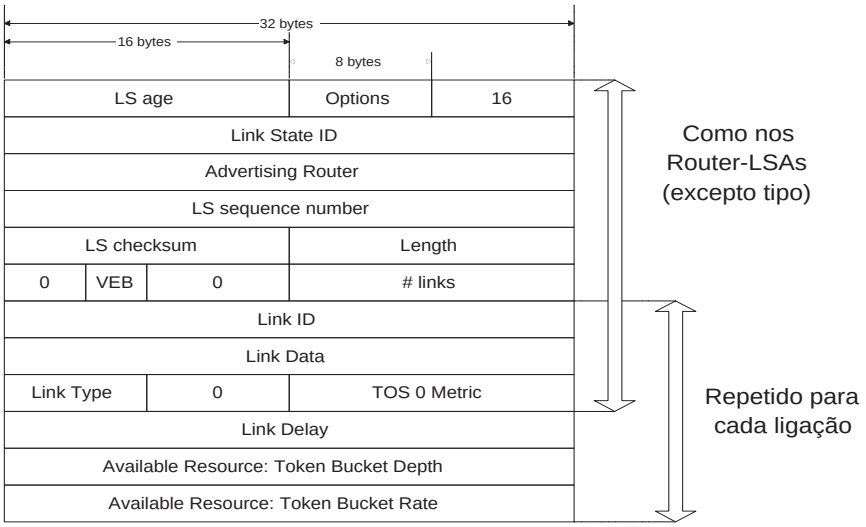


Figura 2.14: Aviso RES-LSA

situações:

- quando a largura de banda disponível ou o atraso duma ligação variam consideravelmente (embora a ordem de grandeza destes valores não seja quantificada na especificação);
- quando uma reserva falha num determinado encaminhador (o que significa que outro encaminhador tem indicações erradas). Neste caso, o encaminhador onde a reserva falhou, tem que propagar informação relativa às suas interfaces.

Resource Reservation Advertisements **(RRA)**

Estes avisos descrevem as reservas efectuadas nas interfaces dum encaminhador (internas a uma área) também sob a forma de parâmetros balde de testemunhos. Estas reservas são sempre feitas em termos de fluxos que, como já dissemos, são definidos, neste protocolo, pelo par {origem, destino}.

Na figura 2.15 está representado o formato do pacote onde é feito o envio de RRAs. Note-se que esta informação é, por sua vez, encapsulada em avisos LSA, que já existiam no OSPF. A utilidade dos avisos RRA pode não ser compreensível numa primeira análise, uma vez que, aparentemente, é suficiente conhecer os recursos disponíveis e

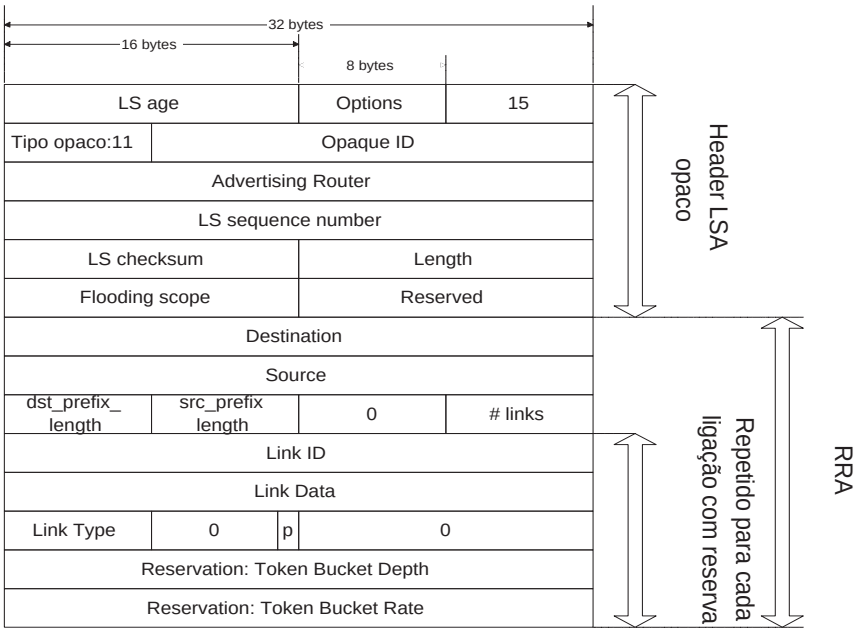


Figura 2.15: Aviso RRA

os recursos solicitados para fazer o encaminhamento com QoS. Ora, essa informação já é disponibilizada pelos avisos RES-LSA.

No entanto, num protocolo de sinalização como o RSVP a determinação dos caminhos e as reservas propriamente ditas são feitas em tempos diferentes. Sempre que surge uma reserva são propagados novos avisos RRA. Isto permite aos encaminhadores tomarem conhecimento das reservas e alterarem, eventualmente, as decisões de encaminhamento relativas ao fluxo em causa.

Outra das razões que motiva a existência de avisos RRA é que estes servem de realimentação (*feedback*) para os encaminhadores, o que lhes permite considerar os recursos consumidos pelos fluxos, quando têm que recalcular caminhos para esses mesmos fluxos. Esta realimentação permite obter decisões de encaminhamento estáveis, como vimos na subsecção 2.3.6.

2.4.3 Cálculo dos Percursos

O esquema proposto para calcular um percurso tomando em consideração parâmetros de QoS é bastante simples. O algoritmo é o de Dijkstra, onde o custo considerado é o atraso previsto das ligações físicas. A questão é que, agora, o algoritmo não vai considerar todas as ligações físicas existentes, mas apenas aquelas que dispõem de largura de banda suficiente para os requisitos exigidos pelo fluxo.

Em termos computacionais, este algoritmo é simples, mas não está previsto nenhum mecanismo capaz de fazer o cálculo prévio dos percursos, pelo que estes terão de ser calculados em tempo real, sempre que aparecer um novo, ou houver alterações nalgum fluxo.

De seguida vamos abordar alguns problemas relacionados com o cálculo dos percursos, quer para destinos *unicast*, quer para destinos *multicast*.

QoSPF *unicast*

O cálculo dum CMC com garantias de QoS é relativamente simples, no caso do percurso estar todo contido dentro da mesma área. Aqui, como os encaminhadores conhecem toda a topologia da área, a tarefa não se apresenta difícil.

Quando o percurso atravessar mais do que uma área, a situação é mais complexa. Neste caso, começa-se por determinar o CMC em termos das métricas tradicionais. O percurso obtido permite identificar as áreas e os ABRs (*Area Border Routers*) que serão envolvidos no percurso final (note-se que esta decisão é, eventualmente, subóptima). No interior de cada área terão de ser determinados os percursos com QoS, desde a origem (ou entrada na área) até ao destino (ou saída da área) tal como estes foram determinados pelo percurso melhor esforço calculado inicialmente.

Esta solução padece, porém, de um problema difícil de resolver. Um ABR de entrada na área de destino, por exemplo, que anunciou o CMC melhor esforço para essa área pode não estar contido em nenhum caminho com as garantias de QoS exigidas até ao destino. Isto é um problema que em situações normais não é detectável pelos encaminhadores das áreas anteriores.

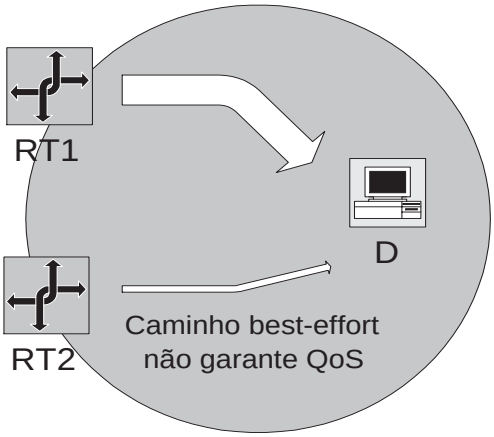


Figura 2.16: Do encaminhador RT2 para o destino D não existe nenhum percurso com as garantias de QoS exigidas

A figura 2.16 pretende ilustrar esta situação: o CMC melhor esforço (*best-effort*) passa por RT2, mas de RT2 para D não há nenhum caminho capaz de oferecer as garantias de QoS requeridas. Este tipo de problemas e a solução proposta são assuntos abordados em (Zhang *et al.*, 1997).

QoSPF *multicast*

Quando a origem e os destinos se encontram na mesma área, o problema resume-se a determinar a árvore de distribuição, com base no atraso propagado e usando as ligações que têm capacidade para a reserva pedida.

Quando a árvore de distribuição atravessa várias áreas, o problema é mais complicado. A ideia é que as áreas a jusante da origem tomam para raiz todos os ABR que suportam o MOSPF e que anunciam a origem nos seus sumários LSAs (isto significa que esses ABR anunciam percursos até à origem e que provavelmente também há percursos no sentido inverso).

As árvores construídas pelas várias áreas são combinadas numa única árvore de distribuição global. Também aqui, alguns dos ABRs, que no caso melhor esforço entrariam na árvore de distribuição, são postos de lado devido às exigências de QoS. A forma de resolver este problema é abordada em (Zhang *et al.*, 1997).

2.4.4 *Explicit Routing OSPF* — EROSPF

A solução que temos vindo a apresentar padece dum problema de escalabilidade que diz respeito à gestão de informação de reservas (número de RRAs gerados). Para minorar este problema é proposta a utilização de encaminhamento pela fonte, em vez de encaminhamento salto a salto, designada por "Explicit Routing OSPF" — EROSPF.

A ideia é que só o encaminhador onde é primeiro encaminhado o fluxo (eventualmente mais do que um, se estivermos a falar de vários ABRs numa área que não é a origem do fluxo) efectua o cálculo do algoritmo de Dijkstra e também só este encaminhador é que mantém informação das reservas relativas a esse fluxo. Haverá, portanto, uma redução drástica no espaço necessário para armazenar RRAs, no esforço despendido em propagar RRAs, que agora seguem directamente para a origem dos fluxos, e no cálculo dos CMC.

Note-se que, quando o encaminhamento é feito salto a salto, não é suficiente propagar os RRAs apenas para os encaminhadores a montante do nó que faz a reserva. Isto porque, se houver alguma alteração das condições da rede, mesmo os encaminhadores que não entrem inicialmente no percurso, poderão passar a fazer parte e ser chamados a calcular um percurso para esse fluxo. Ora, o encaminhador recém entrado no percurso, terá que conhecer as reservas já afectadas ao fluxo em causa para tomar a sua decisão de uma forma correcta. Por este motivo, todos os encaminhadores têm que receber os RRAs relativos a fluxos que não lhes dizem respeito.

Em (Zhang *et al.*, 1997) é feita uma comparação em termos de estimativa entre o número de RRAs geradas por segundo pelo QoSPF com ou sem *Explicit Routing* (ER), que reproduzimos na tabela 2.10. F é o número médio de fluxos com origem num encaminhador.

N.º de encaminhadores:	9	16	25	36	49	64
N.º de RRAs por seg. c/ ER:	$0.3F$	$0.6F$	$1.0F$	$1.5F$	$2.1F$	$2.8F$
N.º de RRAs por seg. s/ ER:	$2.6F$	$6.3F$	$9.9F$	$21.5F$	$34.2F$	$51.1F$

Tabela 2.10: Comparação do QoSPF com e sem *Explicit Routing*

2.4.5 PQC — *Path QoS Collection Method*

Um método denominado *Path QoS Collection Method* (PQC) é proposto por (Goto *et al.*, 1997) para resolver o problema da escalabilidade associado ao QoSPF.

A ideia é que o PQC será usado em conjunto com o RSVP e as mensagens PATH do RSVP incluiriam uma compilação da informação sobre os recursos usados por um fluxo, dispensando assim a realimentação fornecida pelos RRAs. Em conjunto com os avisos com informação de QoS será então possível a um encaminhador determinar um percurso com QoS para um dado fluxo, considerando a largura de banda eventualmente já consumida por esse fluxo, eliminando desta forma o problema da instabilidade na determinação dum caminho.

Para mais detalhes veja-se (Goto *et al.*, 1997).

2.5 Conclusão

Como temos vindo a ver ao longo deste capítulo, o número e a diversidade dos parâmetros de QoS a considerar numa rede que ofereça serviços com garantias pode ser elevado. Vimos também que, ao nível dos algoritmos de encaminhamento, apenas um parâmetro é geralmente considerado, no momento em que este texto está a ser escrito: a largura de banda.

No QoSPF, por exemplo, a largura de banda é o único parâmetro de QoS considerado, caracterizada em termos de parâmetros de balde de testemunhos. O atraso considerado no protocolo não é mais do que o comprimento das ligações físicas, em termos de tempo que estas levam a ser atravessadas. Com esta informação, de natureza estática, não é possível fazer garantias quanto ao tempo de atraso na entrega dos pacotes, uma vez que em geral este aumenta com o aumento do tamanho dos tampões, não dependendo apenas do atraso na propagação do sinal no meio físico de transmissão.

É importante, neste ponto, estabelecermos os requisitos que um algoritmo ou dispositivo de encaminhamento com QoS deve apresentar. Uma hipótese é codificar os

parâmetros de QoS em termos de custos, atribuir um custo a cada ligação física, eliminando as que não satisfaçam todas as exigências de QoS impostas no pedido, e aplicar um método de optimização ao grafo assim definido. Se, eventualmente, houver mais do que um parâmetro de QoS, o custo tem que ser obtido a partir duma expressão qualquer que pondere todo os parâmetros.

Levantam-se aqui outras hipóteses mais ou menos sofisticadas que poderiam considerar o estado actual da rede, a sua evolução passada, o valor dos parâmetros de QoS, além de outros critérios capazes de encontrar pontos de funcionamento mais satisfatório, quer para a entidade responsável pela rede, quer para os utilizadores de aplicações (embora estes sejam normalmente objectivos antagónicos).

Nesta tese vamos considerar apenas a primeira hipótese. Assim sendo, uma vez que o algoritmo de Dijkstra permite determinar o CMC dadas as condições descritas, qualquer alternativa — nomeadamente uma RN — tem que ser mais rápida quando concretizada para ser uma alternativa que valha a pena ser considerada. Note-se que uma solução baseada em RN não oferece a garantia de encontrar o CMC.

É de referir que, mesmo para o algoritmo de Dijkstra, que obtém sempre o percurso óptimo, não é óbvio que a rede atinja o melhor ponto possível de funcionamento. Há duas razões essenciais para isto. Em primeiro lugar, os percursos sucessivamente obtidos pelo algoritmo de Dijkstra dependem da ordem pela qual surgem os pedidos de estabelecimento de reservas. Os mesmos pedidos, apresentados por ordem diferente podem conduzir a rede para estados finais diferentes. Além disto, o óptimo encontrado em cada execução do algoritmo refere-se à forma como os vários parâmetros são codificados em custos e estes custos podem não ter uma correspondência exacta com a realidade. Este dois problemas ultrapassam o âmbito desta tese, motivo pelo qual não vamos tentar resolvê-los.

Posto isto, nesta tese, vamos apresentar soluções que consideram como parâmetros de QoS apenas a largura de banda. Outro facto que motiva esta decisão prende-se com o estado actual dos simuladores de rede existentes, como teremos ocasião de ver no capítulo 5. É de salientar, no entanto, que iremos caracterizar a largura de banda em termos de valor médio, por uma questão de simplicidade. Esta opção apresenta, como

é natural, algumas limitações que a seu tempo não deixarão de se manifestar.

No capítulo seguinte vamos fazer uma panorâmica sobre as RN, com vista a determinar qual ou quais as RN que melhor se adaptam à tarefa que nos propomos realizar.

3

Encaminhamento com Redes Neuronais

3.1 Redes Neuronais — Breve Introdução

3.1.1 Aplicações

Uma rede neuronal artificial é um dispositivo de cálculo inspirado nas redes neuronais existentes na natureza, cujo elemento constituinte básico é o neurónio.

Existem vários tipos de redes neuronais, que diferem entre si, quer na configuração em que são dispostos os seus neurónios, quer nas regras matemáticas que regem o funcionamento destes mesmos neurónios.

Cada um destes tipos de rede foi concebido para desempenhar um conjunto específico de tarefas. No seu conjunto, e de uma forma geral, as redes neuronais são capazes de desempenhar várias, de entre as quais salientamos as seguintes:

- simular funções: dado um conjunto inicial de pares da forma {valor de entrada da função, valor de saída da função} é possível treinar a RN de forma a que a partir destes exemplos, chamados *de treino*, seja possível generalizar e aproximar a função em causa, i.e., calcular o valor da saída para um determinado valor de entrada não existente no conjunto original. Esta capacidade é também aplicada na simulação de sistemas físicos, quer dinâmicos, quer estáticos;
- fazer reconhecimento de padrões: alguns tipos de redes neuronais permitem reconhecer padrões previamente armazenados a partir de amostras incompletas ou danificadas (por existência de ruído, por exemplo);

- classificar: identificar a classe de entre um número finito de classes a que pertence um objecto. Na prática, a classificação reduz-se à aproximação de uma função cujas saídas são valores discretos;

- resolver problemas de optimização: encontrar a solução óptima para um determinado problema que envolva custos (em sentido lato). Geralmente, os problemas que se pretendem resolver não são solúveis em tempo útil (polinomial) por algoritmos sequenciais. Noutros casos, como aqueles que apresentamos nesta tese, as RN obtêm resultados eventualmente inferiores, mas num intervalo de tempo que se espera mais curto.

3.1.2 Neurónio

O neurónio é a unidade de cálculo elementar que compõe uma rede neuronal. Quando considerado individualmente, um neurónio efectua apenas um conjunto de cálculos muito simples, que se resumem, normalmente, a uma adição e à determinação do valor de saída duma função não linear. No entanto, quando interligado de uma forma estruturada com outros neurónios, torna-se capaz de produzir computação útil.

O cálculo do valor de saída efectuado em cada neurónio é feito em função dos valores que lhe são aplicados à entrada[1]. Se designarmos as N entradas dos neurónios por x_i, com $1 \leq i \leq N$ e a saída por y, a relação entre elas é dada pela equação 3.1.

$$y = \varphi\left(\sum_{i=1}^{N} w_i x_i + \theta\right) \tag{3.1}$$

Na equação 3.1, w_i é o peso que afecta a entrada i do neurónio. Note-se que como a cada entrada está associado um peso específico, as entradas contribuem de forma diferente no cálculo do valor da saída. Isto significa que é possível regular o nível de interacção existente entre dois neurónios vizinhos.

[1] É de referir que este cálculo, normalmente, é de natureza determinista mas não tem necessariamente que o ser, havendo uma classe de neurónios com funcionamento estocástico, nomeadamente nas *máquinas de Boltzmann* (Haykin, 1994).

Ainda na mesma equação, θ é conhecido por *viés* ou *limite* (respectivamente *bias* e *threshold* em inglês) e representa um valor imposto a partir do exterior.

Finalmente, $\varphi(\cdot)$ é conhecida por função de activação e destina-se a limitar o valor da saída do neurónio. Este limite situa-se, normalmente, entre -1 e 1 ou entre 0 e 1, podendo a saída da função ser discreta ou contínua, se o valor da saída puder assumir apenas os valores extremos ou qualquer valor no intervalo definido por eles, respectivamente. Mais adiante vamos apresentar um conjunto de funções de activação utilizadas nas arquitecturas de redes mais vulgares.

A expressão $\sum_{i=1}^{N} w_i x_i + \theta$, aplicada à entrada da função de activação, é conhecida por nível interno de actividade do neurónio, v, de tal forma que a equação 3.1 pode ser rescrita à custa de duas relações:

$$\begin{aligned} v &= \sum_{i=1}^{N} w_i x_i + \theta \\ y &= \varphi(v) \end{aligned}$$

Ao longo deste texto, a saída y, e o viés θ de um dado neurónio j são representados, respectivamente, por y_j e θ_j. Ao peso w_i, que afecta a entrada i do neurónio j, é também adicionado o índice j, sendo então o peso da ligação de i para j representado por w_{ji}.

A figura 3.1, que representa o neurónio expresso na equação 3.1, pretende ilustrar com clareza a divisão do neurónio em três componentes essenciais:

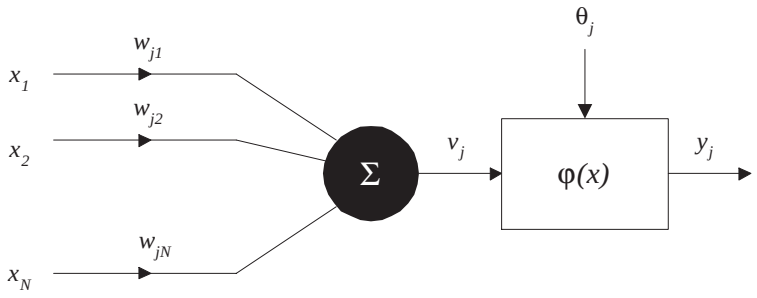


Figura 3.1: Modelo do neurónio

- entradas e respectivos pesos;

- unidade somatória;
- função de activação.

Numa representação alternativa, o viés poderia surgir como sendo uma das parcelas aplicadas à unidade somatória (sendo a entrada fixa a -1 e o peso dessa entrada a θ_j).

Tipos de Funções de Activação

Neste documento vamos apresentar apenas os três tipos mais importantes de funções de activação. A primeira, e mais simples, está expressa na equação 3.2, estando o gráfico correspondente representado na figura 3.2. Esta função é conhecida por *limite* ou *degrau*. O modelo de neurónio representado pela equação 3.1 e com esta função de activação é designado por modelo de *McCulloch-Pitts* ((McCulloch & Pitts, 1943)). Neste caso, a entrada θ costuma representar um limite de actividade do neurónio, acima do qual o neurónio estará activo (saída a 1) e abaixo do qual estará inactivo (saída a -1 ou 0 conforme a representação escolhida). Por θ ser um limite, a equação 3.3 será preferível para exprimir o modelo de McCulloch-Pitts, embora a diferença esteja apenas no sinal do limite.

$$\varphi(x) = \begin{cases} 0 & \text{se } x < 0 \\ 1 & \text{se } x \geq 0 \end{cases} \tag{3.2}$$

$$y = \varphi\left(\sum_{i=1}^{N} w_i x_i - \theta\right) \tag{3.3}$$

Um outro tipo de função de activação, também relativamente comum, é chamada de *linear por partes*. Esta função de activação está expressa na equação 3.4 e representada na figura 3.3.

$$\varphi(x) = \begin{cases} 0 & \text{se } x < -\frac{1}{2} \\ x + \frac{1}{2} & \text{se } -\frac{1}{2} \leq x \leq \frac{1}{2} \\ 1 & \text{se } x > \frac{1}{2} \end{cases} \tag{3.4}$$

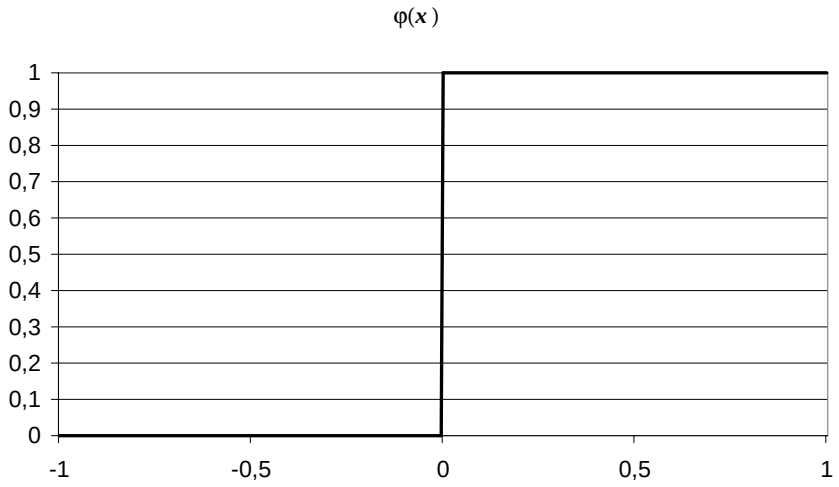


Figura 3.2: Função de activação *limite* ou *degrau*

A função de activação mais importante e mais vulgarmente utilizada é conhecida por

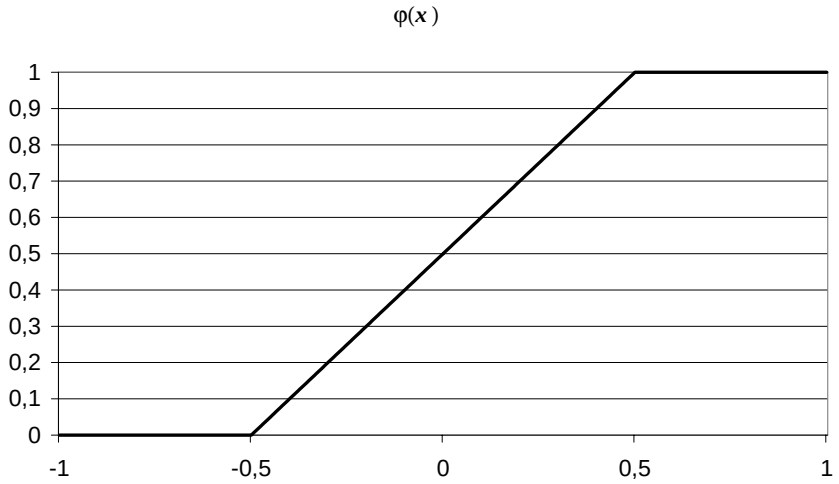


Figura 3.3: Função de activação *linear por partes*

sigmóide e caracteriza-se por ser crescente e ter curvas suaves (i.e., é derivável em todos os seus pontos). Esta função de activação está expressa na equação 3.5 e o gráfico respectivo está representado na figura 3.4. Na equação, α é o *declive* da função sigmóide. Quanto maior for este declive mais a função sigmóide se aproxima da função degrau.

$$\varphi(x) = \frac{1}{1 + e^{-\alpha x}} \tag{3.5}$$

Note-se que, nas definições das funções de activação, considerámos que os valores de saída estavam definidos entre 0 e 1. Se, em vez destes quiséssemos considerar valo-

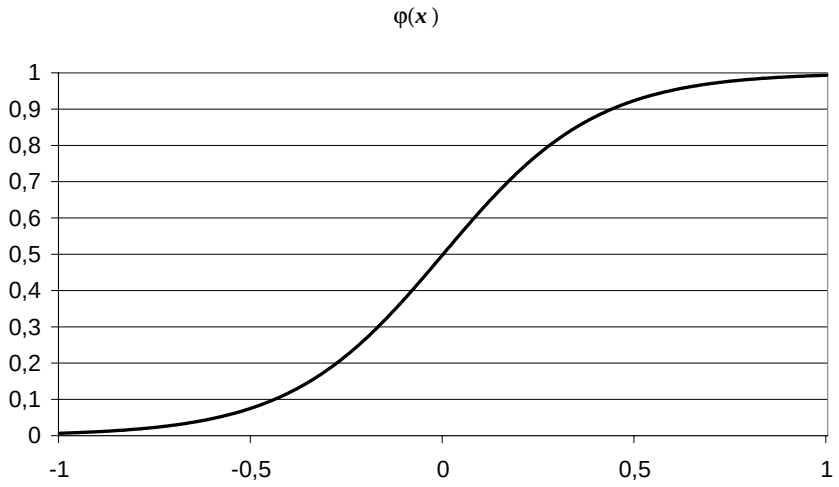


Figura 3.4: Função de activação *sigmóide*

res entre -1 e 1, teríamos que introduzir algumas alterações, em particular no caso da função de activação sigmóide[2].

3.1.3 Redes de Hopfield

A apresentação das diversas arquitecturas de redes neuronais existentes bem como respectivos fins a que se destinam, ultrapassa o âmbito deste documento. Como tal, vamos concentrar-nos, apenas, no estudo das redes de Hopfield (ver (Hopfield & Tank, 1986)).

Consideremos uma rede neuronal como a da figura 3.5, onde todos os neurónios são interligados entre si. Se as ligações entre os neurónios forem simétricas, i.e., se $w_{ij} = w_{ji}$, $i, j = 1, ..., N$, onde N é o número de neurónios da rede, então esta diz-se uma *rede de Hopfield*. Uma representação alternativa para esta rede está ilustrada na figura 3.6, onde se põe em evidência que uma rede de Hopfield é composta por uma única camada com realimentação (note-se que não há qualquer camada escondida). Note-se porém que, neste caso, não há *auto-realimentação*, visto que as saídas dos neurónios não são reaplicadas nas suas próprias entradas. Como teremos ocasião de verificar isto não acontece em todas as redes de Hopfield, havendo algumas em que os

[2]Neste caso, a função de activação poderia ser uma tangente hiperbólica. Após algumas manipulações teríamos: $\varphi(x) = \frac{1-e^{-\alpha x}}{1+e^{-\alpha x}}$.

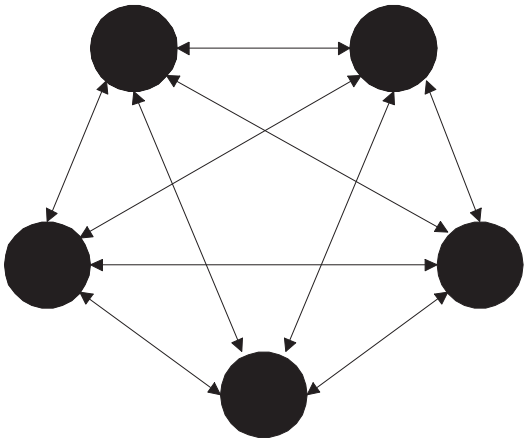


Figura 3.5: Rede neuronal totalmente interligada

neurónios podem apresentar auto-realimentação.

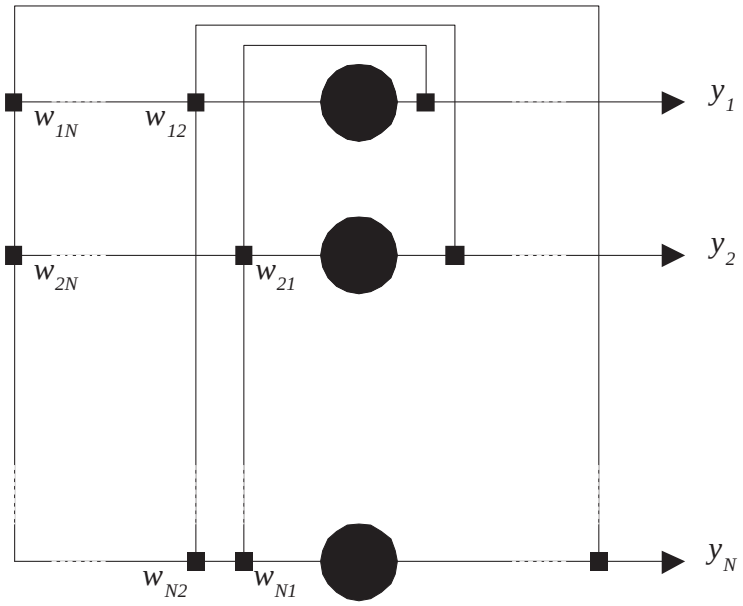


Figura 3.6: Rede de Hopfield binária, com os neurónios representados por um círculo preto e as ligações representadas por um quadrado também preto

Esta construção levanta algumas questões pois, como consequência da própria arquitectura, uma alteração da saída de um único neurónio repercute-se nas entradas de todos os outros. Estas alterações podem, por sua vez, provocar novas alterações nas saídas destes neurónios e assim sucessivamente. Põe-se, então, a questão de saber se a rede converge para um estado final ou se, pelo contrário, diverge indefinidamente. Caso não exista convergência o cálculo produzido pela rede será de pouca utilidade.

Hopfield demonstrou que, desde que os pesos das ligações entre os neurónios sejam simétricos ($w_{ij} = w_{ji}$), é possível garantir que a rede converge. Isto não implica, porém, que uma rede com pesos assimétricos divirja forçosamente (rede diverge $\Rightarrow$ pesos assimétricos; pesos assimétricos $\not\Rightarrow$ rede diverge).

É importante referir, neste ponto, que os pesos das ligações entre os neurónios são previamente fixados, de forma que as redes de Hopfield são incapazes de aprender. Dispensam uma fase de treino e funcionam, portanto, de uma forma não supervisionada. O processo utilizado para calcular o valor dos pesos a atribuir às ligações depende da aplicação a que se destina a rede.

Antes de abordarmos a forma de utilizar a convergência destas redes para fins concretos, vamos proceder a uma divisão adicional das redes de Hopfield em duas classes: redes de Hopfield discretas e redes de Hopfield contínuas.

Redes de Hopfield discretas ou binárias

Esta arquitectura foi apresentada por Hopfield em 1982 ((Hopfield, 1982)). O neurónio utilizado nestas redes é o de McCulloch-Pitts, sendo, portanto, utilizada a função de activação degrau. Neste tipo de neurónios, a saída assume valores discretos, razão pela qual a rede é conhecida por discreta ou binária. A dinâmica deste tipo de rede pode ser representado pelas equações 3.6 e 3.7.

$$y_j = \varphi_j \left(\sum_{i=1}^{N} w_i x_i - \theta_j \right) \qquad j = 1, \ldots, N \tag{3.6}$$

$$x_j(t) = y_j(t-1) \tag{3.7}$$

Na equação 3.7, para simplificar a discussão, foi introduzido um atraso, que representa o tempo de reacção do neurónio e o tempo de propagação da saída, para as entradas dos outros neurónios. Este atraso permite discretizar o tempo e considerar que a saída de cada neurónio só afecta as entradas dos outros neurónios, no instante de tempo seguinte. As alterações necessárias ao circuito da figura 3.6 estão ilustradas na figura 3.7. No que diz respeito à ordem pela qual é feita a actualização dos valores das saídas dos

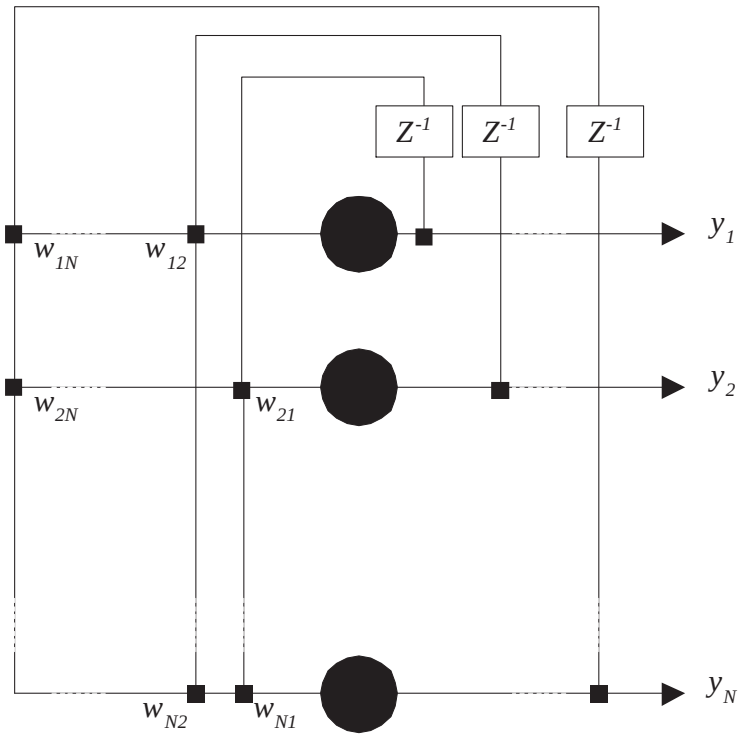


Figura 3.7: Rede de Hopfield binária, com atrasos unitários aplicados aos valores das saídas

neurónios, Hopfield propôs que essa ordem fosse determinada aleatoriamente sendo, em cada instante de tempo, seleccionado de uma forma aleatória, um único neurónio para actualização.

Hipóteses alternativas consistem em actualizar os valores das saídas sequencialmente, respeitando uma ordem previamente estabelecida entre os neurónios ou então, fazer as actualizações em paralelo. Em qualquer dos casos, a actualização termina quando forem percorridos todos os neurónios da rede sem se verificarem alterações em nenhuma das saídas. Quando isso acontecer a rede terá convergido para um determinado estado final, que se espera representar uma resposta útil. Apenas no caso em que a actualização é feita em paralelo não existe garantia de convergência.

Uma das vantagens de fazer a actualização das saídas dos neurónios por uma ordem aleatória consiste na possibilidade de a rede dar respostas distintas, i.e., estabilizar em estados finais distintos a partir de um mesmo estado inicial o que, por vezes, é uma vantagem, em particular quando a rede não responde satisfatoriamente às primeiras

tentativas.

As redes de Hopfield binárias servem essencialmente como *Memórias Endereçáveis pelo Conteúdo* — CAM (Content-Addressable Memory). A ideia chave consiste em armazenar nos pesos das ligações entre neurónios um conjunto limitado de padrões. Por padrão entende-se uma palavra composta por dígitos binários (*bits*), cujo comprimento deverá ser igual ao número de neurónios da rede. A cada dígito binário corresponderá um neurónio.

A memória diz-se endereçável pelo conteúdo porque, aplicando aos neurónios partes de algum dos padrões previamente armazenados ou um dos padrões, mas corrompido, a rede converge para um estado final que corresponde ao padrão que mais se aproxima do padrão aplicado inicialmente. É importante referir que isto nem sempre acontece e que a memória pode evocar um padrão inexistente, ou outro padrão, que não o mais semelhante.

Suponhamos que se pretende armazenar numa rede com N neurónios os m padrões seguintes, A_1, A_2, ..., A_m, sendo o dígito binário na posição j do padrão i, representado por a_{ij}:

$$\begin{aligned} A_1 &= (a_{11}, a_{12}, \ldots, a_{1N}) \\ A_2 &= (a_{21}, a_{22}, \ldots, a_{2N}) \\ &\vdots \\ A_m &= (a_{m1}, a_{m2}, \ldots, a_{mN}) \end{aligned}$$

Para o fazer, temos que determinar os pesos de tal forma que, a partir dum estado inicial qualquer, a rede convirja sempre para o padrão que mais se aproxima desse estado inicial, de entre os m padrões armazenados.

A expressão exacta que permite determinar os pesos das ligações, em função dos padrões a armazenar, depende da função de activação dos neurónios da rede. Se a saída dos neurónios puder assumir os valores -1 e 1, o peso da ligação entre o neurónio

i e o neurónio j será determinado pela equação 3.8.

$$\begin{aligned} w_{ji} &= \sum_{p=1}^{m} a_{pi} a_{pj} \qquad i \neq j \\ w_{ii} &= 0 \end{aligned} \tag{3.8}$$

O somatório da equação 3.8 é, em alguns documentos, afectado da constante $1/N$ (onde N é o número de neurónios da rede), para simplificar alguns cálculos relativos à análise de probabilidades de erro e capacidade de armazenamento da rede de Hopfield binária (secção 8.6, (Haykin, 1994)).

Resulta da equação 3.8, que a determinação dos pesos das ligações é efectuada com base no mesmo princípio em que se baseia a regra de Hebb. Esta regra diz que, quando o neurónio j toma repetidamente parte na excitação do neurónio k, o peso da ligação de j para k deve ser incrementado[3]. De forma semelhante, numa rede de Hopfield binária o peso da ligação entre dois neurónios (note-se que aqui os pesos são sempre simétricos) é incrementado sempre que, para um padrão, o estado final desses dois neurónios é idêntico.

Se a função de activação dos neurónios puder assumir os valores 0 ou 1, então os pesos das ligações serão determinados pela equação 3.9.

$$\begin{aligned} w_{ji} &= \sum_{p=1}^{m} (2a_{pi} - 1)(2a_{pj} - 1) \qquad i \neq j \\ w_{ii} &= 0 \end{aligned} \tag{3.9}$$

Note-se que os pesos das ligações são precisamente iguais aos que são determinados pela equação 3.8, uma vez que a expressão $2a - 1$ estabelece a conversão entre as duas representações das saídas.

[3]A formulação matemática desta regra traduz-se na expressão $\Delta w_{kj}(n) = \eta y_k(n) x_j(n)$, onde η é uma constante conhecida por velocidade de aprendizagem. $\Delta w_{kj}(n)$ é a variação no peso da ligação do neurónio j para o neurónio k.

Compreendida esta transformação é simples verificar que a equação 3.9 se reduz à equação 3.8.

Quando as entradas ponderadas do neurónio forem precisamente iguais a $0,5$ para o par de saídas $\{0,1\}$ (0 para saídas $\{-1,1\}$), i.e., $\sum_{i=1}^{N} w_{ji}x_i = 0,5$, então assumimos que o neurónio se mantém no seu estado anterior. Neste sentido podemos dizer que o neurónio tem memória.

Dissemos já que as redes de Hopfield binárias convergem para um estado final definitivo, mas não vimos porquê. De seguida, vamos apresentar uma prova que, não sendo formal é, pelo menos, intuitiva e que se baseia nas funções de energia de Liapunov (mais adiante retomaremos este assunto de uma forma mais exaustiva).

Podemos considerar como função de energia da rede de Hopfield binária a equação 3.10.

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ji}x_i x_j - \sum_{j=1}^{N} I_j x_j \tag{3.10}$$

Agora, se considerarmos que $I_j = 0$, i.e., que não é imposto qualquer viés a partir do exterior, ficamos com:

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ji}x_i x_j \tag{3.11}$$

Se definirmos, $E_i = \sum_{j=1}^{N} w_{ji}x_j$, então, a função da energia pode ser representada pela equação 3.12.

$$E = -\frac{1}{2}\sum_{i=1}^{N} x_i E_i \tag{3.12}$$

Se não houver variação da saída do neurónio k, então também não há variação da energia. Se há uma variação na saída do neurónio k, só se pode dar uma de duas hipóteses: ou x_k subiu ou x_k desceu, respectivamente, $\Delta x_k > 0$ ou $\Delta x_k < 0$. Resulta da própria dinâmica do neurónio, que estas alterações na saída só são possíveis, em

função do sinal de E_k, de acordo com a seguinte correspondência:

$$\Delta x_k > 0 \Rightarrow E_k > 0$$
$$\Delta x_k < 0 \Rightarrow E_k < 0$$

Ou seja, Δx_k e E_k têm forçosamente o mesmo sinal. Em termos de variação da energia total da rede, que resulta da alteração da saída do neurónio k, temos:

$$\begin{aligned} \Delta E &= -\frac{1}{2}\sum_{\substack{i=1 \\ i\neq k}}^{N} x_{i_f} E_i - \frac{1}{2} x_{k_f} E_k - \left(-\frac{1}{2}\sum_{\substack{i=1 \\ i\neq k}}^{N} x_{i_0} E_i - \frac{1}{2} x_{k_0} E_k \right) \\ &= -\frac{1}{2}\left(\Delta x_k E_k\right) < 0 \end{aligned}$$

O índice f significa final e o 0 inicial. Desta expressão resulta que a energia total da rede decresce sempre. Como a energia é limitada inferiormente, a rede neuronal converge para um estado final definitivo. Esta demonstração, com ligeiras diferenças, pode ser encontrada em (Dayhoff, 1996).

Não demonstraremos que para os padrões previamente armazenados são atingidos mínimos da função de energia.

As redes de Hopfield binárias apresentam algumas limitações, de entre as quais destacamos as seguintes:

- por vezes são evocados padrões *espúrios* não armazenados na rede;
- as memórias evocadas nem sempre são as que mais se assemelham ao padrão de entrada;
- nem todas as memórias são evocadas com igual ênfase.

O número máximo de padrões, p, que se podem armazenar na rede depende da probabilidade de erro que for admissível na evocação dos vários dígitos binários de um padrão. (Dayhoff, 1996) fixa em cerca de $0,15 \times N$ o número máximo de padrões armazenável numa rede com N neurónios, sem que o desempenho se degrade fortemente. Outros critérios podem ser encontrados em (Haykin, 1994).

Redes de Hopfield contínuas

Quando comparadas com as redes de Hopfield binárias as redes de Hopfield contínuas apresentam algumas diferenças ao nível do modelo dos neurónios, que interessa salientar:

- a dinâmica dos neurónios é diferente, i.e., a relação entre as entradas e saídas do neurónio tem uma expressão diferente da que foi apresentada na equação 3.6, sendo, neste caso, sensivelmente mais complexa, quando comparado com o funcionamento dos neurónios reais.

- a função de activação é diferente, como não poderia deixar de ser, uma vez que as saídas da rede deixam de ser discretas para passar a ser contínuas. Por esta razão não é possível continuar a usar uma função de activação degrau, sendo utilizada uma função de activação sigmóide.

Em termos de aplicações práticas as redes de Hopfield contínuas têm uma utilização completamente diferente. Enquanto as redes discretas servem essencialmente para memórias endereçáveis pelo conteúdo, as redes contínuas servem, normalmente, para resolver problemas de optimização, sendo que alguns destes não podem ser resolvidos algoritmicamente em tempo útil por pertencerem à classe dos problemas *NP-completos*[4]. É de referir que, muitas vezes, as redes neuronais não permitem obter soluções óptimas para problemas de optimização, mas apenas soluções aproximadas. A sua vantagem reside na potencial velocidade com que o fazem, que deverá ser superior à dos algoritmos processados sequencialmente. A grande velocidade com que se obtêm os resultados advém do facto de o processamento ser maciçamente paralelo.

[4]Um problema pertence à classe NP — não deterministicamente polinomial — se não existe um algoritmo que o possa resolver em tempo polinomial, i.e., se não existe um r tal que o algoritmo seja de ordem $O\left(n^r\right)$. No entanto, encontrada uma solução, deve haver um algoritmo capaz de verificar a sua validade em tempo polinomial. Um problema diz-se NP-completo quando qualquer outro problema NP se pode converter nele próprio em tempo polinomial.

Modelo Aditivo do Neurónio

Em relação ao modelo simplificado do neurónio apresentado na equação 3.1 e na figura 3.1 o modelo do neurónio representado na figura 3.8 é sensivelmente mais complexo. Este modelo é em (Haykin, 1994) chamado de *Modelo Aditivo do Neurónio*. Na figura 3.8 o condensador representa os efeitos capacitivos associados

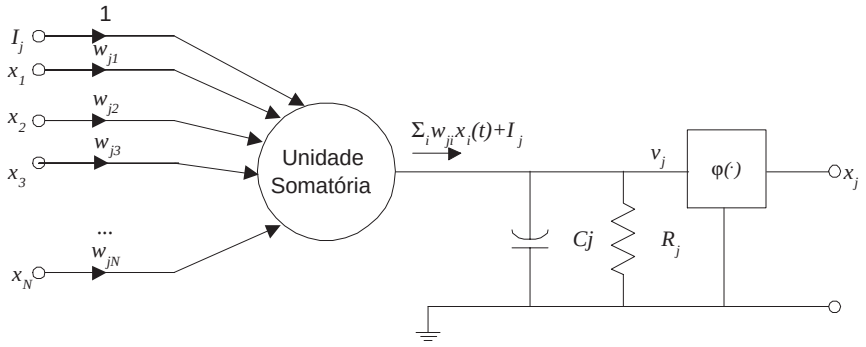


Figura 3.8: Modelo aditivo do neurónio

ao neurónio. Quanto maior for esta capacidade mais lentas serão as alterações do estado do neurónio. Na prática, isto significa que o neurónio tem memória, porque a sua saída não depende apenas das suas entradas, mas também do estado em que se encontrava no instante anterior. Neste aspecto, este modelo representa melhor os neurónios existentes na natureza.

A resistência de fugas, R_j, deve-se ao elemento não linear do neurónio (que produz a função de activação) e que tem resistência de entrada finita. Esta resistência é um elemento dissipativo que contribui para a convergência das redes de Hopfield, uma vez que levam a um decrescimento da energia do sistema.

O parâmetro I_j é o viés imposto a partir do exterior.

O circuito da figura 3.8 é caracterizado pela equação 3.13, onde o nível interno de actividade, v_j, é a corrente à entrada da não-linearidade que produz a função de activação.

$$C_j \frac{dv_j(t)}{dt} + \frac{v_j(t)}{R_j} = \sum_{i=1}^{N} w_{ji} x_i(t) + I_j \tag{3.13}$$

A saída do neurónio que se obtém na saída da não-linearidade é dada por

$$y_j(t) = \varphi(v_j(t))$$

A função de activação utilizada é uma sigmóide dada pela equação 3.5.

A equação 3.13 representa a dinâmica dos neurónios individuais da rede. Se ignorarmos os tempos de propagação do sinal entre neurónios, temos:

$$x_j(t) = \varphi(v_j(t))$$

Não é difícil ver que o comportamento global da rede, como resultado da actividade dos neurónios individuais, pode ser descrito pelo sistema de equações diferenciais não lineares expressos na equação 3.14.

$$\begin{cases} C_1 \frac{dv_1(t)}{dt} = -\frac{v_1(t)}{R_1} + \sum_{i=1}^{N} w_{1i}\varphi(v_i(t)) + I_1 \\ C_2 \frac{dv_2(t)}{dt} = -\frac{v_2(t)}{R_2} + \sum_{i=1}^{N} w_{2i}\varphi(v_i(t)) + I_2 \\ \vdots \\ C_N \frac{dv_N(t)}{dt} = -\frac{v_N(t)}{R_N} + \sum_{i=1}^{N} w_{Ni}\varphi(v_i(t)) + I_N \end{cases} \tag{3.14}$$

O sistema de equações 3.14 permite-nos observar as redes de Hopfield contínuas sob uma perspectiva diferente: estas redes são sistemas descritos por um conjunto de equações diferenciais não lineares. Isto significa também, que podemos simular o funcionamento duma rede de Hopfield, resolvendo este sistema de equações diferenciais. De seguida, vamos fazer uma análise teórica do funcionamento deste sistema.

Note-se que, como não existe um método analítico para resolver este tipo de sistema de equações temos que recorrer a métodos numéricos, nomeadamente ao método de ”Runge-Kutta”, que está descrito no apêndice C.

Função de energia duma rede de Hopfield contínua

Consideremos um sistema físico que possa ser descrito em termos de equações de estado com uma equação como a seguinte:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t) + \mathbf{B}(\mathbf{t}) \cdot \mathbf{u} \tag{3.15}$$

onde $\mathbf{x} = [x_1, x_2, \ldots, x_M]^T$ é o vector de estados do sistema, $\mathbf{f}(\mathbf{x}, t)$ é uma função genérica de $\mathbb{R}^{M+1} \rightarrow \mathbb{R}^M$, onde M é o número de variáveis de estados do sistema (e, portanto, a ordem do sistema), $\mathbf{B}(\mathbf{t})$ é uma matriz $M \times R$ e $\mathbf{u} = [u_1, u_2, \ldots, u_R]^T$ é um vector composto com as R variáveis de entrada do sistema.

Não é difícil verificar que as equações 3.14 são também equações de estado dum sistema físico. $\mathbf{v} = [v_1, v_2, \ldots, v_N]^T$ é o vector de estados do sistema de ordem N e os viés impostos aos neurónios são as entradas do sistema. É de referir ainda, que este sistema não é linear, mas é invariante no tempo — o que em termos da equação 3.15 significa que $\mathbf{f}(\cdot)$ não depende do tempo de uma forma explícita.

A vantagem desta perspectiva reside na possibilidade de usarmos ferramentas matemáticas poderosas provenientes da Teoria dos Sistemas para estudar a estabilidade da rede. Entre estas ferramentas contamos com o segundo método de Liapunov. Para uma introdução mais detalhada ao estudo deste método e teoremas em que se baseia, veja-se (Ogata, 1970). Neste documento limitamo-nos a apresentar alguns resultados teóricos.

Antes de apresentarmos o teorema de Liapunov começamos com algumas definições importantes.

Uma função escalar dependente do tempo $V(\mathbf{x}, t)$, diz-se positiva definida, numa região S do espaço de estados, se:

1. $V(\mathbf{x}) > 0$ para $\mathbf{x} \neq 0$ na região S;
2. $V(\mathbf{x}) = 0$ para $\mathbf{x} = 0$;

Se a função escalar, dependente do tempo, $V(\mathbf{x}, t)$ for positiva em todos os estados da

região S, excepto na origem e em mais alguns estados, em que se anula, então diz-se positiva semidefinida.

Uma função escalar dependente do tempo $V(\mathbf{x}, t)$ diz-se negativa definida ou negativa semidefinida se a sua simétrica, $V(\mathbf{x}, t)$, for positiva definida ou positiva semidefinida, respectivamente.

Seja $\mathbf{x}(t)$ o vector $[x_1(t), x_2(t), \ldots, x_N(t)]^T$. Segundo (Ogata, 1970) uma generalização do teorema de estabilidade de Liapunov diz que, para um sistema descrito pela equação de estado

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t)$$
$$\mathbf{f}(\mathbf{0}, t) = \mathbf{0} \qquad \text{para } t \geq t_0,$$

se existe uma função escalar $V(\mathbf{x}, t)$ com derivadas de primeira ordem parciais contínuas e satisfazendo as seguintes condições:

1. $V(\mathbf{x}, t)$ é positiva definida;
2. $\dot{V}(\mathbf{x}, t)$ é negativa semidefinida;
3. $\dot{V}(\Phi(t; \mathbf{x_0}, t_0), t)$ não é identicamente nula para $t \geq t_0$, qualquer que seja t_0 e qualquer que seja $\mathbf{x_0} \neq \mathbf{0}$, onde $\Phi(t; \mathbf{x_0}, t_0)$ é a trajectória que resolve a equação de estado, começando em $\mathbf{x_0}$ no instante t_0.

então, o estado de equilíbrio na origem do sistema é *globalmente e assimptoticamente estável*. Isto implica que todas as trajectórias do sistema convergem para a origem.

É de referir que para verificar a veracidade do terceiro requisito do teorema não é sempre necessário determinar a trajectória do sistema no espaço de estados. Para isso, seria necessário determinar a solução das equações de estado, $\Phi(\cdot)$, o que à partida é o problema que queremos contornar, por não o sabermos resolver. Nalguns casos simples resulta evidente que esta terceira condição se verifica, mesmo sem conhecermos a

solução das equações de estado. Note-se, também, que só nos interessa estudar o sinal de $V(\mathbf{x},t)$ e $\dot{V}(\mathbf{x},t)$, para a trajectória percorrida pelo sistema.

Uma função escalar $V(\mathbf{x},t)$, que satisfaça estes requisitos diz-se uma função de Liapunov para o estado de equilíbrio, que será sempre a origem - $\mathbf{0}$[5].

À luz deste teorema consideramos para uma rede de Hopfield contínua, a função de energia definida na equação 3.16[6].

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ji}x_i x_j + \sum_{j=1}^{N}\frac{1}{R_j}\int_0^{x_j}\varphi_j^{-1}(x)\,dx - \sum_{j=1}^{N} I_j x_j \tag{3.16}$$

Mais adiante demonstraremos que esta é uma função de Liapunov para o sistema definido pelo sistema de equações 3.14, embora não se demonstre que $\dot{V}(\Phi(t;\mathbf{x_0},t_0),t)$ não seja identicamente nula para alguns pontos $\mathbf{x_0}$ e a partir de um determinado instante de tempo t_0, tal que $t \geq t_0$.

Isto significa que a rede neuronal é assimptoticamente estável, mas possivelmente não será globalmente assimptoticamente estável. Ou seja, a rede converge, independentemente do estado inicial de partida, mas pode haver mais do que um estado atractor. Alguns destes corresponderão a mínimos locais de energia enquanto, possivelmente, apenas um, será o mínimo global. Do estado final é possível extrair o resultado de um cálculo que se espera útil, que pode ser, por exemplo, a solução para um problema de optimização.

Por haver mínimos locais pode não ser possível determinar a solução óptima, mesmo que o problema esteja bem equacionado, mas apenas uma solução aproximada.

Utilizando a expressão da função de energia que acabámos de definir e efectuando algumas manipulações matemáticas podemos obter um resultado importante. Come-

[5]Note-se que este facto não constitui uma limitação, uma vez que para um sistema genérico cujo estado de equilíbrio não seja a origem, podemos aplicar este método à custa duma translação.

[6]Esta expressão assume a forma da equação 3.10, se verificarmos que $\int_0^{x_j}\varphi_j^{-1}(x)\,dx \to 0$, quando a sigmóide considerada nesta equação $\varphi_j(x)$ tende para uma função de activação degrau.

cemos por calcular $\partial E/\partial x_k$:

$$\begin{aligned}
\frac{\partial E}{\partial x_k} = & \\
= & -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\frac{\partial}{\partial x_k}\left(w_{ji}x_i x_j\right) + \sum_{j=1}^{N}\frac{\partial}{\partial x_k}\frac{1}{R_j}\int_0^{x_j}\varphi_j^{-1}(x)\,dx - \sum_{j=1}^{N}\frac{\partial}{\partial x_k}I_j x_j \\
= & -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}w_{ji}\left(\frac{\partial x_i}{\partial x_k}x_j + \frac{\partial x_j}{\partial x_k}x_i\right) + \frac{1}{R_k}\frac{\partial}{\partial x_k}\int_0^{x_k}\varphi_k^{-1}(x)\,dx - I_k\frac{\partial}{\partial x_k}x_k \\
= & -\frac{1}{2}\left(\sum_{j=1}^{N}w_{jk}x_j + \sum_{i=1}^{N}w_{ki}x_i\right) + \frac{1}{R_k}\varphi_k^{-1}(x_k) - I_k
\end{aligned}$$

Note-se que $\partial x_i/\partial x_k = \delta_{ik}$.

Agora, como os pesos das ligações são iguais nos dois sentidos, i.e., $w_{jk} = w_{kj}$, por definição de rede de Hopfield, alterando o índice do somatório de j para i e constatando que, também por definição, $\varphi_k^{-1}(x_k) = v_k$, temos:

$$\frac{\partial E}{\partial x_k} = -\sum_{i=1}^{N}w_{ki}x_i + \frac{v_k}{R_k} - I_k \tag{3.17}$$

Comparando as equações 3.17 e 3.13 e fazendo uma troca de índices, verificamos que $\partial E/\partial x_i$ e dv_i/dt são simétricos a menos de uma constante. Resulta portanto a relação expressa na equação 3.18, onde k é uma constante. Esta equação é conhecida por equação do movimento do neurónio i ("motion equation", em inglês).

$$\frac{dv_i}{dt} = -k\frac{\partial E\left(x_1, x_2, ..., x_N\right)}{\partial x_i} \tag{3.18}$$

Basicamente, esta relação diz que o nível interno de actividade dum neurónio, v_i, tende a opor-se à variação de energia provocada pelo próprio neurónio. Quando esta se anular, terá sido atingido um mínimo de energia e a rede terá convergido.

Em termos intuitivos, se notarmos que $\partial E\left(x_1, x_2, ..., x_N\right)/\partial x_i$ é a componente i do vector gradiente da função energia e sabendo que este vector em cada ponto aponta sempre no sentido de crescimento máximo da função, temos que a variação do nível interno de actividade tenta seguir no sentido exactamente oposto, que será o de decres-

cimento máximo. Em cada instante, portanto, a alteração do estado dos neurónios, que lhes é imposta pela sua própria dinâmica, tende a reduzir a energia total do sistema em direcção ao seu mínimo.

Neste ponto, estamos em condições de demonstrar que a função de energia 3.16 é uma função de Liapunov para o sistema de equações 3.14. Comecemos por calcular dE/dt:

$$\frac{dE}{dt} = \sum_{i=1}^{N} \frac{\partial E}{\partial x_i} \frac{dx_i}{dv_i} \frac{dv_i}{dt} \tag{3.19}$$

Como $dx_i/dv_i = \varphi'(v_i) > 0$, $k > 0$ e pela equação 3.18, temos o resultado da equação 3.20, como queríamos demonstrar.

$$\frac{dE}{dt} = -k \sum_{i=1}^{N} \left(\frac{\partial E}{\partial x_i} \right)^2 \varphi'(v_i) \leq 0 \tag{3.20}$$

Também é vulgar encontrar-se definida a função de energia 3.21 para as redes de Hopfield.

$$E = -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} w_{ji} x_i x_j - \sum_{j=1}^{N} I_j x_j \tag{3.21}$$

Neste caso, não se aplica o resultado da equação 3.18, mas antes o da equação 3.22, onde τ é uma constante. De qualquer forma não é difícil ver que o resultado da equação 3.20 não é válido neste caso e que esta não é um função de Liapunov, para o sistema descrito pelas equações 3.14.

$$\frac{dv_i}{dt} = -\frac{v_i}{\tau} - \frac{\partial E}{\partial x_i} \tag{3.22}$$

Neste ponto, já vimos que a rede converge, independentemente dos pesos que forem seleccionados (desde que sejam simétricos). A questão que se levanta é a de, quando se pretende resolver um determinado problema, encontrar um método para determinar os pesos das ligações entre os neurónios, de forma a construir uma rede que permita realizar o cálculo desejado. À falta de um método rigoroso de o fazer são

utilizados, normalmente, métodos heurísticos.

Um dos problemas NP-completos mais conhecidos que existe é o problema do caixeiro viajante — TSP, do inglês *Traveling Salesman Problem*. De seguida vamos apresentar não só o problema, como também um método para encontrar uma solução, que recorre a redes de Hopfield contínuas. Na concepção desta rede vamos ter oportunidade de estudar o processo pelo qual são determinados os pesos das ligações entre os neurónios.

Note-se que por ser um problema NP-completo, o TSP, não é solúvel em tempo polinomial por um algoritmo a ser executado numa máquina de Touring.

3.1.4 Problema do Caixeiro Viajante — TSP

Enunciado

O Problema do Caixeiro Viajante é um problema clássico de optimização, que pode ser enunciado da seguinte forma:

Um vendedor tem um conjunto de cidades para visitar. Começando e terminando na mesma cidade, pretende saber qual é o percurso mais curto sem passar duas vezes no mesmo sítio.

Vamos apresentar um exemplo, que permita ilustrar melhor o problema, primeiro, e que permita compreender melhor a sua dificuldade, depois.

Suponhamos que o vendedor tem de visitar as cidades de Coimbra, Porto, Lisboa e Évora e que o seu percurso se inicia e termina em Coimbra, sendo a ordem de visita às outras cidades irrelevante. Pretende-se determinar o percurso mais curto.

Na figura 3.9 está representada uma rede onde estão incluídas todas as cidades e respectivas distâncias (em linha recta). As distâncias estão repetidas na tabela 3.1.

Havendo apenas quatro cidades, podemos enumerar sem dificuldade todos os seis itinerários possíveis. A tabela 3.2 apresenta esses itinerários:

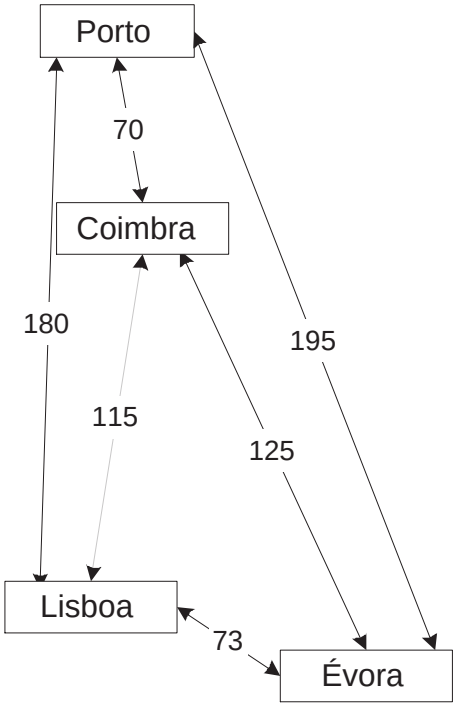


Figura 3.9: Cidades a visitar

Distâncias

Coimbra — Porto:	70 km
Coimbra — Évora:	125 km
Coimbra — Lisboa:	115 km
Porto — Lisboa:	180 km
Porto — Évora:	195 km
Lisboa — Évora:	73 km

Tabela 3.1: Distâncias em linha recta entre as cidades

Neste caso, não é difícil ver que o percurso mais curto seria Coimbra - Porto - Lisboa - Évora - Coimbra ou o percurso exactamente inverso.

Resulta também dum estudo atento à tabela 3.2, que só há três itinerários com custos distintos, uma vez que os outros três — os percursos inversos — são obviamente idênticos em termos de distâncias[7].

Admitimos, sem demonstrar — o que, de resto, não é complicado —, que para n cidades o número total de itinerários é dado por:

[7]Esta conclusão só é válida porque a distância de A para B é a mesma que de B para A. Nalguns tipos de redes não existe a garantia de que as distâncias ou, dito de uma forma mais genérica, os custos, sejam simétricos. Um exemplo deste caso seria este mesmo problema, mas com locais dentro da mesma cidade onde há ruas de sentido único.

Origem	1^a etapa	2^a etapa	3^a etapa	Destino	Total (km)
Coimbra	Porto	Évora	Lisboa	Coimbra	453
Coimbra	Porto	Lisboa	Évora	Coimbra	448
Coimbra	Évora	Porto	Lisboa	Coimbra	615
Coimbra	Évora	Lisboa	Porto	Coimbra	448
Coimbra	Lisboa	Porto	Évora	Coimbra	615
Coimbra	Lisboa	Évora	Porto	Coimbra	453

Tabela 3.2: Itinerários possíveis

$$N_{itinerarios} = \frac{(n-1) \times (n-2) \times \ldots \times 2 \times 1}{2} = \frac{(n-1)!}{2}$$

Partindo deste resultado, construímos a tabela 3.3, onde é estabelecida uma comparação entre o crescimento do número de itinerários e o crescimento duma exponencial. Para um conjunto reduzido de cidades, o número de itinerários mantém-se

N^o de cidades (n)	N^o de itinerários: $(n-1)!/2$	10^n
4	3	1.000
5	12	100.000
8	2.520	100.000.000
10	181.440	10.000.000.000
13	239.500.800	10.000.000.000.000
20	6,082E16	1E20
30	4,42E30	1E30
50	3,04E62	1E50
70	8,556E97	1E70

Tabela 3.3: Comparação entre o número de itinerários e o crescimento duma exponencial

razoavelmente pequeno, mas vemos que o seu crescimento ultrapassa o de uma função exponencial e rapidamente o problema se torna computacionalmente intratável[8].

Concluímos desta introdução que, apesar de se enunciar facilmente, o problema do caixeiro viajante não é tratável computacionalmente senão para os casos mais simples,

[8] A título de curiosidade refira-se que um algoritmo que comparasse as distâncias, percurso a percurso, e que fosse executado num computador que levasse 1ns (tempo muito para além das capacidades de um vulgar computador de secretária actual) a analisar cada um desses percursos, levaria quase 2 anos a encontrar a solução óptima com 20 cidades. Se em vez de 20 cidades fossem 21 o computador levaria perto de 40 anos!

uma vez que o tempo de computação aumenta exponencialmente com o número de cidades. De facto, o problema do caixeiro viajante integra-se na classe dos problemas *NP-completos*.

Método de resolução do problema

Hopfield e Tank inventaram um método capaz de encontrar soluções aceitáveis numa quantidade de tempo razoável para o problema do caixeiro viajante (veja-se (Hopfield & Tank, 1986)).

A ideia chave baseia-se numa Rede de Hopfield contínua, com $N \times N$ neurónios dispostos numa matriz quadrada de ordem N — onde N é o número de cidades a visitar: uma linha por cidade, uma coluna por visita.

Se o cálculo for bem sucedido, a rede deverá convergir para um resultado final em que na matriz existe uma, e apenas uma, unidade activada em cada uma das linhas e em cada uma das colunas, tendo as restantes unidades uma saída nula ou quase nula. A tabela 3.4 apresenta um resultado possível para o exemplo que temos vindo a apresentar. Esta solução corresponderia a visitar Évora em primeiro lugar, Coimbra em

	1^a	2^a	3^a	4^a
Coimbra		•		
Porto			•	
Évora	•			
Lisboa				•

Tabela 3.4: Matriz de neurónios com a solução do problema do caixeiro viajante

segundo, depois o Porto, Lisboa e finalmente voltar a Évora. Note-se, em primeiro lugar, que havia outras matrizes com soluções igualmente válidas para o problema (todas as que correspondem a uma rotação das colunas desta matriz, mais as que correspondem aos percursos inversos). Note-se, também, que não há nenhuma incoerência entre qualquer uma destas soluções e aquela que apresentámos atrás.

A questão que nos propomos abordar é a de como calcular os pesos da rede, de forma a conseguir que a matriz convirja para um estado final semelhante ao da tabela 3.4, que corresponde a uma solução do problema.

O método usado para determinar os pesos das ligações baseia-se em princípios heurísticos. Neste caso, os pesos são calculados como sendo a soma de quatro parcelas que traduzem, em termos matemáticos, um conjunto de pressupostos intuitivos, que são enumerados de seguida:

- é de admitir que a activação dum dos neurónios imponha uma forte inibição em todos os neurónios da mesma linha e da mesma coluna — daqui surgirão dois termos no cálculo dos pesos;
- é também de admitir que deverá haver um termo que entre em conta com as distâncias entre as cidades (esta distância só é contabilizada para neurónios em colunas consecutivas, i.e., para cidades que sejam consecutivas no percurso seleccionado). Como será de esperar, entre neurónios que representam cidades mais distantes haverá inibições maiores;
- além destes, há ainda um termo, que representa a inibição global que a activação duma unidade provoca nas outras.

A expressão seguinte obedece a este conjunto de pressupostos. Os índices devem ser considerados como afectados da operação módulo N, por causa das condições de fronteira.

$$
\begin{array}{ll}
w_{Xi,Yj} = & \\
 & \begin{array}{ll}
-A\delta_{XY}\left(1-\delta_{ij}\right) & \text{ligações inibitórias na mesma linha} \\
-B\delta_{ij}\left(1-\delta_{XY}\right) & \text{ligações inibitórias na mesma coluna} \\
-C & \text{inibição global} \\
-Dd_{XY}\left(\delta_{j,i+1}+\delta_{j,i-1}\right) & \text{termo relativo às distâncias}
\end{array}
\end{array}
$$

onde A, B, C e D são constantes positivas que podem ser ajustadas para afinar o funcionamento da rede;

$w_{Xi,Yj}$ é o peso da ligação do neurónio na linha Y, coluna j para o neurónio na linha X, coluna i. d_{XY} é a distância entre as cidades X e Y.

O viés externo dos neurónios é $I_{Xi} = C \times N$. A função δ_{ij} estjá definida na equação 3.23.

$$\delta_{ij} = \begin{cases} 1 & \text{se } i = j \\ 0 & \text{se } i \neq j \end{cases} \tag{3.23}$$

A esta definição dos pesos, corresponde a função de energia expressa na equação 3.24, a menos da adição de uma constante. Na função de energia não está incluída a parcela do integral da equação 3.16.

$$\begin{aligned} E = & \frac{A}{2} \sum_X \sum_i \sum_{\substack{j \\ j \neq i}} x_{Xi} x_{Xj} + \\ & \frac{B}{2} \sum_i \sum_X \sum_{\substack{Y \\ Y \neq X}} x_{Xi} x_{Yi} + \\ & \frac{C}{2} \left(\sum_X \sum_i x_{Xi} - n \right)^2 + \\ & \frac{D}{2} \sum_X \sum_{\substack{Y \\ Y \neq X}} \sum_i d_{XY} x_{Xi} \left(x_{Y,i+1} + x_{Y,i-1} \right) \end{aligned} \tag{3.24}$$

Os mínimos desta função serão soluções, eventualmente não óptimas, para o problema do caixeiro viajante.

Em termos de resultados, até cerca de 30 cidades, esta rede apresenta soluções que, apesar de não serem óptimas, são consideradas razoáveis quando comparadas com soluções obtidas com o melhor método disponível, conhecido por Lin-Kernighan.

Podemos ainda referir que, se por um lado, a solução proposta por Hopfield e Tank para o problema do caixeiro viajante parece ultrapassar, pelo menos ao nível teórico, as dificuldades computacionais que se deparam aos algoritmos de optimização tradicionais, por outro, a sua utilização prática revela muitas dificuldades, mesmo quando o número de cidades é inferior a 30.

3.2 Problema do CMC com Redes Neuronais

3.2.1 Especificação do Problema

As soluções com RN, que vamos apresentar de seguida propõem-se resolver uma classe de problemas sensivelmente mais simples [9] do que as que temos vindo a considerar.

Resumidamente, nas redes consideradas em (Rauch & Winarske, 1988) e (Ali & Kamoun, 1993), admite-se que as fontes de tráfego, os destinos e os respectivos débitos são conhecidos *a priori*.

Note-se que esta hipótese não é válida nos problemas que temos vindo a tratar. É verdade que é possível obrigar os fluxos a declararem o tráfego que vão gerar, mas não é, em geral, possível prever os fluxos que vão ou não aparecer. O mais que se poderá tentar é fazer uma previsão, mas isso constitui um problema que será provavelmente tão difícil como o que nos propúnhamos resolver à partida (fazer o encaminhamento com QoS, não conhecendo o padrão de tráfego gerado).

Se, porém, a informação que mencionámos (fontes, destino e débito) for conhecida, é possível tentar minimizar o tempo médio de atraso global da rede, sendo o problema conhecido por “quasi-static bifurcated routing problem”.

O tempo médio de atraso de extremo a extremo na rede pode ser calculado pela equação 3.25, se forem assumidas as seguintes condições:

- tráfego gerado por processos de Poisson;
- comprimento dos pacotes distribuído exponencialmente;
- hipótese de independência do comprimento dos pacotes de Kleinrock (Kleinrock, 1964);
- tempo de propagação nulo entre nós.

[9] Estes problemas podem-se considerar mais simples, essencialmente do ponto de vista do tratamento das exigências de QoS que se propõem fazer.

Veja-se sobre este tema (Scharwtz, 1987).

$$E(T) = \frac{1}{\gamma} \sum_{l=1}^{L} \frac{f_l}{C_l - f_l} \tag{3.25}$$

Na equação, f_l é o fluxo médio que atravessa a ligação física l, C_l é a capacidade dessa ligação, L é o número total de ligações físicas da rede e γ é a carga média externa oferecida à rede. f_l, C_l e γ medem-se em unidades de dados por unidades de tempo (*dígitos binários por segundo*, por exemplo).

Como o *quasi-static bifurcated routing problem* tenta minimizar o tempo médio de atraso global da rede, então, o objectivo é minimizar a equação 3.25, sujeita, no entanto, a duas restrições: todo o tráfego oferecido pelo exterior tem que ser de facto encaminhado; e a capacidade das ligações físicas existentes não pode ser ultrapassada pelos fluxos atribuídos. Esta segunda restrição não tem que ser explicitamente introduzida na formalização do problema, porque o tempo de atraso tende para infinito quando, qualquer que seja a ligação física l, f_l se aproxima de C_l. Portanto, desde que exista uma solução, esta restrição nunca é violada sob pena da função objectivo (que mede o atraso) tender para infinito.

Intuitivamente, e fazendo uma análise qualitativa desta equação, vemos que os fluxos devem ser tão divididos quanto possível e atribuídos sempre a ligações físicas com a maior capacidade possível (f_l no numerador, por um lado, e a subtrair C_l no denominador, por outro).

Uma apresentação formal para este problema, que é obviamente não linear, bem como a enumeração de alguns dos métodos existentes para o resolver, podem ser encontrados em (Ali & Kamoun, 1993). Interessa-nos apenas saber que um dos métodos propostos [10] necessita de calcular múltiplas vezes os CMC para todos os fluxos existentes. Este facto representa um esforço de cálculo considerável, razão pela qual, uma RN se apresenta como uma alternativa para o efeito, por questões de desempenho.

É na concepção das RN capazes de calcular CMC que nos vamos debruçar neste documento.

[10] Método do "desvio de fluxo" (FD — *Flow Deviation*).

3.2.2 Solução para o Problema da afectação de fluxos de Rauch e Winarske

Na solução de Rauch e Winarske, a RN procura encontrar a distribuição dum fluxo, que lhe permita minimizar uma equação semelhante à 3.25. Note-se que pode inclusivamente acontecer que o fluxo seja dividido por mais do que um percurso.

O problema a resolver é diferente do que se encontra noutras soluções com RN posteriores, nomeadamente a de Ali e Kamoun, que teremos oportunidade de apresentar na subsecção 3.2.4. Na solução apresentada por estes últimos autores a RN é usada para calcular vários CMC, sendo esses cálculos apenas uma parte do método completo, que também visa minimizar a equação 3.25.

A primeira limitação do método de Rauch e Winarske surge na configuração da RN: os neurónios são dispostos numa matriz em que a cada linha corresponde um nó da rede e a cada coluna corresponde um salto no percurso. Por exemplo, numa rede de dados com 20 nós e em que fossem necessários 4 saltos para chegar da origem ao destino (i.e., o percurso atravessasse 5 nós), seria usada uma matriz com $20 \times 5 = 100$ neurónios.

A limitação reside precisamente no facto de ser necessário conhecer previamente o número de saltos usados no percurso, o que geralmente não faz muito sentido. Para determinar este número é possível utilizar um método, que passamos a descrever.

Comecemos por definir a matriz de custos C, quadrada de ordem N, onde N é o número de nós da rede e onde cada um dos seus elementos, $c_{ij}, i, j = 1, \ldots, N, i \neq j$, representa a capacidade da ligação entre o nó i e o nó j. À diagonal principal da matriz (que representa ligações de cada um dos nós para eles próprios), bem como a todas as ligações que não existam correspondem capacidades nulas.

Se o elemento C_{sd}, onde s é a origem e d o destino existir, i.e., se $C_{sd} \neq 0$, então, há uma ligação directa de s para d. Se $(C \times C)_{sd}$ existir, onde $(C \times C)$ 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 mínimo de saltos para atingir um destino mas, como será evidente, o caminho com menor número de saltos não é necessariamente o melhor caminho. Esta é, na realidade, uma limitação inerente à arquitectura da própria RN. Admitido este facto, utilizamos este método para determinar o número de colunas necessárias na matriz de neurónios.

Consideremos o caso duma rede com 16 nós (numerados sequencialmente) em que a origem dum percurso é o nó 1, e o destino o nó 13, e em que são necessários, pelo menos, 4 saltos para atingir o destino. Neste caso, a matriz de neurónios vai ser constituída por 16 linhas e 5 colunas, num total de $16 \times 5 = 80$ neurónios.

Na tabela 3.5 estão representados os níveis internos de actividade de cada um dos neurónios que compõem a RN no instante inicial. Estes valores constituem um exemplo e dependem da topologia da rede de dados em questão. Em (Rauch & Winarske, 1988) está descrita a forma de inicializar esta matriz. O elemento u_{ij} da matriz representa a percentagem do tráfego que é encaminhada pelo nó i no $(j-1)$-ésimo salto 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 encaminhado. Na primeira coluna o neurónio que corresponde ao nó de origem é fixado a 1, enquanto que na última coluna é fixado a 1 o neurónio que corresponde ao nó de destino. Para conseguir a convergência da RN é definida uma "função de perdas", J, com um significado semelhante à função de energia das redes de Hopfield, definidas na subsecção 3.1.3, a partir da página 75. A função J, representada na equação 3.26 define-se à custa de duas parcelas J_1 e J_2, representadas, respectivamente, nas equações 3.27 e 3.28. Nestas equações $\mathbf{U}_k$ representa a coluna k da matriz e u_{ik} representa o elemento i da coluna k.

$$J = \frac{1}{2}\sum_{k=1}^{4} \mathbf{U}_k^T W \mathbf{U}_{k+1} + \frac{\beta}{2}\sum_{k=2}^{4}\left(\sum_i u_{ik} - 1\right)^2 \tag{3.26}$$

$$J_1 = \frac{1}{2}\sum_{k=1}^{4} \mathbf{U}_k^T W \mathbf{U}_{k+1} \tag{3.27}$$

Elemento	u_1	u_2	u_3	u_4	u_5
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 3.5: Níveis de actividade iniciais, para uma matriz de 16×5 neurónios

$$J_2 = \frac{\beta}{2} \sum_{k=2}^{4} \left(\sum_i u_{ik} - 1 \right)^2 \tag{3.28}$$

Na equação 3.26, J_1, está representada em forma matricial; W representa a matriz de pesos entre os neurónios (que não são mais do que custos).

A forma de calcular os elementos da matriz de pesos está representada na equação 3.29, onde A_{ij} é o tráfego na ligação do nó i para o nó j e C_{ij} é a capacidade da mesma ligação.

$$\begin{aligned} F_1\left(A_{ij}, C_{ij}\right) &= \frac{\rho}{(1-\rho)} \\ \rho &= \left(\frac{A_{ij}}{C_{ij}}\right) \end{aligned} \tag{3.29}$$

Note-se que os termos representados em 3.29 são as parcelas utilizadas no somatório da expressão do cálculo do atraso global (equação 3.25). Esta expressão apresenta um problema, que se prende com o custo F_1, que é difícil de usar para fins computacionais, uma vez que o seu crescimento é ilimitado, quando $\rho \to 1$. Por este motivo, é adoptada uma expressão diferente, que se encontra representada na equação 3.30, 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}).

$$F_3\left(A_{ij}, C_{ij}\right) = F_0 + \rho^2 \tag{3.30}$$

Como facilmente se verifica por observação desta equação, w_{ij} depende do tráfego consignado à ligação, razão pela qual os pesos da RN (i.e., a matriz W), a partir dos quais é determinado precisamente esse tráfego, devem ser reajustados ao fim dum determinado número de iterações da RN (ver (Rauch & Winarske, 1988)). Em termos de concretização em *hardware* esta poderia ser uma questão difícil de resolver, mas que não levanta problemas de maior, enquanto a avaliação da solução se ficar pela 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 3.29 e 3.30, o custo será tanto mais baixo quanto menor for a ocupação e o tempo de transmissão das ligações, o que significa que deverão ser seleccionadas as ligações mais curtas e menos saturadas.

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

A variação na função de perdas J, produzida pela variação apenas numa das colunas k da matriz de activação U será dada pela equação 3.31. $\mathbf{e_i}$ é um vector com todos seus elementos iguais a 1.

$$\begin{aligned}\Delta J =& \Delta \mathbf{U}_k^T \left(\frac{1}{2}\right) [W\mathbf{U}_{k-1} + W\mathbf{U}_{k+1} + \\ & 2\beta \mathbf{e_i} \left(\sum_i u_{ik} - 1\right)\Bigg]\end{aligned} \tag{3.31}$$

Será, porventura mais simples compreender este resultado, se considerarmos $\partial J/\partial u_{im} = 0$, se $m \neq k$ e usarmos a regra da derivação em cadeia: $dJ/dt = \sum_{i=1}^{n} \partial J/\partial u_{ik} \times du_{ik}/dt$.

À semelhança do que foi visto na subsecção 3.1.3, a partir da página 75, o decres-

cimento da função de perdas J é máximo quando a variação de u_k, Δu_k, se opõe à direcção do gradiente da função J, que está expresso entre parêntesis rectos na equação 3.31. Sendo assim, Δu_k é dado pela equação 3.32, embora esta equação esteja escrita em forma vectorial, para toda a coluna $\mathbf{U_k}$.

$$\begin{aligned}\Delta \mathbf{U}_k = &-\alpha\left(\frac{1}{2}\right)\left[W\mathbf{U}_{k-1} + \beta\mathbf{e_i}\left(\sum_i u_{ik} - 1\right)\right] \\ &-\alpha\left(\frac{1}{2}\right)\left[W\mathbf{U}_{k+1} + \beta\mathbf{e_i}\left(\sum_i u_{ik} - 1\right)\right] \\ &\text{para } k = 2,3,4\end{aligned} \tag{3.32}$$

Observando com atenção esta equação, facilmente se verifica que o estado interno de actividade de cada neurónio, depende não só do estado dos seus vizinhos da coluna anterior, mas também dos da coluna seguinte e de todos os neurónios da sua própria coluna, incluindo ele próprio.

3.2.3 Avaliação do Método de Rauch e Winarske

Este é considerado um trabalho pioneiro no que diz respeito à utilização de RN para fazer encaminhamento em redes de dados. Talvez por este motivo, a solução apresentada está afectada de algumas limitações que tivemos ocasião de descrever e que dizem respeito ao facto de o número de saltos do percurso ter que ser determinado de antemão. Outra limitação, que não ocorre noutras soluções, prende-se com o facto de os pesos (bem como a própria disposição) da RN terem que ser configurados de forma única para cada problema. Mesmo na resolução dum único problema, os pesos da RN têm que ser dinamicamente alterados porque dependem da evolução da solução calculada.

De qualquer forma, os autores clamam que a RN converge em 250 iterações para uma rede com 16 nós e para percursos com até 4 ligações físicas.

3.2.4 Solução Proposta para o Problema do CMC por Ali e Kamoun

Ao contrário de outras soluções com RN onde os neurónios representam nós da rede de dados, em (Ali & Kamoun, 1993) é apresentada uma solução onde os neurónios representam ligações físicas.

O modelo proposto para a RN tem $n^2 - n$ neurónios, onde n é o número de nós da rede. Assim, o número máximo de ligações físicas dessa rede será $n(n-1)$, se a conectividade for total, como se impõe numa rede de Hopfield.

Os neurónios da RN são dispostos numa matriz quadrada bidimensional de lado n. Não existem neurónios na diagonal principal, o que totaliza os $n^2 - n$ neurónios da matriz.

O neurónio na linha m, coluna i, representa a ligação do nó m para o nó i, sendo a numeração dos nós feita sequencialmente. Assim, se o neurónio na linha m, coluna i estiver activo ($x_{mi} = 1$), isto significa que a ligação física representada pelo neurónio fará parte do CMC calculado pela RN.

A função de activação dos neurónios está expressa na equação 3.33, onde λ_i é o ganho dos amplificadores que implementam os neurónios.

$$x_i = g_i(v_i) = \frac{1}{1 + e^{-\lambda_i v_i}} \tag{3.33}$$

Função de Energia

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

$$\rho_{mi} = \begin{cases} 1 & \text{se o arco do nó } m \text{ para o nó } i \text{ não existe} \\ 0 & \text{caso contrário} \end{cases}$$

A função de energia para esta rede está representada na equação 3.34, onde os

índices d e s se referem, respectivamente, ao destino e à origem.

$$\begin{aligned} E =& \frac{\mu_1}{2} \sum_{m=1}^{n} \sum_{\substack{i=1 \\ i \neq m \\ (m,i) \neq (d,s)}}^{n} C_{mi} x_{mi} + \frac{\mu_2}{2} \sum_{m=1}^{n} \sum_{\substack{i=1 \\ i \neq m \\ (m,i) \neq (d,s)}}^{n} \rho_{mi} x_{mi} + \\ & \frac{\mu_3}{2} \sum_{m=1}^{n} \left\{ \sum_{\substack{i=1 \\ i \neq m}}^{n} x_{mi} - \sum_{\substack{i=1 \\ i \neq m}}^{n} x_{im} \right\}^2 + \frac{\mu_4}{2} \sum_{m=1}^{n} \sum_{\substack{i=1 \\ i \neq m}}^{n} x_{mi} \left(1 - x_{mi}\right) + \\ & \frac{\mu_5}{2} \left(1 - x_{ds}\right) \end{aligned} \tag{3.34}$$

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

- μ_1: minimiza o custo total do percurso;
- μ_2: elimina ligações inexistentes do percurso;
- μ_3: esta parcela será nula se todos os nós que forem destino de uma ligação activa, forem também origem de outra. Isto assegura que todos os percursos são circulares, sendo fechados por uma ligação forçada do destino para a origem;
- μ_4: força a RN a convergir para um estado válido, que estará sempre num dos cantos do hipercubo definido pelos $x_{mi} \in \{0, 1\}$ [11];
- μ_5: força o percurso a fechar-se num ciclo, incluindo para isso uma ligação do destino para a origem, independentemente dela existir ou não (note-se que mesmo que exista o seu custo não está incluído em μ_2). Esta parcela não faz parte do percurso, mas tem que ser adicionada por causa da parcela μ_3.

Os autores desta RN descrevem ainda um método que permite determinar, de forma alegadamente mais objectiva do que é usual, o valor das constantes $\mu_i, i = 1, \ldots, 5$, que decidimos não abordar aqui.

[11] O número de cantos do hipercubo será de $\underbrace{2 \times 2 \times \ldots \times 2}_{\text{número de neurónios}}$, i.e., 2^{n^2-n}.

O resultado final para a rede da figura 3.10, que serve de exemplo, em termos de activação dos neurónios está representado na figura 3.11, admitindo que o CMC 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 facto de 3 para 1 para fechar o circuito, como tivemos ocasião de dizer.

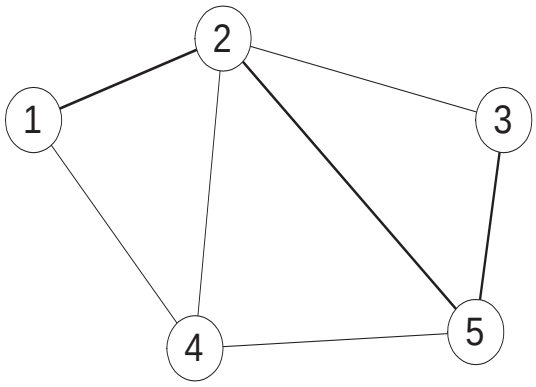


Figura 3.10: Configuração de uma rede e respectivo CMC

Destino

Origem	1	2	3	4	5
1		1	0	0	0
2	0		0	0	1
3	1	0		0	0
4	0	0	0		0
5	0	0	1	0	

Figura 3.11: Activação final da RN

São ainda de realçar dois aspectos. Em primeiro lugar, a existência de ciclos é evitada pela parcela μ_1. Em segundo lugar, não há nenhuma restrição quanto ao número de saltos do percurso, que pode ser qualquer. Neste caso podemos constatar que as linhas e colunas número 4 (quatro) 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.

Determinação dos Pesos

Definindo a energia como na função 3.34, fazendo $\tau_i = R_i C_i$, na equação 3.13 e assumindo que os τ_i são todos iguais, deduzimos as equações 3.35 e 3.36. Note-se que os pesos e os viés podem diferir numa constante em relação aos que são usados na equação 3.13.

$$\frac{dv_{mi}}{dt} = -\frac{v_{mi}}{\tau} + \sum_{z=1}^{n}\sum_{\substack{j=1 \\ j\neq z}}^{n} w_{mi,zj}x_{zj} + I_{mi} \tag{3.35}$$

$$\frac{dv_{mi}}{dt} = -\frac{v_{mi}}{\tau} - \frac{\partial E}{\partial x_{mi}} \tag{3.36}$$

Estas equações são semelhantes à equação 3.22.

A partir destas equações, deduzimos as equações 3.37, 3.38 e 3.39.

$$\begin{aligned}\frac{dv_{mi}}{dt} = &-\frac{v_{mi}}{\tau} - \frac{\mu_1}{2}C_{mi}\left(1-\delta_{md}\delta_{is}\right) - \frac{\mu_2}{2}\rho_{mi}\left(1-\delta_{md}\delta_{is}\right) - \\ &\mu_3\sum_{\substack{z=1 \\ z\neq m}}^{n}\left(x_{mz}-x_{zm}\right) + \mu_3\sum_{\substack{z=1 \\ z\neq i}}^{n}\left(x_{iz}-x_{zi}\right) - \frac{\mu_4}{2}\left(1-2x_{mi}\right) + \frac{\mu_5}{2}\delta_{md}\delta_{is} \\ &\forall\,(m,i)\in \mathbb{N}\times\mathbb{N}, m\neq i\end{aligned} \tag{3.37}$$

$$w_{mi,zj} = \mu_4\delta_{mz}\delta_{ij} - \mu_3\delta_{mz} - \mu_3\delta_{ij} + \mu_3\delta_{jm} + \mu_3\delta_{iz} \tag{3.38}$$

$$\begin{aligned}I_{mi} = &-\frac{\mu_1}{2}C_{mi}\left(1-\delta_{md}\delta_{is}\right) - \frac{\mu_2}{2}\rho_{mi}\left(1-\delta_{md}\delta_{is}\right) - \\ &\frac{\mu_4}{2} + \frac{\mu_5}{2}\delta_{md}\delta_{is} \\ = &\begin{cases} \frac{\mu_5}{2} - \frac{\mu_4}{2} & \text{se } (m,i) = (d,s) \\ -\frac{\mu_1}{2}C_{mi} - \frac{\mu_2}{2}\rho_{mi} - \frac{\mu_4}{2} & \text{caso contrário} \end{cases}\end{aligned} \tag{3.39}$$

Nestas expressões, δ_{ij} é a mesma função delta de Kronecker que já definimos na equação 3.23.

Uma característica interessante desta rede e que poderá ser confirmada por

observação da equação 3.38 é que os pesos das ligações entre neurónios dizem respeito a critérios fixos, que só dependem da posição relativa dos neurónios dentro da RN. A origem, o destino e os custos são todos configurados nos viés, expressos na equação 3.39. No que diz respeito à operação em tempo real, através da concretização em *hardware*, esta solução apresenta a clara vantagem de ser adaptável, porque basta alterar os viés para resolver um problema diferente, mantendo sempre fixos os pesos, que mais dificilmente poderiam ser alterados.

3.2.5 Avaliação do Método de Ali e Kamoun

Pelos resultados apresentados pelos autores, a rede converge para problemas que vão até aos 15 nós (com 20 fluxos de dados diferentes a atravessar a rede). (Park & Choi, 1998) afirmam que este tipo de RN não converge para uma solução válida quando a rede tem mais de 20 nós.

De qualquer forma, a grande vantagem deste algoritmo reside na capacidade que tem de se adaptar a novas situações de cálculo, o que se deve ao facto de a configuração da rede ser introduzida nos viés. Isto permite também configurar a RN de forma a exprimir redes de dados que apresentem ligações que não sejam bidireccionais, o que é essencial em termos de aplicação prática.

3.2.6 Solução Proposta para o Problema do CMC por Park e Choi

Função de Energia

Em (Park & Choi, 1998) é apresentada uma solução que se baseia na ideia de (Ali & Kamoun, 1993). No entanto, a nova solução aplica-se também para destinos *multicast*.

A função de energia dada pela equação 3.40 proposta é, como facilmente se cons-

tata, semelhante à que foi apresentada em 3.34.

$$\begin{aligned} E =& \frac{A}{2}\sum_{m=1}^{n}\sum_{\substack{i=1\\ i\neq m}}^{n} C_{mi}x_{mi} + \frac{B}{2}\sum_{m=1}^{n}\sum_{\substack{i=1\\ i\neq m}}^{n}\rho_{mi}x_{mi} + \\ & \frac{C}{2}\sum_{m=1}^{n}\left\{\sum_{\substack{i=1\\ i\neq m}}^{n} x_{mi} - \sum_{\substack{i=1\\ i\neq m}}^{n} x_{im} - \phi_i\right\}^2 + \frac{D}{2}\sum_{m=1}^{n}\sum_{\substack{i=1\\ i\neq m}}^{n} x_{mi}\left(1 - x_{mi}\right) + \\ & \frac{F}{2}\sum_{m=1}^{n}\sum_{\substack{i=1\\ i\neq m}}^{n} x_{mi}x_{im} \end{aligned} \tag{3.40}$$

A função ϕ_i está definida na equação 3.41 e aparece na função de energia porque, agora, não é incluído no percurso a ligação de retorno, que liga o destino (note-se que pode haver vários destinos) à origem, de forma que o nó de origem não é destino de qualquer ligação, assim como o(s) nó(s) de destino não são origem de nenhuma outra. Para mais detalhes veja-se (Park & Choi, 1998).

$$\phi_i = \begin{cases} 1, & \text{se } i = s \\ -1, & \text{se } i = d\text{, para todos os destinos} \\ 0, & \text{caso contrário} \end{cases} \tag{3.41}$$

Aplicando a este método o mesmo raciocínio que utilizámos no método de Ali e Kamoun, podemos facilmente deduzir os pesos e os viés que devem afectar cada neurónio da rede, representados nas equações 3.42 e 3.43, respectivamente.

$$w_{mi,zj} = D\delta_{mz}\delta_{ij} - F\delta_{mj}\delta_{iz} + C\left(-\delta_{mz} - \delta_{ij} + \delta_{jm} + \delta_{iz}\right) \tag{3.42}$$

$$I_{mi} = -\frac{A}{2}C_{mi} - \frac{B}{2}\rho_{mi} - \frac{D}{2} + C\left(\delta_{ms} - \delta_{is} - \delta_{md} + \delta_{id}\right) \tag{3.43}$$

As constantes A, B, C e D e o significado das respectivas parcelas tem um correspondência directa com as constantes μ_1, μ_2, μ_3 e μ_4 da equação 3.34. Não existe correspondência com a parcela μ_5 desta equação pelo motivo que já referimos, de que o percurso já não tem que ter um retorno do destino para a origem. Em vez disso, é acres-

centada a parcela F que tenta assegurar que $x_{ij} \times x_{ji} = 0, \forall_{i,j}$. A ideia desta parcela é impedir ciclos directos (por exemplo ligação de 3 para 5 e ligação de 5 para 3 activas em simultâneo). Note-se que isto não aconteceria normalmente, porque se a equação de energia estiver bem formulada não poderão haver ciclos. Acontece, porém, que esta parcela, segundo os autores, acelera consideravelmente o processo de convergência da RN.

Método de *Screening*

Uma ideia apresentada em (Park & Choi, 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 do CMC.

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

Comecemos por definir o conceito de custo limiar, $C_{sl}(n)$, na n-ésima iteração do algoritmo como sendo a seguinte (N é o número máximo de iterações, C_{min} é a ligação de custo mínimio e C_{max} a de custo máximo):

$$C_{sl}(n) = C_{min} + AL \times (n-1)$$
$$AL = \frac{1}{n}\left(C_{max} - C_{min}\right) \qquad \text{onde } 2 < n < N-1.$$

Dada esta definição, o algoritmo pode ser descrito da seguinte forma:

- determinar as ligações de custo mínimo e de custo máximo — C_{min} e C_{max}, respectivamente;
- repetir os seguintes passos, até ser possível encontrar um percurso, ou até à $N-1$-ésima iteração:

– calcular o limiar, $C_{sl}(n)$ (crescente e que se inicia em C_{min});

– eliminar todas as ligações com custo superior a $C_{sl}(n)$.

Para determinar se nas redes que vão resultando da avaliação deste método por eliminação de algumas das ligações é possível encontrar um percurso desde a origem até ao destino, é usado outro método chamado "Depth-First Search" (DFS). Veja-se (Weiss, 1993).

3.2.7 Avaliação do Método de Park e Choi

Para a rede proposta no artigo e para os pares {origem, destino} considerados, a RN depois do método de *screening* logrou sempre atingir, não só resultados superiores aos atingidos pela RN de Ali e Kamoun, mas inclusivamente a solução óptima. 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 de *screening* ou das alterações introduzidas na função de energia.

3.3 Conclusão

Como tivemos ocasião de ver neste capítulo, tanto as redes de Ali e Kamoun como de Park e Choi são passíveis de ser utilizadas para fazer encaminhamento. Uma das características que apresentam é que codificam os custos nos viés, pelo que é relativamente simples, em termos de concretização por *hardware*, alterar dinamicamente o valor dos custos e adicionar ou eliminar ligações entre nós.

Esta capacidade é essencial para um algoritmo de encaminhamento numa rede de dados com garantias de QoS, uma vez que os custos a considerar na rede podem variar ao longo do tempo, se forem o resultado da codificação de parâmetros de QoS. No

QoSPF, por exemplo, é a própria configuração da rede que se altera, uma vez que algumas das ligações físicas são eliminadas se não oferecerem largura de banda suficiente.

Como já tivemos ocasião de referir, outra das características essenciais da RN deverá ser o tempo de que esta necessita para convergir para uma solução, uma vez que em função deste tempo e da qualidade das soluções obtidas (que normalmente não serão óptimas) se justificará, ou não, a sua utilização num caso real.

Nesta tese vamos considerar precisamente a aplicação dos dois tipos de RN que mencionámos (Ali e Kamoun; Park e Choi) a redes de dados e vamos simular a sua utilização num ambiente que teremos ocasião de descrever no capítulo 5. Não vamos utilizar o método de *Screening* nas redes de Park e Choi por duas razões: em (Park & Choi, 1998) não é especificado o tempo despendido na sua utilização; além disso, este mesmo método poderia ser aplicado a um grafo antes de se utilizar qualquer outro algoritmo ou dispositivo capaz de calcular um CMC.

Estes dois tipos de RN apresentam, porém, duas limitações importantes, que se prendem com o elevado número de neurónios necessários e com a dificuldade em obter soluções válidas. Por este motivo, propomos uma RN alternativa, que apresentaremos no capítulo seguinte.

4

Redes Neuronais de Variáveis Dependentes

4.1 Introdução

Uma das principais dificuldades quando se utilizam RN de Hopfield para resolver problemas de optimização prende-se com a obtenção de soluções válidas. Este problema ocorre, porque acontece muitas vezes a RN convergir para um ponto que não corresponde a um mínimo global, mas apenas a um mínimo local. Como este mínimo local não corresponde a nenhuma solução válida (mesmo que subóptima), o resultado final é que não se obtém qualquer solução, seja ela óptima ou não.

Outro dos problemas com as soluções de Ali e Kamoun e de Park e Choi é que o número de neurónios da RN cresce com o quadrado do número de nós do grafo. Como os resultados obtidos com uma RN se degradam, à medida que o número de neurónios aumenta, é evidente que incrementos no número de nós do grafo resultam em incrementos mais rápidos no número de neurónios com degradações respectivas nos resultados da RN.

Neste capítulo apresentamos uma solução que, tanto quanto sabemos, é original em vários aspectos e que procura solucionar estes dois problemas. Em primeiro lugar atinge mais facilmente soluções válidas, mesmo que subóptimas, além de não apresentar uma dependência quadrática do número de neurónios, mas apenas linear (mais exactamente, o número de neurónios será igual ao número de arcos do grafo).

Para isso, a arquitectura tradicional monocamada das redes de Hopfield será alterada para conter duas camadas e a função de activação dos neurónios passará a ser um polinómio do terceiro grau em vez duma sigmóide.

4.2 Restrições do Grafo

Consideremos o problema de determinar o CMC num grafo qualquer com *N* nós e *M* arcos que pode representar, por exemplo, uma rede de dados, que é o caso concreto que nos interessa.

Dados a origem e o destino do percurso, existe sempre um sistema de equações, tal como representado na equação 4.1, que representa o conjunto de restrições a que obedecem todas as soluções válidas para o problema do CMC.

$$\mathbf{A}\mathbf{v} = \mathbf{b} \tag{4.1}$$

Nesta equação, $\mathbf{A}$ é uma matriz $(N-1) \times M$, que exprime a configuração do grafo, $\mathbf{b}$ é um vector com $(N-1)$ elementos, que permite especificar a origem e o destino do percurso e $\mathbf{v}$ é um vector com M elementos, onde cada um deles está biunivocamente associado a um dos M arcos do grafo. Cada um dos elementos deste vector pode assumir o valor 1 ou 0, conforme o arco que lhe corresponde participe ou não no percurso seleccionado, respectivamente. Este conjunto de restrições deve-se ao seguinte: comecemos por designar os arcos com destino num determinado nó como sendo arcos de entrada desse nó e os arcos que aí têm a sua origem como arcos de saída; então, em cada nó da rede, o número dos seus arcos de entrada que participam no percurso tem que ser igual ao número de arcos de saída que também participam no percurso. Os dois nós que constituem excepções são a origem e o destino do percurso. No nó de origem o número de arcos de saída tem que exceder em uma unidade o número de arcos de entrada. No nó de destino ocorre o inverso, i.e., o número de arcos de entrada tem que exceder em uma unidade o número de arcos de saída.

Não existem N restrições, como seria inicialmente de pensar, mas apenas $N-1$, porque a N-ésima pode ser representada como uma combinação linear das outras $N-1$.

Note-se que, mesmo que uma solução verifique o conjunto das restrições, isto não significa que seja válida, porque há uma infinidade de soluções para a equação 4.1 (desde que $M > N-1$, o que em geral é verdade). Isto porque uma solução só será

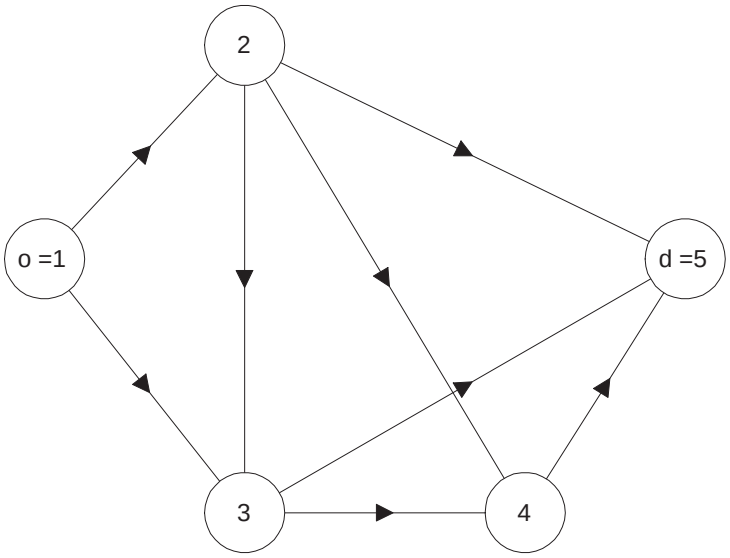


Figura 4.1: Exemplo de Grafo

válida se todos os elementos de v assumirem os valores 0 ou 1.

Para o grafo da figura 4.1, teremos para os nós 1, 3 e 5, as equações 4.2 a 4.4. Nestas equações v_{AB} representa o arco do nó A para o nó B. O nó 1 é a origem do percurso e o nó 5 o seu destino.

$$v_{12} + v_{13} = 1 \tag{4.2}$$

$$v_{13} + v_{23} = v_{34} + v_{35} \tag{4.3}$$

$$v_{25} + v_{35} + v_{45} = 1 \tag{4.4}$$

4.3 Equação de Energia

Para determinarmos o CMC vamos usar uma RN composta por M neurónios, onde cada neurónio representa um dos arcos do grafo. Mais precisamente, o nível de actividade do neurónio representa a participação do arco no percurso — ou seja, as variáveis v_i do vector v da equação 4.1. Portanto, os únicos estados válidos para um neurónio são 0 ou 1. Isto não significa, porém, que a saída do neurónio seja binária — não o será — mas apenas que no estado final de convergência da RN o nível de actividade

de cada neurónio terá de ser 0 ou 1.

Para já, vamos considerar apenas o problema da obtenção duma solução válida. A questão da optimização será abordada mais adiante. Uma solução será válida se o conjunto de restrições expressas na equação 4.1 se verificar. Além disso, é necessário que o nível de actividade de cada um dos neurónios seja 0 ou 1.

4.3.1 Verificação das Restrições

Para garantir que as restrições da equação 4.1 se verificam configuramos a RN de forma a que as restrições sejam satisfeitas pela própria dinâmica da rede. Para isso, em cada uma das $N-1$ equações do sistema 4.1 seleccionamos uma variável diferente. Atribuindo índices sequenciais às variáveis, podemos escolher os índices desde 1 até $N-1$ para as variáveis assim seleccionadas (que designaremos por *dependentes*) e os índices desde N até M às restantes (*independentes*).

Utilizando o método de eliminação de Gauss-Jordan [1] podemos transformar o sistema de restrições 4.1 no sistema 4.5.

$$\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_{N-1} \end{bmatrix} = \begin{bmatrix} \alpha_{1N} & \alpha_{1,N+1} & \cdots & \alpha_{1M} \\ \alpha_{2N} & \alpha_{2,N+1} & \cdots & \alpha_{2M} \\ \cdots & & & \\ \alpha_{N-1,N} & \alpha_{N-1,N+1} & \cdots & \alpha_{N-1,M} \end{bmatrix} \begin{bmatrix} v_N \\ v_{N+1} \\ \vdots \\ v_{M-1} \\ v_M \end{bmatrix} + \begin{bmatrix} o_1 \\ o_2 \\ \vdots \\ o_{N-1} \end{bmatrix} \tag{4.5}$$

Isto significa que haverá dois tipos de variáveis, que por sua vez serão representadas por dois tipos de neurónios, que também designaremos por *dependentes* e *independentes*. A RN resultante será composta por duas camadas: uma primeira de neurónios independentes, cuja actividade pode variar livremente (neurónios N até M), tal como qualquer neurónio numa rede de Hopfield e que procuram minimizar a energia total

[1] Note-se que a utilização do método de Gauss-Jordan pode não ser razoável em todos os problemas. Não o é seguramente no problema do CMC, que tem uma solução algorítmica de ordem inferior $O(MlogN)$. Mais adiante apresentaremos um método mais fácil de conseguir a eliminação das variáveis, que se aplica aos grafos.

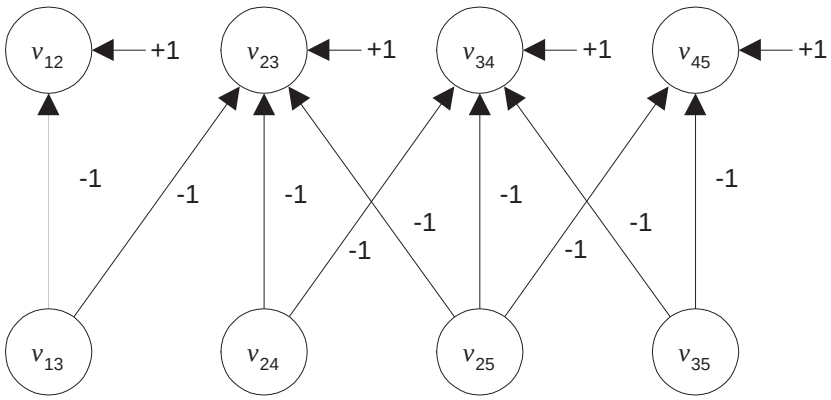


Figura 4.2: Ligações para os neurónios dependentes

da RN; e uma segunda camada, de neurónios (1 até $N-1$) cuja actividade depende linearmente das actividades dos neurónios da primeira camada, de acordo com o que está expresso na equação 4.5.

Com uma RN assim construída não é possível atingir um estado que não satisfaça a equação 4.5 e, por isso, também a equação 4.1, uma vez que estas duas equações são equivalentes. Resta garantir que as saídas dos neurónios serão 0 ou 1.

Tomemos de novo como exemplo a figura 4.1. Como há $N = 5$ nós e $M = 8$ arcos, temos 4 variáveis dependentes e 4 variáveis independentes. Seleccionando para variáveis dependentes v_{12}, v_{23}, v_{34} e v_{45}, obtemos o sistema de equações 4.6. Entre parêntesis encontra-se o número do nó a partir do qual foi obtida a equação.

$$\begin{aligned}
&(1) \quad v_{12} &&= 1 - v_{13} \\
&(2) \quad v_{23} &&= 1 - v_{13} - v_{24} - v_{25} \\
&(3) \quad v_{34} &&= 1 - v_{24} - v_{25} - v_{35} \\
&(4) \quad v_{45} &&= 1 - v_{25} - v_{35}
\end{aligned} \tag{4.6}$$

A RN para este grafo e para esta escolha de variáveis dependentes está representada na figura 4.2. Faltam ainda algumas ligações (as de realimentação), que serão introduzidas mais adiante.

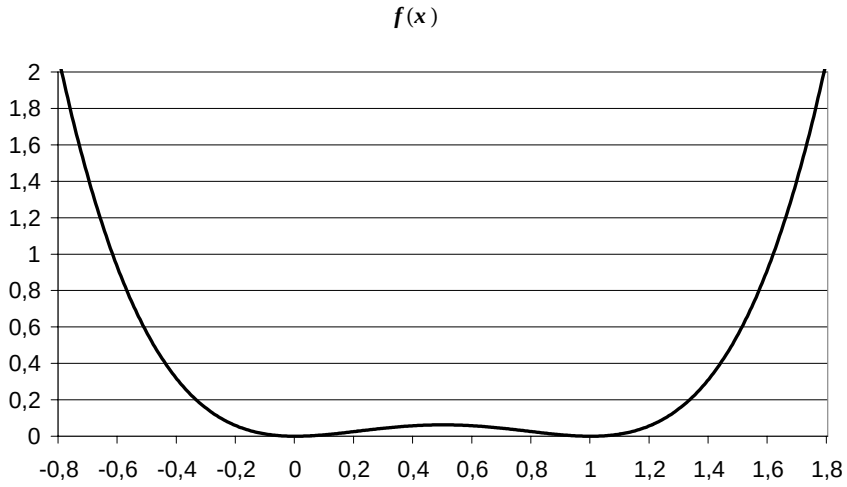


Figura 4.3: Função de energia de cada neurónio

4.3.2 Saídas Binárias

Para forçar os neurónios para o estado final binário 0 ou 1, definimos, para a RN, a função de energia de Liapunov 4.7.

$$E = \sum_{i=1}^{M} \mu_1 f(v_i) \tag{4.7}$$

A função $f(\cdot)$, que designaremos de *função energia* (mas apenas de um neurónio e não de toda a rede) está definida na equação 4.8 e representada na figura 4.3. Anula-se apenas em $x = 0$ e $x = 1$ e é simétrica em relação ao eixo $x = 0.5$.

$$f(x) = (x-0)^2 (x-1)^2 = x^4 - 2x^3 + x^2 \tag{4.8}$$

Para estas definições, o mínimo da função de energia será atingido quando a activação de todos os neurónios for 0 ou 1. Neste caso $E = 0$. Fora disso $E > 0$.

A derivada da função energia, que se encontra calculada na equação 4.9 servirá, como veremos, de função de activação para todos neurónios, não só para os independentes, mas também para os dependentes. A sua representação é feita na figura 4.4. É uma função simétrica em relação ao ponto $(0,5; 0)$.

$$f'(x) = 4x^3 - 6x^2 + 2x \tag{4.9}$$

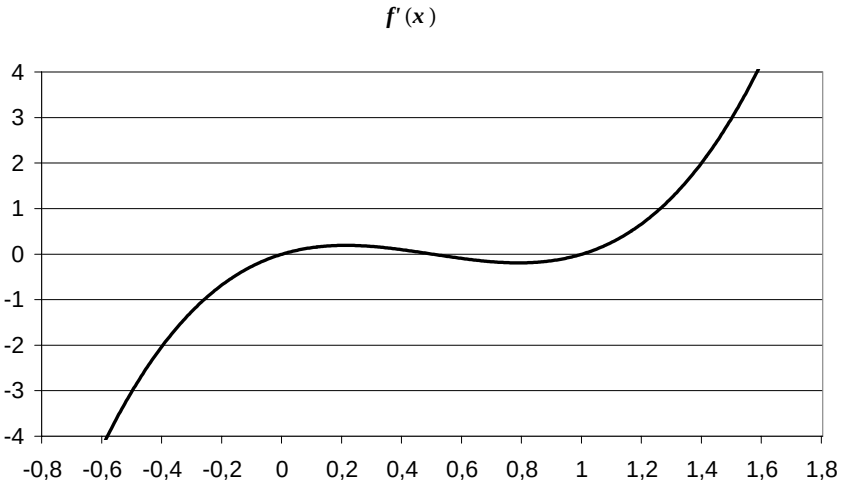


Figura 4.4: Função de activação (derivada da função de energia)

4.4 Equação do Movimento

O problema da convergência das redes de Hopfield (que como sabemos têm realimentação) é já bem conhecido e foi abordado no capítulo 3. No caso desta rede, apesar de haver duas camadas, o estudo é bastante semelhante. Em primeiro lugar, vamos definir a equação 4.10 como sendo a equação do movimento dos neurónios independentes. A equação de energia, E, é a que foi definida em 4.7.

$$\frac{dv_i}{dt} = -\frac{\partial E}{\partial v_i} \tag{4.10}$$

Pela regra da derivação em cadeia e atendendo ao facto de só haver $M-N+1$ variáveis independentes e pela equação 4.10 temos:

$$\frac{dE}{dt} = \sum_{i=N}^{M} \frac{\partial E}{\partial v_i}\frac{dv_i}{dt} = -\sum_{i=N}^{M}\left(\frac{\partial E}{\partial v_i}\right)^2 \leq 0 \tag{4.11}$$

ou seja, a RN nunca ganha energia.

Calculando agora $\partial E/\partial v_i$ e como $\partial v_j/\partial v_i = 0$, para $i, j = N, \ldots, M$ e $i \neq j$, temos:

$$\begin{aligned}\frac{\partial E}{\partial v_i} &= \mu_1 \sum_{j=1}^{M} \frac{\partial f(v_j)}{\partial v_i} \\ &= \mu_1 f'(v_i) + \mu_1 \sum_{j=1}^{N-1} f'(v_j)\frac{\partial v_j}{\partial v_i}\end{aligned}$$

Ora, $\partial v_j/\partial v_i = \alpha_{ji}$, para $j = 1, \ldots, N-1; i = N, \ldots, M$, de acordo com a equação 4.5, pelo que obtemos a equação 4.12.

$$\frac{\partial E}{\partial v_i} = \mu_1 f'(v_i) + \mu_1 \sum_{j=1}^{N-1} \alpha_{ji} f'(v_j) \tag{4.12}$$

Podemos conseguir que os neurónios funcionem de acordo com a equação 4.10, garantindo duas condições. Em primeiro lugar, a variação do nível de actividade do neurónio deve estar de acordo com a equação 4.13, onde os vários símbolos têm o significado introduzido no capítulo 3. Este neurónio é idêntico ao que foi estudado no capítulo 3, equação 3.13, com a diferença de que aqui se elimina o termo resistivo. Relembramos que w_{ik} é o peso do neurónio k para o neurónio i.

$$\frac{dv_i}{dt} = \sum_{k=1}^{M} w_{ik} x_k + I_i \tag{4.13}$$

Em segundo lugar, fazemos:

$$\begin{aligned} x_k &= f'(v_k) \\ w_{ik} &= \begin{cases} -\mu_1 & \text{se } k = i \\ 0 & \text{se } k \neq i, k = N, \ldots, M \\ -\mu_1 \alpha_{ki} & \text{se } k = 1, \ldots, N \end{cases} \\ I_i &= 0 \end{aligned}$$

Nestas condições, e observando as equações 4.12 e 4.13 é imediato constatar que a equação 4.10 se verifica.

Chamamos a atenção para o facto de as entradas de realimentação para um

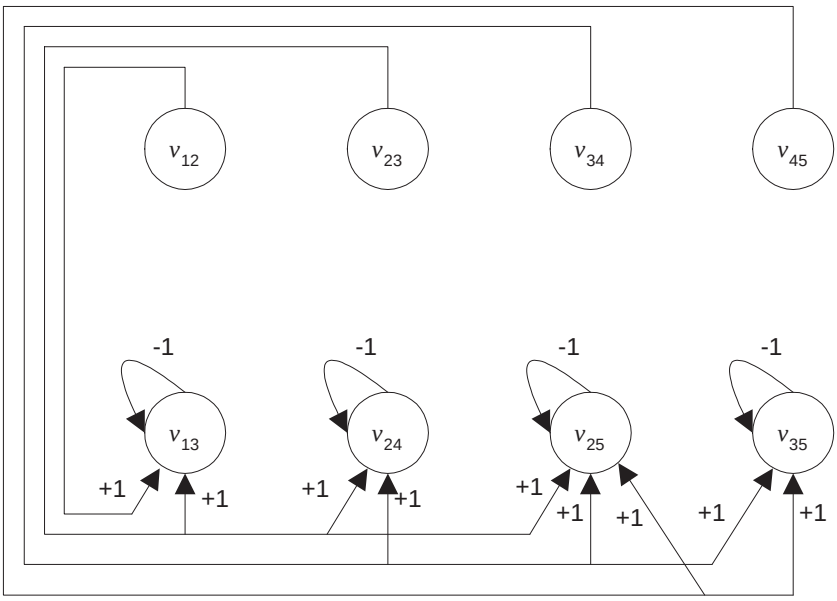


Figura 4.5: Ligações para os neurónios independentes

neurónio independente (x_k na equação 4.13) serem obtidas, não directamente da sua própria saída e das saídas dos neurónios dependentes, mas a partir duma função de activação que afecta todas essas saídas. Esta função de activação tem a forma da função de energia derivada que apresentámos na equação 4.9.

Voltando ao exemplo que temos vindo a apresentar, da figura 4.1, obtemos a RN da figura 4.5, com os pesos dos neurónios dependentes para os independentes incluídos e com auto-realimentação dos neurónios independentes. Assumimos que $\mu_1 = 1$, para simplificar a compreensão da figura.

4.5 Consideração dos Custos

A solução que temos vindo a apresentar não tem em consideração os custos dos arcos do grafo, mas apenas a validade da solução obtida. Para introduzir estes custos na solução, propomos o seguinte método:

- alterar a função de energia, de forma a considerar estes custos;
- iniciar a RN e deixá-la convergir até um resultado final (que não é uma solução válida);

- partindo deste ponto de funcionamento, retirar os custos da função de energia e deixar a rede convergir para uma solução válida, que se espera óptima (ou pelo menos próxima da óptima).

Este método é bastante simples de concretizar porque, como veremos, os custos serão introduzidos nos viés, o que permite à RN considerar ou não os custos, e fazer variar a sua grandeza.

A função de energia expressa em 4.14 difere da função de energia 4.7 apenas na parcela afectada da constante μ_2. C_i designa o custo do arco i.

$$E = \sum_{i=1}^{M} \left[\mu_1 f\left(v_i\right) + \mu_2 C_i v_i\right] \tag{4.14}$$

Aplicando o mesmo raciocínio que já utilizámos antes, chegamos à equação 4.15.

$$\begin{aligned} \frac{\partial E}{\partial v_i} =& \mu_1 f'(v_i) + \mu_1 \sum_{j=1}^{N-1} \alpha_{ji} f'(v_j) + \\ &+ \mu_2 C_i + \mu_2 \sum_{j=1}^{N-1} \alpha_{ji} C_j \end{aligned} \tag{4.15}$$

Comparando este resultado com a equação 4.13 e atendendo novamente a que $dv_i/dt = -\partial E/\partial v_i$, temos o resultado da equação 4.16, para $i = N, \ldots, M$.

$$I_i = -\mu_2 C_i - \mu_2 \sum_{j=1}^{N-1} \alpha_{ji} C_j \tag{4.16}$$

É portanto muito simples considerar, ou não, os custos dos arcos: basta, respectivamente, aplicar ou não os viés assim calculados.

Note-se que os custos das variáveis dependentes são, desta forma, codificados nas variáveis independentes, que são as únicas cujos neurónios podem variar livremente. É de referir que essa codificação reflecte já o facto de a variável independente participar, ou não, e com que peso, no cálculo da variável dependente, como resulta dos coeficientes α_{ji} da equação 4.16. Se a variável independente k não participar no cálculo da variável dependente z, então o custo do arco associado à variável z também não é

considerado no neurónio independente k, uma vez que $\alpha_{zk} = 0$.

Regressando ao exemplo da figura 4.1, teríamos os seguintes viés:

$$I_{13} = -\mu_2 C_{13} - \mu_2 \left(+1 \times C_{12} + 1 \times C_{23}\right)$$
$$I_{24} = -\mu_2 C_{24} - \mu_2 \left(+1 \times C_{23} + 1 \times C_{34}\right)$$
$$I_{25} = -\mu_2 C_{25} - \mu_2 \left(+1 \times C_{34} + 1 \times C_{45}\right)$$
$$I_{35} = -\mu_2 C_{35} - \mu_2 \left(+1 \times C_{34} + 1 \times C_{45}\right)$$

Os resultados experimentais que apresentaremos no capítulo 5 foram obtidos com os seguintes valores para as constantes da função de energia: $\mu_1 = 500$ e $\mu_2 = 80$. Também nesta RN se veio a revelar complicado determinar estas constantes por um método não empírico. Por este motivo, foram experimentadas múltiplas combinações possíveis para os pesos, tendo sido esta a que apresentava um compromisso mais razoável entre os seguintes critérios, que são muitas vezes antagónicos: velocidade de convergência, qualidade das soluções e respectiva validade.

Outro aspecto que se veio a revelar importante diz respeito à possibilidade de haver múltiplos caminhos com custos equivalentes. Muitas vezes estes caminhos deixam a RN num estado em que não se consegue decidir por um deles e em que acaba por encalhar num resultado final inválido ou noutro caminho que está muito longe do óptimo.

Para resolver este problema introduzimos um factor aleatório nos viés, de forma a que estes representem os custos duma forma propositadamente inexacta. O viés aplicado, de facto, aos neurónios independentes é escolhido aleatoriamente no intervalo $[(1 - \alpha/2)\, v, (1 + \alpha/2)\, v]$ à volta do viés exacto, v. O valor de α nas simulações que fizemos nesta trabalho é de 5%. Este procedimento tende a quebrar simetrias, mas naturalmente pode levar a RN a atingir resultados subóptimos.

Na figura 4.6 está representada a totalidade da RN, para o exemplo da figura 4.1, incluindo o viés a apresentar aos neurónios independentes.

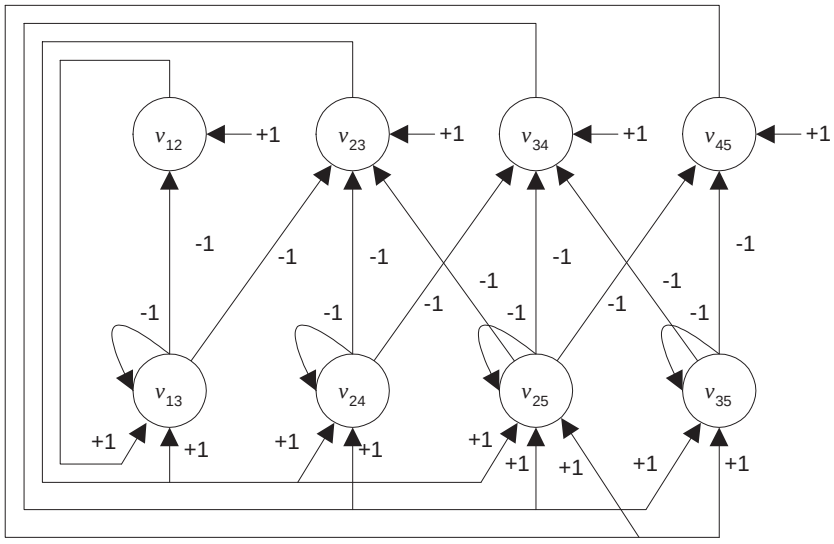


Figura 4.6: Rede Neuronal de Variáveis Dependentes completa

4.6 Cálculo do CMC

Como já referimos, no problema do CMC, o método de eliminação de Gauss-Jordan não é passível de ser utilizado no cálculo das variáveis dependentes, uma vez que é de ordem mais elevada que o problema que nos propúnhamos resolver inicialmente. Esto não é necessariamente um problema insolúvel, uma vez que este cálculo pode ser feito numa fase separada. Isto porque é possível configurar previamente a RN de acordo com o grafo; depois, quando necessário, calcula-se o CMC entre uma dada origem e um dado destino. É necessário, no entanto, que seja possível calcular muito rapidamente os viés a aplicar às variáveis dependentes (segunda parcela do segundo membro da equação 4.5 — vector o), uma vez que a equação 4.1 se altera de cada vez que a origem ou o destino mudam.

Nesta subsecção, vamos apresentar um método de selecção das variáveis dependentes, que nos permite resolver o problema que acabámos de referir e aplicar directamente à RN o vector o da equação 4.5, que contém a informação relativa à origem e ao destino.

Para isso vamos começar por constatar que é possível à RN continuar a funcionar mesmo que haja variáveis dependentes que dependam de outra variáveis dependentes (e não apenas das independentes, como apresentámos nas subsecções anteriores).

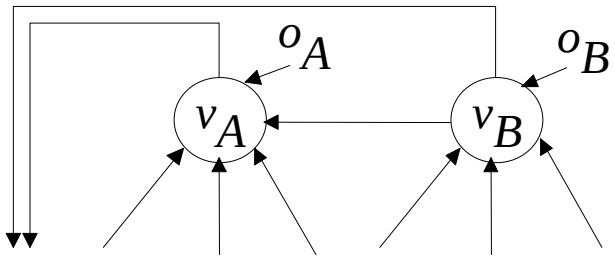


Figura 4.7: A variável dependente v_A depende de outra variável dependente, v_B

Isto constitui uma generalização que deve obedecer à condição de não ser fechado um ciclo de dependências, i.e., se a variável v_A depende da variável v_B, (seja directamente, seja por intermédio de outras variáveis dependentes), então, B não pode depender de v_A (seja directamente, seja indirectamente). No que diz respeito ao cálculo da realimentação e dos viés, se v_A depende de v_B, então, o que acontece é que v_A depende de todas as variáveis independentes de que v_B depende, de acordo com as equações 4.17, 4.18 e 4.19. Os neurónios relativos às variáveis v_A e v_B estão representados na figura 4.7.

$$v_B = \sum_{k=N}^{M} \alpha_{Bk} v_k + o_B \tag{4.17}$$

$$v_A = \alpha_{AB} v_B + \sum_{k=N}^{M} \alpha_{Ak} v_k + o_A \tag{4.18}$$

$$v_A = \alpha_{AB} \left(\sum_{k=N}^{M} \alpha_{Bk} v_k + o_B \right) + \sum_{k=N}^{M} \alpha_{Ak} v_k + o_A \tag{4.19}$$

Como não há ciclos entre variáveis dependentes é possível, para efeitos do cálculo dos viés e da realimentação das variáveis independentes, chegar a uma equação com o aspecto da 4.5. Em termos das dependências lineares entre variáveis não nos interessa este resultado, mas antes um outro que nos permita alterar instantaneamente a origem e o destino, conforme o percurso em causa.

No entanto, para podermos alterar instantaneamente as origens e os destinos do fluxo (vector o, que é aplicado nos viés das variáveis dependentes), interessa-nos o formato das equações 4.17 e 4.18, mas não o resultado da equação 4.19, que é mais

```
Inicia o conjunto N para conter todos os nós
Inicia dois conjuntos vazios:  V (vizinhos) e T (tratados)
Para todos os nós
   Se T é vazio
      Selecciona um nó qualquer de N e retira-o (S)
      Põe S em T
      Passa os nós vizinhos de S de N para V.
   Senão
      Se V é vazio
         Termina (com erro se houver nós em N)
      Senão
         Selecciona um nó qualquer de V e retira-o (S)
         Selecciona um vizinho qualquer de S
            que esteja em T (R)
         Se existe o arco RS
            RS é a variável dependente
         Senão
            SR é a variável dependente
         Escreve a equação das restrições para o nó S tendo
            em conta a origem e o destino do fluxo
         Selecciona os vizinhos de S que estão em N
         Passa-os para V
         Passa o nó S de V para T
```

Figura 4.8: Algoritmo para determinação das equações que representam as restrições do grafo

difícil de calcular, sobretudo quando há vários níveis de dependências entre variáveis dependentes. Por este motivo, para efeitos da concretização em *hardware* é mais simples manter as dependências entre variáveis dependentes e alterar directamente os seus viés sempre que necessário, sem efectuar qualquer cálculo adicional.

Põe-se agora a questão de saber se é possível construir uma RN de Variáveis Dependentes (VD) aproveitando a generalização que acabámos de descrever, sem aplicar o método de Gauss-Jordan e evitando a existência de ciclos. A resposta a esta questão é sim, mas não é totalmente óbvia a forma de seleccionar um conjunto de variáveis dependentes que obedeçam às condições anteriormente referidas.

O algoritmo que apresentamos na figura 4.8 permite seleccionar as $N-1$ equações, além duma variável dependente por cada equação. No conjunto das variáveis dependentes assim seleccionadas não existem garantidamente ciclos. O algoritmo mantém

três conjuntos de nós: T — tratados pelo algoritmo; V — todos os vizinhos dos nós já tratados; N — todos os restantes nós. No início os conjuntos T e V estão vazios e o conjunto N inclui todos os nós.

A ideia é que para um nó entrar no conjunto T tem que passar pelo conjunto V (excepto o primeiro nó a ser tratado). Quando um nó S é passado de V para T, é utilizada a equação que exprime a restrição do grafo associada a esse mesmo nó S. Note-se que, para que o nó S esteja em V é necessário que haja antes, pelo menos, um vizinho de S no conjunto T. Seja R um desses vizinhos. A variável dependente utilizada será precisamente a que representa o arco RS (v_{RS}), se existir, ou SR (v_{SR}) caso contrário (pelo menos um destes arcos existe senão R e S não seriam vizinhos).

Nem a variável que representa o arco RS nem a que representa o arco SR poderão voltar a entrar em qualquer das equações que vierem a ser escritas num passo posterior do algoritmo (uma vez que S passou para o conjunto T e o nó R já lá estava). Note-se que v_{RS} e v_{SR} já tinham entrado numa equação (relativa ao nó R), mas, ainda assim, como nenhuma outra variável dependente poderá vir a depender de v_{RS} ou v_{SR} (ou de qualquer outra que dependa destas duas) a existência de ciclos entre variáveis dependentes é impossível.

Não está incluído no algoritmo o processo de eliminar interdependências entre variáveis dependentes, que permita calcular um resultado semelhante ao da equação 4.19 que permite calcular o viés e a realimentação das variáveis independentes, por se tratar de um problema de solução trivial.

4.7 Conclusão

Esta solução pretende resolver dois problemas essenciais existentes nas RN de Ali e Kamoun e de Park e Choi.

Um dos problemas, e o que diz respeito ao número de neurónios, pode-se considerar resolvido. Com efeito, nas RN de VD, o número de neurónios depende linearmente em vez de quadraticamente do número de nós. Mais exactamente, o número

de neurónios necessário é igual ao número de ligações físicas, número este que tem a mesma ordem de grandeza do número de nós.

Outro dos problemas diz respeito à validade das soluções. Nas RN que apresentámos no capítulo 3, como todos os neurónios podem variar livremente, torna-se muito complicado garantir que a dinâmica conjunta leva a RN a convergir para um resultado válido. Este problema, que já era conhecido, vai ser bem evidente no capítulo 5. Neste aspecto, as RN de VD representam uma evolução evidente porque, agora, basta garantir que a actividade interna final de cada um dos neurónios é 0 ou 1. Verificada esta garantia é certo que a solução obtida é válida (embora não necessariamente óptima). Como veremos também no capítulo 5, este facto constitui uma vantagem sensível, quando comparados os resultados e no que diz respeito à obtenção de soluções válidas.

A desvantagem das RN de VD reside na maior dificuldade com que poderão ser concretizada em *hardware*. Com efeito, nas RN de VD, e ao contrário do que sucede nas RN de Ali e Kamoun e de Park e Choi, não é possível usar sempre os mesmos pesos entre neurónios. Agora, uma alteração na rede de dados pode introduzir uma alteração na configuração da RN — que poderá ser mais ou menos difícil de efectuar, consoante a concretização que for feita dessa mesma RN.

5

Comparação de Resultados

5.1 Introdução

Nos capítulos 3 e 4 apresentámos diversas soluções com RN capazes de procurar CMC em redes de dados.

Neste capítulo vamos apresentar um conjunto de cenários, que nos permitirão estabelecer uma comparação entre as diversas RN que apresentámos, usando sempre como referência o algoritmo de Dijkstra. Para efectuar as diversas comparações é utilizado um simulador de redes de dados com as alterações que, como veremos, serão necessárias para garantir QoS.

Antes de apresentarmos os resultados das comparações propriamente ditas, são definidos os parâmetros em que vai ser feita a avaliação, as alterações efectuadas ao simulador de rede, o processo de simular as RN em *software* e, além disto, é feita uma descrição exaustiva dos cenários da comparação. Por cenário de comparação entende-se, não só, a topologia da rede, mas também as aplicações que nela serão incluídas.

5.2 Critérios em Comparação

5.2.1 Parâmetros de QoS

Como tivemos ocasião de dizer na secção 2.5, nesta tese vamos considerar apenas um parâmetro de QoS: a largura de banda. Isto não impede porém, que sempre que possível, sejam considerados outros critérios, que no caso são o número de saltos e

o atraso das ligações. Mais precisamente, o número de saltos é considerado apenas como factor de desempate quando é utilizado o algoritmo de Dijkstra. Por sua vez, o atraso duma ligação física é ponderado no custo dessa mesma ligação, juntamente com o fluxo e a largura de banda totais, de acordo com a equação 5.1, onde d_l é o atraso da ligação, f_l é o fluxo (resultante do fluxo anterior mais a largura de banda pedida) e C_l a capacidade da ligação. Veja-se (Scharwtz, 1987) para mais detalhes relacionados com esta equação.

$$custo_l = f_l \times d_l + \frac{f_l}{C_l - f_l} \tag{5.1}$$

A primeira parcela desta equação representa o atraso nas ligações físicas. A segunda parcela corresponde ao termo dentro do somatório da equação 3.25 e procura representar o atraso introduzido pelas filas de espera à entrada de cada ligação. Nos diversos testes que teremos ocasião de descrever utilizámos o custo que acabámos de apresentar nas ligações físicas, mas desprezámos o atraso na propagação do sinal, ou seja, considerámos apenas a segunda parcela da equação 5.1.

5.2.2 Dispositivos a Comparar e Objectivos da Comparação

Resultou do capítulo 3, que as RN que mais se adequavam para fazer encaminhamento eram as de Ali e Kamoun e de Park e Choi. Vamos, por este motivo, utilizar estas duas arquitecturas nos testes a realizar. Iremos também testar as RN de VD que apresentámos no capítulo 4. Além destas, vai ser usado o algoritmo de Dijkstra, que servirá de referência para a comparação.

Uma vez que o algoritmo de Dijkstra encontra sempre o óptimo, enquanto que as RN podem não o atingir, interessa conhecer dois resultados fundamentais, que na medida do possível terão que ser quantificados:

- que vantagem, em termos de tempo de execução, é que poderá advir da utilização duma RN;
- quão piores são as soluções obtidas pelas RN, quando utilizadas num ambiente

que simula o real.

Além dos quatro métodos para determinar C.M.C. numa arquitectura de rede com QoS, que acabámos de enumerar, vamos ainda utilizar o algoritmo de Dijkstra, mas em redes sem qualquer tipo de ligação ou reservas e, portanto, sem garantias de QoS. Este quinto método serve para validar a arquitectura com QoS.

Resumindo, o objectivo dos testes a realizar é o de comparar, por um lado, as soluções baseadas em RN com o algoritmo de Dijkstra, por outro, comparar as três soluções baseadas em RN; finalmente, aferimos também a qualidade da arquitectura com garantias de QoS que estamos a usar, comparando-a com uma arquitectura tradicional sem garantias.

5.2.3 Parâmetros da Avaliação

De acordo com o que foi referido na secção 2.3.3, vamos avaliar a qualidade das soluções propostas, segundo os seguintes parâmetros:

- tempo de atraso médio dos dados (equação 5.2);
- rácio de utilização das ligações físicas ou taxa de ocupação (equação 5.7);
- percentagem de rejeição de ligações (equação 5.3);
- percentagem de perdas (equação 5.4);

O tempo de atraso médio é calculado de acordo com a equação 5.2, onde T_i é o tamanho do pacote i, e d_i o tempo que vai desde que este é enviado pelo emissor até que é entregue ao receptor. Este não é mais do que um atraso ponderado que leva em consideração o tamanho de cada pacote que circula na rede.

$$d_{medio} = \frac{\sum_i T_i \times d_i}{\sum_i T_i} \tag{5.2}$$

As equações 5.3 e 5.4 permitem determinar, respectivamente, a percentagem de rejeição de ligações e a percentagem de perdas de dados. L_{na} e L_t são, respectivamente, as

ligações não aceites e as ligações tentadas (entre estas incluem-se as ligações aceites, as rejeitadas e as falhadas de que falaremos adiante); R é a percentagem de ligações rejeitadas e P a percentagem de perdas.

$$R = \frac{L_{na}}{L_t} \times 100 \tag{5.3}$$

$$P = \frac{\sum_{i=perdidos} T_i}{\sum_i T_i} \times 100 \tag{5.4}$$

Além das ligações não aceites pela rede de dados, por falta de recursos, definimos também o conceito de ligação falhada como sendo uma ligação pedida por uma aplicação, mas que por incapacidade do algoritmo de encaminhamento em seleccionar um percurso não foi concluída com sucesso. Nas ligações falhadas incluem-se duas situações diferentes: uma delas dá-se quando não existe nenhum percurso capaz de satisfazer o pedido em questão (por não haver largura de banda suficiente, por exemplo); a outra situação ocorre quando a RN não converge para um resultado final válido, ainda que este exista. Este segundo caso não ocorre, obviamente, com o algoritmo de Dijkstra.

A percentagem de utilização das ligações físicas é, de entre estes, o parâmetro cujo cálculo se apresenta mais complexo. Começamos por definir a capacidade da rede como sendo o número de octetos que esta consegue comportar em cada instante (não contando com as filas de espera). A expressão da capacidade é dada pela equação 5.5, onde L_l é a largura de banda da ligação física l, d_l o respectivo atraso e C é a capacidade total, medida em dígitos binários, que resulta do somatório discreto efectuado sobre todas as ligações físicas existentes na rede.

$$C = \sum_l L_l \times d_l \tag{5.5}$$

No instante t, a capacidade utilizada da rede, C_{ur}, está representada na equação 5.6, onde $b_l(t)$ é a quantidade de dados (medida em dígitos binários) em trânsito na ligação física l.

$$C_{ur}(t) = \sum_l b_l(t) \tag{5.6}$$

Como $b_l(t) \leq L_l \times d_l$, isto significa que $C_{ur}(t)/C \leq 1$, como não poderia deixar de ser.

Dadas estas definições, entre o instante de tempo t_i e o instante t_f, o rácio médio de utilização da ligações físicas, R_u, é dado pela equação 5.7.

$$R_u = \int_{t_i}^{t_f} \frac{C_{ur}(t)}{C} = \int_{t_i}^{t_f} \frac{\sum_l b_l(t)}{\sum_l L_l \times d_l} \tag{5.7}$$

5.3 Descrição do Simulador de Rede

Para fazer a validação experimental dos diversos dispositivos / algoritmos de encaminhamento, bem como da arquitectura de rede com QoS, foi utilizado o simulador de rede ns-2.1b4 — *Network Simulator* versão 2.1b4.

O ns-2 é um simulador orientado para redes sem ligações, que se adequa facilmente para testar uma rede IP, mas não permite que sejam simuladas redes com garantias de QoS, pelo menos sem alterações consideráveis. De qualquer forma, o facto de ser bastante modular, de ser gratuito e de ter o código fonte disponível para possíveis alterações, elegeram-no para a tarefa que nos propúnhamos realizar.

No apêndice A encontra-se um manual que descreve o funcionamento interno do ns-2, que não se encontra, até este momento, devidamente documentado nos manuais que acompanham o pacote de *software*. Este conhecimento revelou-se necessário à realização das alterações ao simulador, que permitiram construir a arquitectura com QoS, de forma a ser possível efectuar reservas de recursos baseadas na largura de banda. Estas alterações constituíram uma fracção significativa do trabalho desta tese e estão resumidas no apêndice B.

Em termos de funcionamento do simulador, estas alterações têm reflexo sobretudo no comportamento das aplicações, que antes de poderem enviar dados têm que estabelecer uma ligação até ao destino. Caso não o façam os seus pacotes são pura e simplesmente descartados. As ligações são estáticas, na medida em que não há a capacidade de alterar o seu percurso em caso de falha de algum nó ou ligação física da rede, embora possam ser dinamicamente iniciadas e terminadas a qualquer momento.

Outra das limitações do ambiente de testes diz respeito a endereços *multicast*, que não são suportados, mas este é precisamente um dos assuntos que está para além dos objectivos desta tese e que poderá ficar para estudo posterior.

Finalmente, também não há controlo do fluxo produzido pelas aplicações, que se assumem bem comportadas, mas isto é um problema relativamente simples de resolver, sempre que o experimentador puder escolher aplicações que gerem tráfego que satisfaça os requisitos pretendidos. Como teremos ocasião de verificar num dos testes que realizámos, o desrespeito desta regra resulta em perdas no desempenho da rede, nomeadamente no que diz respeito à quantidade de pacotes descartados por sobrecarga nas ligações.

Convém também que fique claro que, na medida do possível, todo o mecanismo necessário ao estabelecimento das ligações foi excluído dos cenários de realização dos testes. A ideia é que nestes cenários os acontecimentos decorram em três tempos diferentes: estabelecimento de ligações; troca de pacotes; terminação das ligações.

Naturalmente que um cenário construído nestes termos apresenta algumas limitações, já que não considera o aparecimento de novas ligações e eliminação de ligações existentes, mas procura minimizar as interferências do protocolo que estabelece as ligações nos resultados estatísticos finais.

5.4 Simulação por *Software* das Redes Neuronais

Como temos vindo a referir ao longo do texto, as RN utilizadas nos testes foram as de Ali e Kamoun, Park e Choi e de VD.

A utilização destas três redes no ambiente de testes exigiu um programa capaz de fazer a sua simulação.

Como tivemos ocasião de ver no sistema de equações 3.14, o circuito eléctrico que constitui um neurónio e o conjunto de neurónios que compõem uma rede de Hopfield, na verdade, não fazem mais do que resolver um sistema de equações diferenciais.

Para resolver este sistema de equações diferenciais e para efeitos de cálculo

numérico utilizámos o método de Runge-Kutta de quarta ordem, sobre o sistema de equações representado na equação 5.8 (veja-se (Ali & Kamoun, 1993)).

$$\frac{dv_i}{dt} = \sum_{j=1}^{N} w_{ij}x_j - \frac{v_i}{\tau} + I_i \qquad i = 1, \ldots, N \tag{5.8}$$

Note-se que da equação 3.13, podemos facilmente passar para esta equação, à custa duma multiplicação dos pesos e dos viés por uma constante. Também sem perda de generalidade poderemos fazer $R = C = 1$ para efeitos da simulação por *software* (ver (Ali & Kamoun, 1993)).

Os pesos e os viés da rede de Ali e Kamoun estão representados nas equações 3.38 e 3.39, respectivamente, enquanto que os pesos e os viés da rede de Park e Choi estão representados nas equações 3.42 e 3.43. A forma de calcular os pesos e os viés da RN de VD foi abordada no capítulo 4.

A simulação decorreu essencialmente nas condições descritas em (Ali & Kamoun, 1993). O passo considerado para o método de Runge-Kutta foi de 10^{-5} segundos, sendo as simulações dadas por terminadas quando não houvesse qualquer neurónio cuja alteração da saída excedesse 10^{-5}, duma iteração para a seguinte.

Outro aspecto importante diz respeito aos custos utilizados na rede de dados, uma vez que os resultados obtidos pelas RN dependem da ordem de grandeza dos referidos custos. Por exemplo, uma das condições impostas em (Ali & Kamoun, 1993) é de que, na equação 3.34 $\mu_5 \gg \mu_1 \times (C_i)_{max}$. Por este motivo, nas RN de Ali e Kamoun e de Park e Choi, foi aplicado a todas as redes de dados um processo de normalização dos custos, segundo a expressão presente na equação 5.9.

$$C_{ln} = \frac{C_l - C_{min}}{C_{Max} - C_{min}} \cdot (C_{Maxn} - C_{minn}) + C_{minn} \tag{5.9}$$

Nesta expressão, C_{Maxn} e C_{minn} são, respectivamente, o custo máximo e mínimo desejados (ou normalizados). C_{Max} é o custo mais alto presente na rede original, C_{minn} o custo mais baixo e C_l o custo não normalizado da ligação física l. O resultado do cálculo desta expressão será um custo normalizado $C_{ln} \in [C_{minn}, C_{Maxn}]$. Nos testes que conduzimos fizemos $C_{Maxn} = 0,6$ e $C_{minn} = 0,03$. Note-se que, se o custo mínimo nor-

malizado for não nulo, a transformação é não linear, o que introduz alguma distorção nos custos normalizados (poderá acontecer, por exemplo, que $C_1 > C_2 + C_3$ antes da normalização e $C_{1n} < C_{2n} + C_{3n}$, depois da normalização). Algumas experiências indicaram uma clara vantagem ao nível do número de iterações necessárias à convergência e da validade das soluções apresentadas nas redes de Ali e Kamoun. De qualquer forma, o prejuízo introduzido pela normalização numa rede em que o número de saltos entre quaisquer dois nós seja limitado não é grande.

Na RN de VD foi igualmente aplicada uma normalização dos pesos, com os seguintes limites: $C_{Maxn} = 2$ e $C_{minn} = 0$. Isto significa que a normalização é linear, não havendo lugar a qualquer distorção.

O método de Runge-Kutta e o código escrito na linguagem de programação *C* utilizado para as simulações das RN são apresentados no apêndice C.

5.5 Ambiente de Testes

5.5.1 Topologia

Uma vez concluído que a forma mais adequada de comparar o desempenho das RN com o algoritmo de Dijkstra, em termos da qualidade das soluções obtidas é através de simulações, há que definir o ambiente em que estas terão lugar.

O primeiro aspecto a definir é precisamente a topologia da rede de dados a considerar. Na ausência duma rede tipo resolvemos considerar 18 topologias diferentes, organizadas segundo os seguintes parâmetros:

- redes com 20, 30 e 40 nós;
- para cada uma destas dimensões, redes com hierarquia e redes sem hierarquia;
- para cada dimensão e para cada tipo de hierarquia, considerar 3 exemplares distintos.

Para definir todas estas redes foram utilizadas duas aplicações de *software* — *gt-itm* e *sgb2ns*.

Gt-itm é um acrónimo para *Georgia Tech Internetwork Topology Models*, que é o nome duma aplicação que permite criar grafos de forma pseudo-aleatória, a partir de um conjunto de características previamente especificadas. Os grafos podem ser planos (i.e., sem hierarquia) ou podem estar organizados em dois tipos de hierarquias (denominadas de N-níveis e *transit-stub*, respectivamente — nesta tese utilizámos este segundo tipo de hierarquia). O gt-itm é construído sobre o *Stanford GraphBase* (sgb), que é um conjunto de estruturas de dados e de rotinas que permitem representar e manipular grafos (veja-se (Knuth, 1994)). É precisamente num formato especificado por esta plataforma, que são representados os grafos criados pelo gt-itm.

Para converter os ficheiros do formato sgb para um formato inteligível pelo simulador de rede ns-2 foi usada a aplicação *sgb2ns*, escrita por Haobo Yu.

Acresce dizer, que procurámos definir as redes hierárquicas e não hierárquicas de tal forma que o número de ligações físicas existente (i.e., o número de arestas dos grafos) fosse igual para cada número de nós (20, 30 ou 40). É ainda de referir que, as redes geradas da forma que acabámos de descrever só iniciam o atraso das ligações, deixando ao critério do experimentador a determinação da largura de banda, que será igual para todas as ligações físicas. Por este motivo, todas as ligações de todas as redes que utilizámos nas simulações têm uma largura de banda de 1 Mbps (125 koctetos / segundo).

Os parâmetros utilizados para definir cada uma das redes estão representados na tabela 5.1. Antes de tentarmos esclarecer o significado de cada um deles, apresenta-

	20 nós	30 nós	40 nós
sem hierarquia	geo 5 20 20 3 0.13	geo 5 30 30 3 0.086	geo 5 40 40 3 0.066
com hierarquia	ts 5 3 0 0 1 20 3 1.0 2 20 3 0.8 3 10 3 0.6	ts 5 3 0 0 1 20 3 1.0 3 20 3 0.8 3 10 3 0.6	ts 5 3 0 0 1 20 3 1.0 3 20 3 0.8 4 10 3 0.6

Tabela 5.1: Parâmetros de definição das redes de dados a simular

mos as figuras 5.1 e 5.2 que, respectivamente, exemplificam uma rede com 20 nós, não hierárquica, e uma rede de 39 nós, hierárquica, ambas resultantes dos parâmetros definidos na tabela 5.1 [1]. Em termos breves, o significado de cada uma das linhas de

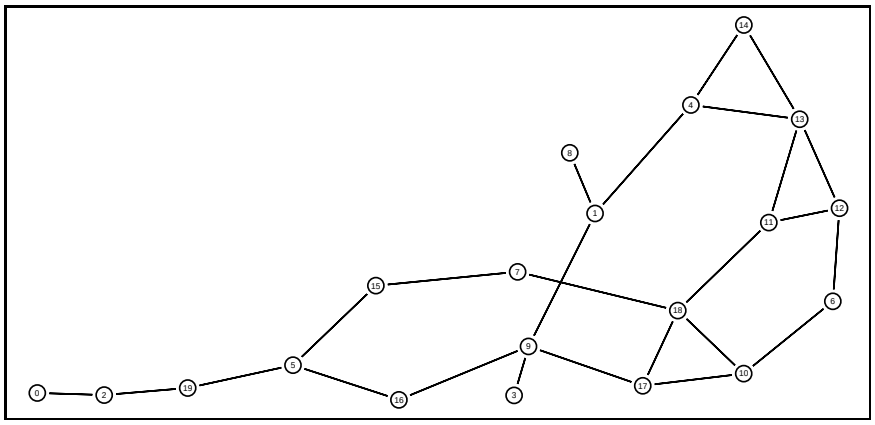


Figura 5.1: Rede não hierárquica de 20 nós

definição dos grafos é o seguinte:

- a primeira linha indica o tipo de grafo (no caso, *geo* — plano — ou *ts* — *transit-stub*) e o número de grafos que devem ser criados (note-se que como são criados cinco grafos ignorámos dois deles nos testes que realizámos, já que só usámos três exemplares de cada tipo de rede);

- nos grafos não hierárquicos (*geo*), os parâmetros existentes na segunda linha indicam, pela ordem em que aparecem, o número de nós, a escala (da qual resulta o cálculo das distâncias), o método utilizado para determinar a existência de arestas entre os nós (o método indicado pelo número três é puramente aleatório) e, finalmente, o último parâmetro indica a probabilidade de existir uma aresta entre dois nós;

- nos grafos hierárquicos (*ts*), os parâmetros existentes na segunda linha indicam, respectivamente, o número de subgrupos de nós por cada nó do grupo de trânsito (sempre igual a três no nosso caso — nós 0, 1 e 2, da figura 5.2), o número de arestas adicionais entre subgrupos e grupo(s) de trânsito, e o número de arestas entre

[1] Nas redes hierárquicas as redes não têm 40 nós, mas apenas 39. O mesmo não se passa nas redes planas, que têm efectivamente 40 nós. Por este motivo, vamos adoptar para ambos os tipos a designação genérica de redes com 40 nós.

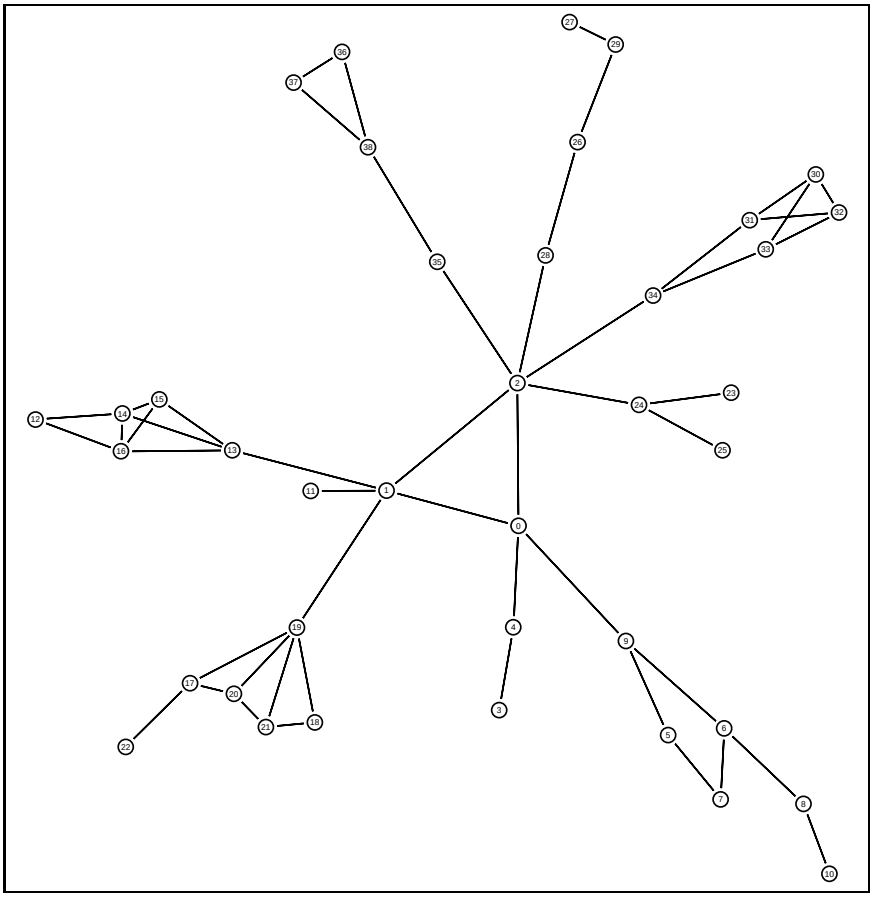


Figura 5.2: Rede hierárquica de 39 nós

subgrupos diferentes (sendo que fizemos estes dois últimos parâmetros sempre iguais a zero);

- nos grafos hierárquicos, a terceira linha define os parâmetros do nível de topo, tal como são definidos para os grafos não hierárquicos (número de nós, escala, método de determinação de arestas e respectiva probabilidade), a quarta e a quinta linha especificam estes parâmetros para o(s) grupo(s) de trânsito (que aqui é sempre apenas um, já que o primeiro parâmetro da terceira linha é igual a um) e para os subgrupos, usando exactamente os mesmos que acabámos de enumerar para o nível de topo.

Da descrição que acabámos de fazer e da análise da figura 5.2, resulta que o número médio de nós é igual ao número de nós no domínio de trânsito $\times$ (1 + número de subgrupos por nó do nível de trânsito $\times$ número de nós por subgrupo). Para os três

diferentes tamanhos de redes temos:

$$2 \times (1 + 3 \times 3) = 20$$

$$3 \times (1 + 3 \times 3) = 30$$

$$3 \times (1 + 3 \times 4) = 39$$

5.5.2 Cenários

Criadas as topologias, têm que ser definidos outros parâmetros, em particular, quantos e que circuitos virtuais (i.e. conexões) devem ser estabelecidos na rede e que fontes de tráfego vão ser utilizadas.

Quanto ao número de circuitos virtuais este foi tomado como igual ao número de nós existente na rede. Para cada rede utilizámos separadamente dois tipos de fontes de tráfego com débitos idênticos em termos estatísticos: fontes deterministas de débito constante (*Constant Bit Rate — cbr*) e fontes estocásticas (*exponenciais*). Em ambos os casos, todas as fontes utilizadas num determinado cenário têm precisamente as mesmas características. Naturalmente que esta situação não é muito realista mas, como veremos, é suficiente para chegarmos a algumas conclusões esclarecedoras.

As fontes deterministas apresentam um débito constante de 400 kbps (50 koctetos / s enviados em pacotes de 200 octetos espaçados por 0,004 s). As fontes estocásticas são ligeiramente mais difíceis de caracterizar, podendo encontrar-se num de dois estados possíveis: activas ou inactivas. Só quando estão activas é que geram tráfego, na forma de pacotes com o tamanho fixo de 200 octetos a um débito de 800 kbps. A transição de um estado de actividade para o seguinte é controlada por um processo aleatório que obedece a uma distribuição exponencial. O tempo médio entre transições é de 200 ms. Como estas fontes passam, em média, metade do tempo inactivas, espera-se que o seu débito médio seja também de 400 kbps. Obviamente que a gestão da rede com fontes deste tipo é mais complicada porque, provavelmente, haverá períodos de tempo em que o débito despejado para rede será muito superior ao valor médio contratado com as fontes.

Note-se que, para cada fonte de tráfego, deverá ser estabelecido um circuito virtual, i.e., uma reserva. É de referir que muitas das fontes acabam por não participar activamente na simulação, porque as suas reservas não são aceites pelo módulo de CAC, seja por deficiência da RN que faz o encaminhamento seja, simplesmente, porque não há largura de banda suficiente na rede.

Como é evidente, os cenários produzidos de acordo com a descrição que acabámos de fazer representam uma simplificação grosseira da realidade. Porém, como teremos ocasião de verificar, servirão como casos limite para verificar a validade da arquitectura com QoS, que desenvolvemos. Como teremos ocasião de verificar, os resultados obtidos alteram-se consideravelmente dum cenário com fontes de tráfego deterministas, para outro com fontes de tráfego estocásticas.

Convém compreender que, para o mesmo cenário, os resultados do encaminhamento não diferem das fontes deterministas para as estocásticas, porque as fontes efectuam reservas precisamente nos mesmos instantes e anunciam exactamente os mesmos valores de fluxo de tráfego, para os mesmos destinos. A única diferença eventual poderá surgir da inicialização aleatória que se faz às RN.

Para as 18 redes diferentes e dados estes cenários, testámos os quatro métodos de encaminhamento que descrevemos (algoritmo de Dijkstra, RN de Ali e Kamoun, RN de Park e Choi e RN de VD), além dum quinto cenário de controlo sem reservas e portanto sem garantias de QoS. Neste, utilizámos a mesma configuração da rede, mas sem qualquer controlo das fontes. À partida, o resultado que esperávamos obter era uma utilização da rede até ao ponto de saturação de algumas ligações físicas e consequente perda de pacotes nas filas à entrada dessas ligações.

É ainda importante referir os seguintes aspectos relativos aos cenários que construímos:

- cada ligação física suporta no máximo duas ligações, uma vez que a largura de banda é de 1 Mbps e o fluxo anunciado pelas aplicações é de 400 kbps;
- como as ligações são todas pedidas com um intervalo de tempo muito pequeno, há actualizações do estado das ligações físicas que não se propagam a tempo de

serem consideradas.

Estes factos explicam porque é que mesmo no algoritmo de Dijkstra, que encontra sempre um percurso válido (desde que isso seja possível) em face dos dados que dispõe, há ligações recusadas. Esta situação levanta também a possibilidade de haver decisões subóptimas no encaminhamento, mas este é um problema com que já contávamos à partida e que dificilmente poderíamos resolver. Esta é aliás uma das justificações mais fortes para se utilizar encaminhamento pela origem e não salto a salto, como tivemos ocasião de ver no capítulo 2.

5.6 Resultados

Os resultados estatísticos que vamos apresentar dividem-se em três grupos diferentes. Dois dos grupos contêm dados relativos à avaliação de diversos parâmetros exclusivos da rede de dados. São eles a percentagem de perdas de dados, a taxa de ocupação da rede e o atraso médio dos dados. O terceiro grupo procura avaliar os métodos de encaminhamento propriamente ditos, à custa da contabilização do número de reservas que foram pedidas à rede e que foram aceites ou rejeitadas; as reservas podem ainda não ser sucedidas por deficiência do método de encaminhamento — este número também é contabilizado. No caso das RN é indicado o número de iterações necessário à sua convergência.

Cada gráfico apresentado é o resultado dos valores médios de três cenários diferentes. Assim, por exemplo, quando é dito que os resultados se referem ao cenário *geo40*, isto significa que foram simulados três cenários diferentes todos eles com redes não hierárquicas de 40 nós, de acordo com o que já referimos.

É importante referir, desde já, que os gráficos que representam os resultados estatísticos não podem ser analisados separadamente. A forma correcta de fazer esta análise será vista mais adiante.

Da figura 5.3, até à 5.8 e depois da figura 5.9 até à 5.14 são apresentadas as estatísticas relativas à ocupação, perdas e atraso na rede para fontes de tráfego determi-

nista (*cbr*) e para fontes de tráfego estocásticas (*exp*), respectivamente.

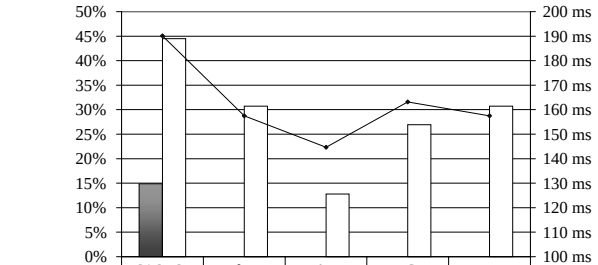
Da figura 5.15 até à 5.20 apresentamos o número de chamadas aceites, rejeitadas e falhadas para cada um dos cenários existentes. É ainda apresentado o número de iterações de que necessitaram as RN para convergir, na simulação pelo método de Runge-Kutta.

Antes de entrarmos na análise dos resultados propriamente dita, convém sabermos quais as conclusões que podemos tirar a partir dos dados que temos.

O primeiro facto a ter em conta, é que não é muito fácil tirar conclusões apenas a partir da capacidade ocupada e do atraso. Como existe uma forte correlação entre estes dois dados, acontece que a um atraso elevado corresponde normalmente uma taxa de ocupação elevada, sendo que estes dados não garantem um aproveitamento eficiente da rede. Muito pelo contrário, os atrasos e taxa de ocupação elevados podem ocorrer por decisões deficientes do mecanismo de encaminhamento.

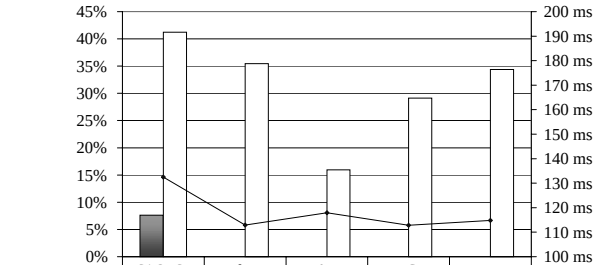
Para podermos tirar alguma conclusão fiável necessitávamos de saber o débito que a rede está a conseguir entregar às aplicações. Esse dado não está disponível, mas pode ser inferido a partir do número de reservas aceites e da taxa de perdas, sabendo que todas as fontes de tráfego emitem com o mesmo débito.

Há ainda alguns casos em que a uma taxa de ocupação mais elevada não corresponde o atraso mais elevado. É o caso, por exemplo, do algoritmo de Dijkstra, no cenário *ts40* com fontes deterministas — figura 5.7. Neste caso, e depois de verificarmos que é com este algoritmo que a rede atinge o débito máximo (15,3 reservas aceites em média, segundo a figura 5.19), é evidente que isso se deve a um conjunto mais eficiente de decisões de encaminhamento. Exactamente o contrário se passa com a RN de Ali e Kamoun no mesmo cenário, que tem a menor taxa de ocupação da rede e um atraso muito elevado.



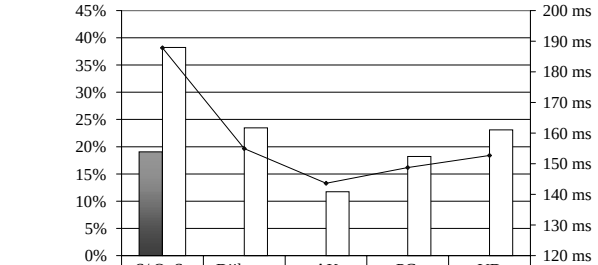
	S/ QoS	Dijkstra	AK	PC	VD
Perdas	15%	0%	0%	0%	0%
Taxa Ocup.	44%	31%	13%	27%	31%
Atraso	190,1 ms	157,5 ms	144,6 ms	163,2 ms	157,5 ms

Figura 5.3: Estatística *ts20* (*cbr*)



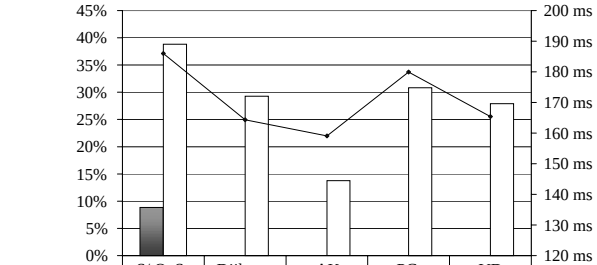
	S/ QoS	Dijkstra	AK	PC	VD
Perdas	8%	0%	0%	0%	0%
Taxa Ocup.	41%	35%	16%	29%	34%
Atraso	132,5 ms	112,9 ms	117,9 ms	112,8 ms	114,8 ms

Figura 5.4: Estatística *geo20* (*cbr*)



	S/ QoS	Dijkstra	AK	PC	VD
Perdas	19%	0%	0%	0%	0%
Taxa Ocup.	38%	23%	12%	18%	23%
Atraso	187,8 ms	154,9 ms	143,6 ms	148,8 ms	152,7 ms

Figura 5.5: Estatística *ts30* (*cbr*)



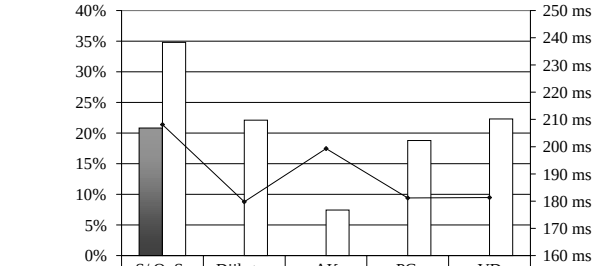
	S/ QoS	Dijkstra	AK	PC	VD
Perdas	9%	0%	0%	0%	0%
Taxa Ocup.	39%	29%	14%	31%	28%
Atraso	186,0 ms	164,3 ms	159,0 ms	179,9 ms	165,4 ms

Figura 5.6: Estatística *geo30* (*cbr*)

5.7 Análise dos Resultados

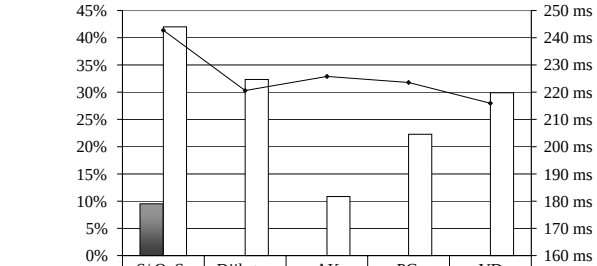
5.7.1 Análise da Arquitectura com QoS

Uma análise comparativa dos gráficos que dizem respeito às fontes deterministas e dos gráficos relativos às fontes estocásticas permitem-nos tirar algumas conclusões



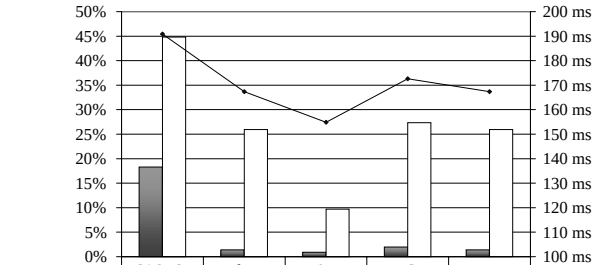
	S/ QoS	Dijkstra	AK	PC	VD
Perdas	21%	0%	0%	0%	0%
Taxa Ocup.	35%	22%	7%	19%	22%
Atraso	208,1 ms	179,7 ms	199,3 ms	181,2 ms	181,3 ms

Figura 5.7: Estatística *ts40* (*cbr*)



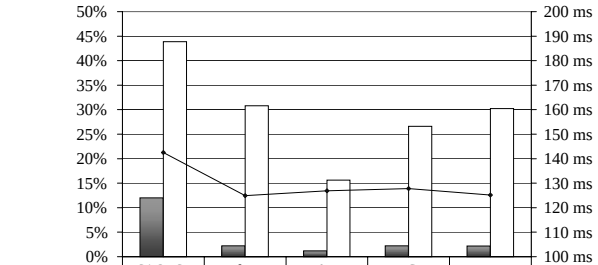
	S/ QoS	Dijkstra	AK	PC	VD
Perdas	10%	0%	0%	0%	0%
Taxa Ocup.	42%	32%	11%	22%	30%
Atraso	242,8 ms	220,6 ms	225,8 ms	223,6 ms	215,9 ms

Figura 5.8: Estatística *geo40* (*cbr*)



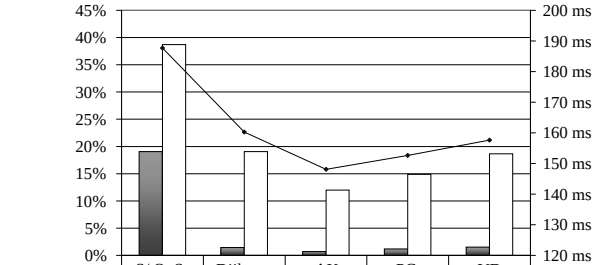
	S/ QoS	Dijkstra	AK	PC	VD
Perdas	18%	1%	1%	2%	1%
Taxa Ocup.	45%	26%	10%	27%	26%
Atraso	190,9 ms	167,3 ms	154,8 ms	172,6 ms	167,3 ms

Figura 5.9: Estatística *ts20* (exp.)



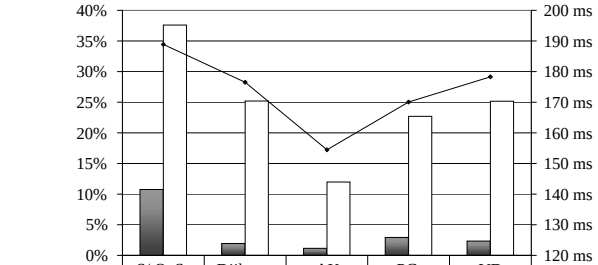
	S/ QoS	Dijkstra	AK	PC	VD
Perdas	12%	2%	1%	2%	2%
Taxa Ocup.	44%	31%	16%	27%	30%
Atraso	142,5 ms	125,0 ms	126,9 ms	127,8 ms	125,2 ms

Figura 5.10: Estatística *geo20* (exp.)



	S/ QoS	Dijkstra	AK	PC	VD
Perdas	19%	1%	1%	1%	1%
Taxa Ocup.	39%	19%	12%	15%	19%
Atraso	187,7 ms	160,3 ms	148,1 ms	152,6 ms	157,6 ms

Figura 5.11: Estatística *ts30* (exp.)

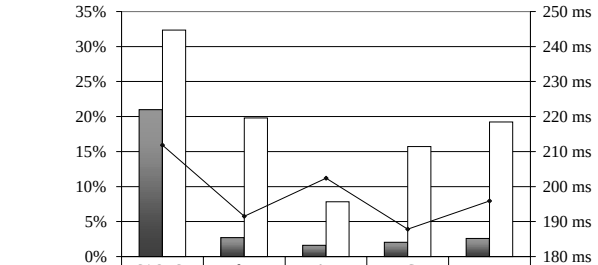


	S/ QoS	Dijkstra	AK	PC	VD
Perdas	11%	2%	1%	3%	2%
Taxa Ocup.	38%	25%	12%	23%	25%
Atraso	188,9 ms	176,5 ms	154,5 ms	170,1 ms	178,3 ms

Figura 5.12: Estatística *geo30* (exp.)

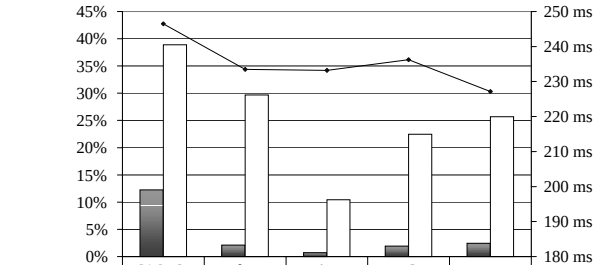
importantes.

A primeira delas é que a arquitectura com QoS, mesmo sendo baseada apenas no valor médio da largura de banda é eficiente. Para isso basta verificar que as perdas e o atraso são sempre inferiores aos que ocorrem no cenário de controlo sem QoS. Não podemos contudo esquecer que há um preço a pagar por isso, e que esse preço é a menor utilização da rede. Com efeito, há sempre fontes de tráfego que ficam de fora



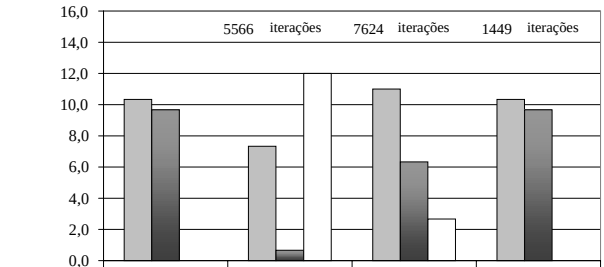
	S/ QoS	Dijkstra	AK	PC	VD
Perdas	21%	3%	2%	2%	3%
Taxa Ocup.	32%	20%	8%	16%	19%
Atraso	211,8 ms	191,5 ms	202,4 ms	187,8 ms	195,9 ms

Figura 5.13: Estatística *ts40* (exp.)



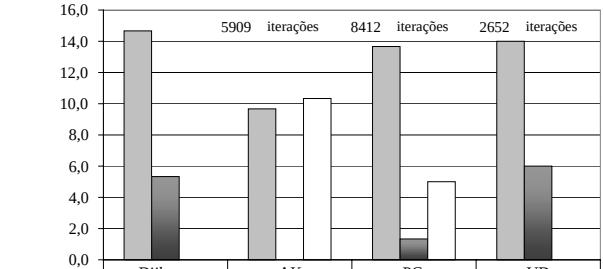
	S/ QoS	Dijkstra	AK	PC	VD
Perdas	12%	2%	1%	2%	2%
Taxa Ocup.	39%	30%	10%	22%	26%
Atraso	246,5 ms	233,5 ms	233,2 ms	236,2 ms	227,2 ms

Figura 5.14: Estatística *geo40* (exp.)



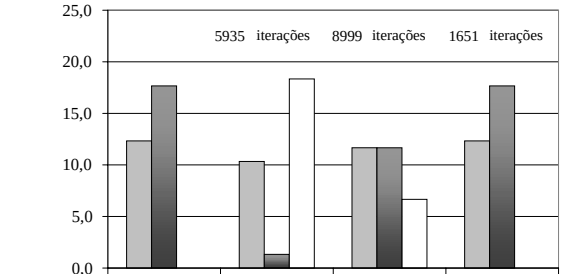
	Dijkstra	AK	PC	VD
Aceites	10,3	7,3	11,0	10,3
Rejeitadas	9,7	0,7	6,3	9,7
Falhadas	0,0	12,0	2,7	0,0

Figura 5.15: Reservas do cenário *ts20*



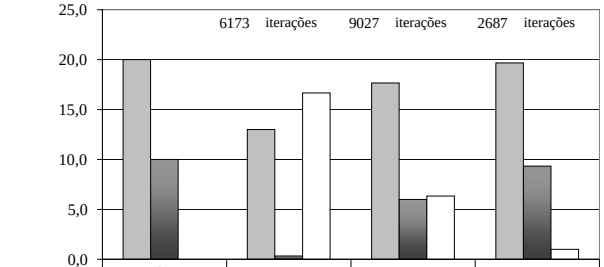
	Dijkstra	AK	PC	VD
Aceites	14,7	9,7	13,7	14,0
Rejeitadas	5,3	0,0	1,3	6,0
Falhadas	0,0	10,3	5,0	0,0

Figura 5.16: Reservas do cenário *geo20*



	Dijkstra	AK	PC	VD
Aceites	12,3	10,3	11,7	12,3
Rejeitadas	17,7	1,3	11,7	17,7
Falhadas	0,0	18,3	6,7	0,0

Figura 5.17: Reservas do cenário *ts30*

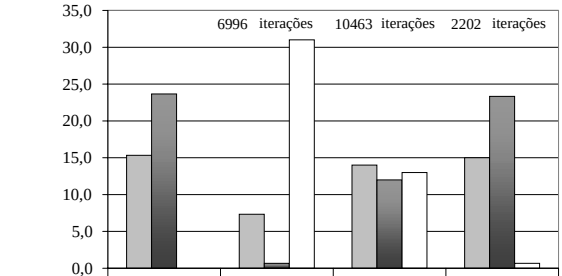


	Dijkstra	AK	PC	VD
Aceites	20,0	13,0	17,7	19,7
Rejeitadas	10,0	0,3	6,0	9,3
Falhadas	0,0	16,7	6,3	1,0

Figura 5.18: Reservas do cenário *geo30*

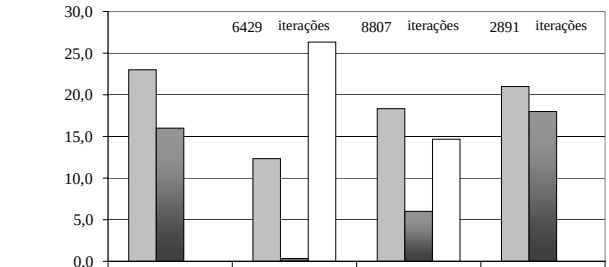
enquanto que, não havendo restrições, todas podem utilizar a rede.

A segunda conclusão a que podemos chegar é que esta arquitectura não é totalmente eficaz. No caso pouco realista em que todas as fontes geram tráfego a um débito constante (figuras 5.3 a 5.8) a rede consegue, de facto, obter perdas nulas. No outro caso, que é ligeiramente mais complexo (figura 5.9 a 5.14), os resultados, ao nível das perdas e atrasos, apresentam evidente degradação.



	Dijkstra	AK	PC	VD
Aceites	15,3	7,3	14,0	15,0
Rejeitadas	23,7	0,7	12,0	23,3
Falhadas	0,0	31,0	13,0	0,7

Figura 5.19: Reservas do cenário *ts40*



	Dijkstra	AK	PC	VD
Aceites	23,0	12,3	18,3	21,0
Rejeitadas	16,0	0,3	6,0	18,0
Falhadas	0,0	26,3	14,7	0,0

Figura 5.20: Reservas do cenário *geo40*

5.7.2 Cenário com Fontes Deterministas

No cenário em que utilizámos fontes de tráfego puramente deterministas (figuras 5.3 a 5.8), os resultados são relativamente simples de analisar. O primeiro facto que salta à vista é que quando não são estabelecidas reservas (i.e., quando não há QoS) se consegue um aproveitamento máximo da capacidade da rede. O preço a pagar pelo maior aproveitamento é um atraso maior (como geralmente acontece) e a existência de perdas, que chegam a atingir 21% no cenário *ts40* (figura 5.7). Esta conjugação ocorre em todos os cenários: aqueles que não têm QoS são, sem excepção, os que têm níveis de ocupação, atraso e perdas mais elevados.

De um modo geral, podemos dizer que as RN apresentam resultados piores que o algoritmo de Dijkstra — o contrário, aliás, seria de estranhar. Acontece em diversos cenários que com RN se atinge um débito menor, acompanhado de atrasos mais elevados. É o caso da RN de Ali e Kamoun nos cenários *geo20*, *ts40* e *geo40* (além dos cenários com fontes estocásticas *geo20* e *ts20*); da RN de Park e Choi nos cenários *geo30*, *ts40* e *geo40* (*geo20* e *geo40* com fontes estocásticas); e da rede de VD nos cenários *geo20*, *geo30* e *ts40* (*geo20*, *geo30* e *ts 40* com fontes estocásticas).

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 RN, excepto no caso da rede *ts20*, onde a RN de Park e Choi consegue um número mais elevado de reservas. Exceptuando este caso, e entre as RN, as de VD conseguiram sempre o número mais elevado de reservas aceites. Pelo contrário, a solução de Ali e Kamoun é a que apresenta, de longe, os piores resultados. Comparativamente a esta, na RN de Park e Choi, a degradação não é tão significativa mas é, ainda assim, bastante sensível.

Estes resultados levam-nos a crer que, tal como era sugerido nalguns artigos (veja-se (Park & Choi, 1998)), a RN 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 RN de Park e Choi a degradação não é tão rápida, mas ainda assim acontece, sobretudo quando comparada com os resultados do algoritmo de Dijkstra. Nas RN de VD não podemos concluir que haja uma degradação nítida dos resultados, com o aumento do tamanho da rede, pelo menos até aos 40 nós, pois o número de reser-

vas bem sucedidas é apenas 10% inferior no cenário *geo40* e quase igual no *ts40* aos números homólogos do algoritmo de Dijkstra.

Sobre o cenário *ts20*, numa primeira análise, pode parecer estranho que a rede de Park e Choi consiga resultados superiores aos alcançados por um algoritmo que obtém o óptimo, em termos do número de reservas aceites, mas convém não esquecer que o algoritmo de Dijkstra encontra o óptimo, mas apenas para uma codificação em particular dos custos. Na codificação, que foi feita com base na fórmula expressa em 5.1, resolvemos ignorar os atrasos das ligações físicas, como tivemos oportunidade de dizer. É de referir ainda, que a própria dedução da equação 5.1 admite certas hipóteses que podem não se verificar nem no ambiente da simulação, nem na realidade.

O problema de encontrar a codificação ideal para os custos das ligações físicas é complicado de resolver e ultrapassa o âmbito desta tese, mas assumimos na análise dos resultados que, com outra codificação diferente, que permitisse ao algoritmo de Dijkstra obter os melhores resultados possíveis [2], as RN conseguiriam resultados não muito distantes daqueles que foram encontrados com esta codificação.

5.7.3 Cenário com Fontes Estocásticas

À parte da questão que já referimos, de que a arquitectura de rede com QoS obtém resultados mais fracos nestes cenários, em tudo o resto, os resultados assemelham-se muito aos descritos na subsecção anterior, salvo em situações pontuais, que tivemos oportunidade de enumerar. Em particular, continua a ser verdade que a RN de Park e Choi consegue nas redes de menores dimensões resultados que se aproximam dos alcançados pelo algoritmo de Dijkstra, e que estes resultados vão sendo cada vez piores à medida que o número de nós na rede aumenta. De igual forma, a RN de Ali e Kamoun obtém aqui resultados bastante fracos, enquanto que a RN de VD continua a ser a que obtém resultados mais próximos do algoritmo de Dijkstra.

[2]Repare-se que a ordem pela qual aparecem as reservas também influencia o desempenho global, uma vez que um algoritmo pode tomar uma decisão que se afigura óptima num dado momento, mas que pode vir a revelar-se incorrecta em face dos pedidos de reserva que surgem mais tarde. Mas esta é uma questão que dificilmente poderíamos aspirar a resolver no âmbito desta tese.

Outra situação não seria, aliás, de esperar, uma vez que os percursos seleccionados são os mesmos, mudando apenas a carga oferecida à rede pelas aplicações.

5.7.4 Análise do Resultado das Reservas

Um aspecto que é importante salientar é que a partir do número de reservas rejeitadas não é possível tirar conclusões fiáveis. Uma reserva rejeitada acontecerá, em princípio, por deficiência da arquitectura da rede e não do algoritmo de encaminhamento. Este tipo de situação acontecerá normalmente quando o nó que faz o encaminhamento tem informação desactualizada relativamente aos recursos disponíveis em algumas das ligações físicas da rede. Como vimos, isto não é um problema fácil de resolver, mas de qualquer forma não pode ser atribuído qualquer demérito ao método de encaminhamento pelo sucedido.

Pelo contrário, o número de reservas falhadas é um importante parâmetro de avaliação. Com efeito, uma reserva falhada resulta sempre duma incapacidade do algoritmo em encontrar uma solução. Esta incapacidade pode ser justificada pelo facto de não existir, de facto, nenhum percurso elegível com largura de banda suficiente, mas também pode ocorrer por incapacidade do próprio método de encaminhamento. Obviamente que esta segunda hipótese nunca ocorre com o algoritmo de Dijkstra. Aliás, por análise dos resultados, verificamos que a primeira hipótese também nunca ocorre com este algoritmo, pelo que é lícito concluir que, senão todas, pelo menos quase todas as reservas falhadas das RN se devem à sua própria incapacidade em encontrar um resultado e não a uma insuficiência de largura de banda da rede.

Em termos de número de reservas falhadas duas conclusões são imediatas: as redes de Ali e Kamoun falham muito mais reservas; e a proporção de reservas falhadas aumenta com a dimensão da rede, não só para as redes de Ali e Kamoun, como para as de Park e Choi, embora o mesmo já não se passe com as RN de VD. Este é um problema relevante por duas razões essenciais. Em primeiro lugar, seja qual for a dimensão da rede, não é admissível que as aplicações vejam as suas reservas bloqueadas por uma falha do método de encaminhamento, quando há recursos disponíveis — e esta situação acontece frequentemente com as duas arquitecturas de RN que referimos. Em segun-

do lugar, em redes com um reduzido número de nós é difícil justificar a substituição do algoritmo de Dijkstra por uma RN, porque as vantagens em termos de rapidez não poderão ser tão significativas em termos absolutos.

Quanto ao número médio de iterações necessárias à convergência, não deixa de ser interessante constatar que a RN de Ali e Kamoun converge mais depressa do que uma rede que lhe é posterior — a de Park e Choi —, pelo menos nos casos que experimentámos. Quanto à RN de VD os resultados, apesar de suplantarem os das outras RN, não podem ser tomados como garantidos por uma razão: outra escolha para os valores das constantes μ_1 e μ_2 da equação 4.14, que permita obter melhores resultados, pode ter por consequência uma alteração substancial do número de iterações necessárias à convergência. Em geral, quanto maior for μ_1 menor será o número de iterações necessário.

Não poderemos terminar esta análise sem estimar o tempo de que uma RN precisará, se realizada em *hardware*, para convergir para uma solução válida. Como dissemos, o passo utilizado no método de Runge-Kutta, é de 10^{-5} segundos; ora, se o método pode nalgumas simulações exceder as 10.000 iterações, então a convergência da RN poderá levar mais de 1 décimo de segundo ($10^{-5} \times 10.000$). A questão que nos interessa é que, ou se utiliza outra forma de concretizar uma RN de Hopfield, ou então, segundo o modelo puramente assíncrono representado pela equação 3.14, o tempo de convergência andará sempre por essa ordem de grandeza ou, na melhor das hipóteses, por volta das centésimas de segundo [3].

[3] Note-se que é difícil indicar se este tempo é muito ou pouco elevado, mas o problema é que um algoritmo de Dijkstra a ser executado num computador digital tenderá a ser cada vez mais rápido à medida que a tecnologia evoluir, de forma que qualquer solução baseada numa rede de Hopfield será cada vez menos competitiva.

6

Conclusão

A tese aborda a aplicação de RN à resolução do problema do encaminhamento. Começámos por fazer um levantamento dos componentes necessários a uma rede capaz de oferecer garantias de QoS. Verificámos neste estudo que entre estas necessidades se inclui um algoritmo de encaminhamento que considere parâmetros de QoS nas suas decisões. Mostrámos também que, para este fim, não é possível ou é, pelo menos, bastante limitativo construir encaminhadores que efectuem os cálculos dos percursos previamente e que continuem a recorrer a tabelas para determinar o próximo salto desses percursos.

Não sendo possível determinar os percursos previamente, então, é necessário calculá-los sempre que surja um pedido para estabelecer uma ligação ou reserva. Deste facto surge a necessidade de construir um mecanismo capaz de calcular percursos, que seja o mais rápido possível e que considere um conjunto de parâmetros de QoS.

Esta necessidade de elevado desempenho aponta para a utilização de RN, que têm o potencial de ser mais eficientes que um algoritmo executado sequencialmente. De modo a seleccionar o tipo de redes mais adequadas apresentámos um conjunto de soluções bem conhecidas, baseadas em RN, sendo duas delas redes de Hopfield capazes de procurar percursos óptimos numa rede de dados. Foram também identificadas as limitações destas redes.

Para resolver alguns dos problemas destas soluções, apresentámos uma nova configuração para uma rede de Hopfield, com duas camadas de neurónios, em vez de apenas uma, a que chamámos RN de Variáveis Dependentes (VD).

Finalmente, avaliámos as várias soluções consideradas ao longo da tese: RN de Ali

e Kamoun, RN de Park e Choi e RN de VD. Como base de comparação utilizámos o algoritmo de Dijkstra, que por encontrar sempre percursos óptimos serviu para estabelecer os limites a atingir.

Os diferentes métodos de encaminhamento foram aplicados a um conjunto de cenários que foram construídos para efectuar todas as comparações. Os valores apresentados foram obtidos recorrendo a simulação, tendo sido seleccionado o simulador de redes de dados ns-2. Uma parte do trabalho consistiu em expandir o ns-2 de forma a incluir uma arquitectura com suporte básico de QoS e de alguns requisitos que este suporte pressupõe, tais como a reserva de recursos.

Concluída esta arquitectura e efectuadas as comparações foi possível concluir que:

- o algoritmo de Dijkstra obtém, como seria de esperar, os melhores resultados;
- a RN de Ali e Kamoun obtém os piores resultados;
- a RN de Park e Choi, apesar de conseguir resultados nitidamente melhores que a RN anterior, apresenta grandes limitações quanto à validade da sua convergência (i.e., quanto à validade da solução obtida);
- a RN de VD é a que obtém os melhores resultados entre as RN e, sobretudo, apresenta um número de convergências mal sucedidas muito mais reduzido.

Pelo que dissemos e para redes de dados até, pelo menos, 40 nós, as RN de VD parecem, nas condições expostas nesta tese, estar muito próximas dos resultados conseguidos por um algoritmo capaz de determinar percursos óptimos.

Outro aspecto que considerámos foi a velocidade de convergência duma rede de Hopfield, se concretizada em *hardware*. Como vimos, não é garantido que os tempos de convergência sejam suficientemente baixos para justificar a sua utilização. No entanto, o desenvolvimento de concretizações capazes de oferecer essa garantia é um dos pontos de trabalho futuro para que esta tese aponta.

Para além disso, em termos de evolução futura deste trabalho pensamos que, a partir da RN de VD que apresentámos, será possível determinar novas funções de energia,

capazes de ultrapassar algumas das limitações existentes na arquitectura actual e que conduzam a melhorias nos resultados da convergência. Esta melhoria terá necessariamente que se reflectir na validade das soluções a obter, que deverá aumentar.

Outro aspecto a considerar, em termos de evolução da presente arquitectura, prende-se com a necessidade de tornar a RN facilmente configurável em função das alterações que surjam na rede de dados.

Uma última palavra para a questão da arquitectura com QoS. Utilizámos, numa primeira abordagem ao problema da construção desta arquitectura, a largura de banda, caracterizada em termos de média, como parâmetro de QoS. Será previsível que os resultados melhorem sensivelmente se, num trabalho futuro, a largura de banda vier a ser caracterizada por parâmetros de balde de testemunhos (*token bucket*), tal como previsto no QoSPF. Outra alteração que se afigura necessária é a formatação do tráfego das aplicações, também com o mecanismo do balde de testemunhos ou, eventualmente, com outro mecanismo diferente mas igualmente válido.

A

Network Simulator-2

A.1 Introdução

A.1.1 Objectivos

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 objectivo do presente capítulo é o de apresentar com o detalhe possível o funcionamento interno do simulador.

Tomou-se a opção de adicionar a este documento um capítulo com este assunto por duas razões essenciais, relacionadas entre si. Em primeiro lugar, porque o estudo do funcionamento do simulador constituiu uma tarefa de dificuldade relevante no âmbito do trabalho que temos vindo a apresentar. Em segundo lugar, porque a documentação

[1] A *release* que apresentamos neste documento é a 2.1b4.

anexada ao simulador (Fall & Varadhan, 1999) é, quanto a nós, bastante deficiente e não permite uma imediata compreensão das partes que são relevantes no seu funcionamento. Assim, pretende-se que este capítulo possa esclarecer dúvidas sobre assuntos não abordados ou insuficientemente abordados no manual do simulador, para todos quantos o venham a utilizar no futuro.

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 profunda do funcionamento do simulador e das suas linguagens de programação são pressupostos essenciais para a utilização de algumas partes não documentadas dessa interface.

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: *ftp*, *telnet*, servidor e cliente de http, etc. É de referir a existência duma aplicação, capaz de gerar tráfego com débito constante (cbr — *Constant Bit Rate*), 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 fiável que inclui confirmações e eventuais retransmissões, 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 protocolar TCP/IP. Na verdade, existe apenas uma igualdade aproximada, mas parece ser obviamente vantajoso utilizar os mesmo nomes e os mesmo acrónimos, de forma a identificar rapidamente as funcionalidades dos protocolos [2].

Geralmente, para concretizar os protocolos existentes é usado um mecanismo genérico com funcionamento autónomo — 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.

[2] Note-se que os protocolos existentes no ns-2 não se limitam aos que aqui foram enumerados.

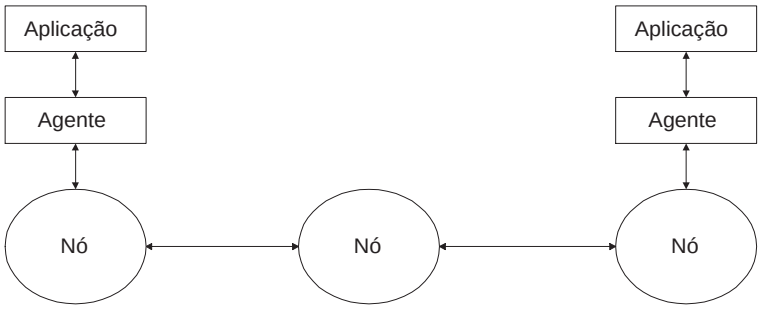


Figura A.1: Elementos constituintes duma rede típica no ns-2

A figura A.1 procura ilustrar a posição de cada uma das partes descritas até aqui e respectivas interacções.

A.1.3 Escalonador

O ns-2 é um simulador de eventos discretos, i.e., 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 autónomo 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 escalonador*, 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 activação 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 desactivação ou activação 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, i.e., os resultados duma determinada simulação terão que ser precisamente iguais (ignorando factores 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 interacção entre os objectos existentes nas duas linguagens.

Na prática, a fronteira entre o C++ e o OTcl (i.e., 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 objectos: uma delas compilada e outra interpretada, embora muitos dos objectos existentes nessas classes sejam os mesmos. Ou seja, a maioria dos objectos 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 objectos são definidos com estas duas metades [4].

[3] Tcl orientado aos objectos.

[4] Os nós, por exemplo, são uma importante excepção a esta regra, uma vez que só têm existência na classe hierárquica interpretada, embora contenham objectos com existência dual.

É importante compreender que muitos destes objectos 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 falámos.

A.2 Componentes da Topologia

A.2.1 Nó *unicast* (Node)

Um nó é um objecto que só está definido em OTcl, i.e., é um objecto que não existe na hierarquia das classes compiladas do simulador. Contém, como parte integrante, os seguintes objectos, que serão estudados mais adiante: um classificador de endereços e um classificador de portos (estes dois objectos são normalmente definidos exclusivamente em C++, embora isso não impeça que seja possível criar objectos desse tipo em OTcl). Além destes dois objectos classificadores, possui algumas variáveis, que armazenam valores que dizem respeito ao nó em questão, como o seu endereço e a sua identificação, o nome dos objectos 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 objectos 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 correcto (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 exactamente 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

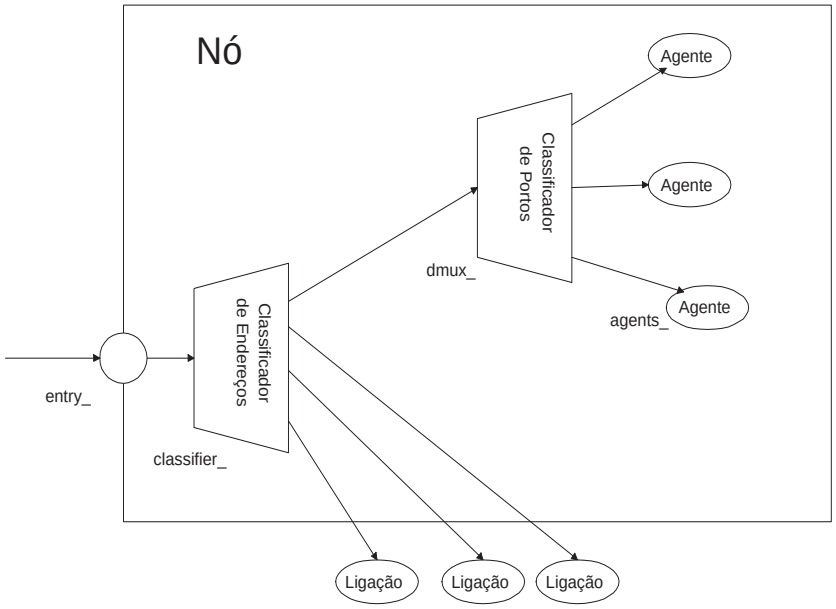


Figura A.2: Elementos constituintes dum nó

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.

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 género:

```
$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 objecto 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 facto, a instalação.

[5] Os objectos OTcl também têm um construtor, à semelhança do que acontece no C++. Sempre que é criado um novo objecto com a instrução `new` é invocado o construtor para esse objecto. Note-se que o objecto 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`, `node_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.

O objecto `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 arquitectura 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 objectos do simulador têm partes escritas em C++ e outras em OTcl — depende dos 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 objecto `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 facto 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 detalhes 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 (*Links*)

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 objectos: uma fila, um atraso que representa a ligação física propriamente dita, um verificador de TTL [6] e opcionalmente, um número arbitrário de objectos de monitorização [7].

Em termos gráficos, a configuração mínima numa ligação física está representada

[6] TTL — *Time To Live*, que em inglês significa *Tempo De Vida*

[7] Este tipo de objectos permite controlar por exemplo, grandezas relativas a fluxos de dados, pacotes descartados, etc..

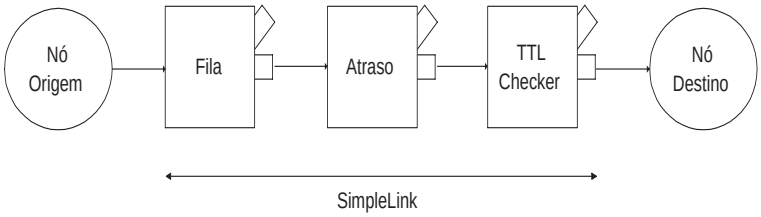


Figura A.3: Elementos constituintes duma ligação física (*link*)

na figura A.3. Da figura A.3 transparece a seguinte sequência no percurso efectuado 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 *checker*, 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 nalgum dos elementos anteriores.

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 seleccionada no momento da criação da ligação física;
- todos os objectos 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 objectos 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 objectos representados na figura A.3 (excepto os nós, obviamente) estão

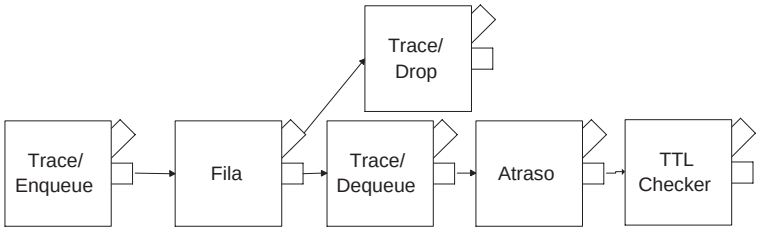


Figura A.4: Ligação física com monitores incluídos

todos incluídos num objecto único do tipo `SimpleLink` (ou de tipo derivado) que é definido em OTcl.

No manual do ns-2 são abordados os métodos que permitem adicionar objectos de monitorização e de verificação às operações feitas em toda a ligação. A figura A.4 representa uma ligação com alguns desses objectos já incluídos. Neste contexto, "interface de saída" de uma ligação será, geralmente, o nó de destino, enquanto que "interface de entrada" será o primeiro objecto existente na cadeia.

Resulta disto, que os classificadores de endereços dos nós não enviam os pacotes directamente 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 objecto adicional no início da ligação. Se o nó e a ligação já estiverem conectados, um objecto 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, i.e., 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 objectos 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 objectos que temos vindo a mencionar e respectivas classes onde são definidos, destacamos duas classes chave na arquitectura 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 arquitectura da rede que se quer construir. Com efeito, comecemos por olhar para os grandes blocos constituintes duma rede e para os objectos que os compõem:

- nó: classificador de endereços e classificador de portos — ambos classificadores;
- ligação física: fila, objectos 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, i.e., 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 projectista 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 objecto em questão deva receber um pacote.

A ideia é, quando, nalgum 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 facto de a jusante dum classificador poder haver múltiplos objectos como aliás dever ser evidente da figura A.2.

Note-se que as classes `Classifier` e `Connector` são classes básicas abstractas a partir das quais devem ser construídas classes específicas para os objectos concretos a introduzir na rede.

A.3.1 Conector

Começamos por descrever o objecto 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 objecto 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 objecto;
- `send()`: usado para enviar um pacote pela saída normal (`target_`);
- `target_` e `drop_`: objectos 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 objecto 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 correcta 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 objecto `Connector`.

No contexto desta classe, *slot* designa uma posição dentro de um vector onde são armazenadas as saídas para os objectos 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 objecto 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 objecto a jusante para onde deverá ser enviado o pacote `p`;
- `install(slot, objecto)`: instala o objecto `objecto` no *slot* `slot`;
- `clear()`: limpa um *slot*.

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-

[8] Note-se que o classificador é uma mera passagem para o pacote, i.e., um pacote que entra é, imediatamente, reencaminhado para uma das saídas.

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);
}
```

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 factores: 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 *fid*).

A primeira classe de classificadores chama-se de endereços e já temos vindo a mencioná-la múltiplas vezes [9]. A segunda classe de classificadores, chamados de classificadores de *hash*, permite efectuar classificação com base nos seguintes elementos:

[9] Também é usada para o classificador de portos. Para isso são alteradas a máscara e o deslocamento.

- origem, destino e *fid*;
- origem e destino;
- *fid*;
- 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 *hash*).

Note-se que, em todos os tipos de classificadores, a situação normal será ter na posição (*slot*), 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 objecto colocado no *slot* correspondente ao endereço do próprio nó é o classificador de portos.

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 objectos 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 (`lookup{}`), 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 efectuado com `lookup{}` que é 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 objecto 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 objectos desta classe. Mais exactamente, existe um objecto em cada nó que utiliza um protocolo de encaminhamento dinâmico.

[10] É possível usar um protocolo baseado em vectores de distâncias (DV — *Distance Vector* ou nos estados das ligações (LS — Link State).

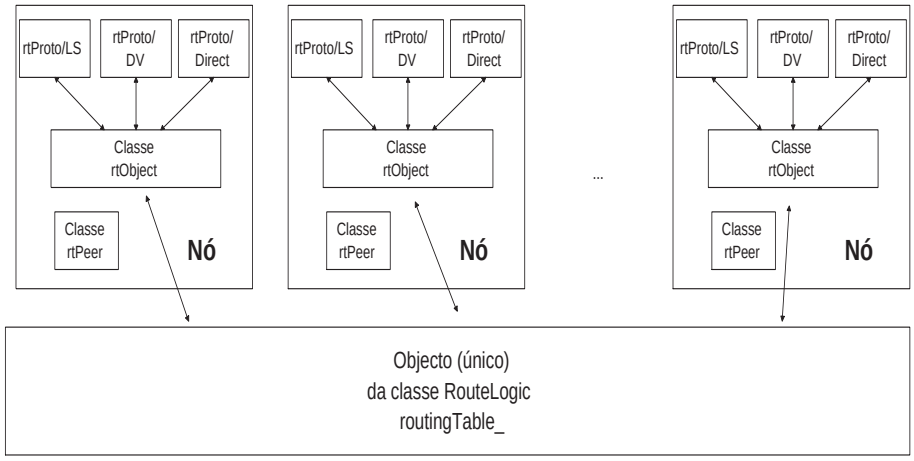


Figura A.5: Objectos envolvidos no encaminhamento

Outras classes

Além destas duas classes, existem mais duas, que são relevantes para a compreensão do código relativo ao encaminhamento:

- classe `rtPeer`: existe um objecto deste tipo em cada nó. Guarda informação sobre os vizinhos directos 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 objecto 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 objectos existentes nestas classes. A classe `rtProto/Direct` será apresentada mais adiante.

A.4.2 Criação dos Objectos Necessários ao Encaminhamento

O momento em que os diversos objectos apresentados na figura A.5 são criados difere.

O primeiro objecto a ser criado é a `routingTable_`, que é um objecto da classe `RouteLogic`, incluído na classe `Simulator`. Este objecto 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 `routingTable_` é feita por um procedimento chamado `get-routelogic{}` 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 `configure{}` da classe `RouteLogic` (também no procedimento `Simulator run`), que percorre todo o vector `rtprotos_`, se este existir, e invoca para cada protocolo que indexa este vector, o procedimento estático de inicialização respectivo, chamado `init-all{}`. Isto 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 vector `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 agen-

te. Este procedimento começa por invocar o procedimento estático homónimo da classe `rtObject`, que para cada um dos nós fornecidos em `args` vai instalar um objecto 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-all{}`.

Terminado o `init-all{}` do `rtObject`, o `init-all{}` do agente vai criar um agente em cada um dos nós da lista, com o procedimento `add-proto{}` do `rtObject`, 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 ficheiro 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 register ...`	adiciona o protocolo de nome (`nome`) ao vector `rtprotos_`, de forma que `rtprotos_(nome)` = lista dos nós onde deve correr o protocolo `nome`.

Tabela A.1: Criação dos objectos de encaminhamento — fase de configuração

`RouteLogic configure`	`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 $nodes-list; [$node rtObject?] add-proto nome $node` Chama o procedimento `init-all{}` da classe `rtObject`. Para cada nó adiciona um objecto do protocolo designado por `nome`.
`rtObject proc init-all args`	Instala em cada nó (presente na lista `args`) um novo objecto `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 objectos de encaminhamento — fase de execução

A.4.3 Inicialização

Depois de criados todos os objectos 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, exceptuando 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 *Direct*, 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 já 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 objecto `rtObject`. Estes agentes conhecem sempre o caminho para os vizinhos directos do nó em que são instalados, pelo que o nó apresenta sempre conectividade, pelo menos, com os seus vizinhos. Este facto é essencial, para que protocolos de encaminhamento

distribuídos como são o DV e o LS possam iniciar os procedimentos necessários ao seu funcionamento, que passam por enviar mensagens para os seus vizinhos directos.

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/Direct*. 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 efectuar 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-routes{}`. Os resultados da execução desses cálculos são armazenados em quatro vectores, existentes nos objectos da classe `rtObject`, 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 vectores iguais a estes com as suas próprias informações. A informação destes vectores é introduzida nos procedimentos onde os agentes de cada protocolo efectuam os seus próprios cálculos. Sendo assim, o objecto `rtObject`, no procedimento `compute-routes`{}, vai percorrer os vectores de todos os protocolos em busca do que oferece as condições mais favoráveis de encaminhamento para um determinado destino.

Seleccionado o protocolo que oferece essas condições, preenche as entradas para esse destino nos seus próprios vectores copiando, para isso, os valores presentes nos vectores homónimos do agente vencedor. O único vector que é preenchido de forma diferente é o `rtVia_`, 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-routes`{}, o nó vizinho a utilizar para atingir esse destino é armazenado no classificador com o procedimento `Node instproc add-routes`{}. Se, por ventura, já existisse informação antiga e que entretanto tenha ficado desactualizada é utilizado primeiro o procedimento `Node instproc delete-routes`{}, 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 objecto o responsável pela gestão de mensagens e pelo cálculo dos caminhos mais curtos para todos os nós. A apresentação

[11] À data em que este documento foi escrito o seu endereço de correio electrónico era `sunmin@networks.ecse.rpi.edu`.

completa da definição da classe `LsRouting` seria de todo o interesse para a correcta 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 objecto 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 `LsRouting`.

Um agente do protocolo LS pode receber três tipos de mensagens: actualizações dos estados das ligações, topologias completas e confirmações. Neste texto vamos ignorar os detalhes relacionados com estas últimas.

Actualizações 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 exceptuando, 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 actualização; `LsRouting::sendLinkState()` — enviar uma actualização.

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 actualiza a sua BD sempre que alguma desta informação for mais recente que a informação conhecida até ao momento.

No fim de fazer a actualização 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 actualização, i.e., 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 activados em simultâneo.

Depois de o agente ser criado e anexado a um nó é iniciado um vector 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 `$self compute-routes`; • Envia mensagens que ficaram no tampão, no protocolo LS.
`Agent/rtProto/LS instproc intf-changed`	• Actualiza estado das interfaces armazenado no agente (necessário para construir um mapa interno da topologia); • chama função `command()` do C++ para chamar `intfChanged()`.
`rtProtoLS::command()`	Chama `intfChanged()`.
`rtProtoLS::intfChanged()`	• Obtém o estado das interfaces; • envia o estado da interface para os vizinhos com `routing.linkStateChanged();` • chama `route-changed{}` do OTcl.
`Agent/rtProto/LS instproc route-changed`	• `$self install-routes` (ver a seguir); • `$rtObject_ compute-routes` (já 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 vectores `nextHop_`, `metric_` e `rtpref_`; • para determinar o `nextHop_` usa o procedimento `lookup{}`.

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
}
```

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 Service*). 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 adicionámos ao ns-2, de forma a conseguir a transformação desejada.

B.1 Modelo do Nó

O modelo actual 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 adicionámos à arquitectura tradicional do ns-2:

- `ConnectedClassifier`: este é um novo tipo especial de classificador que faz uma separação de pacotes conforme o seu tipo;
- agente `Setuper`: 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 objectos chamados `SimpleLink`'s, mas com informação adicional de QoS (`QoSLink`'s);

- `QoSLink`: não fazendo parte do nó, este objecto é uma extensão aos `SimpleLink`'s normais. Numa fase posterior do trabalho deveria ser transformado numa classe autónoma, 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 efectuadas 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 subóptimas.

A interacção 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 já, interessa saber que há uma *flag* no pacote que indica se ele foi ou não gerado pelo agente local;

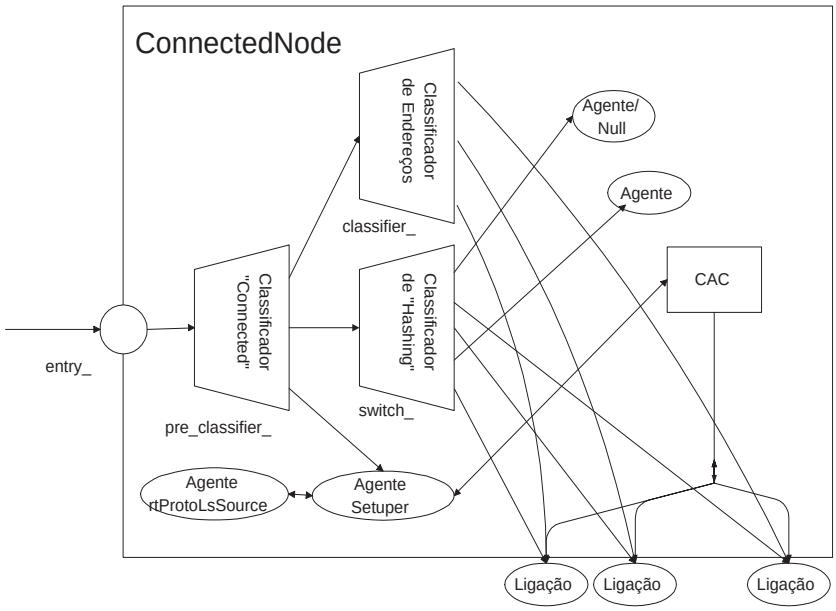


Figura B.1: Estrutura interna dum nó ligado (`ConnectedNode`)

- 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 (i.e., 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 *fid*, 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 factos, 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 (`ConnectedNode`) `A`, com destino ao agente `Y`, no nó (`ConnectedNode`) `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 objecto `A`, que pertence à classe `ConnectedNode`;
- a instrução tem que ser executada já durante a execução do simulador, i.e., não pode ser executada antes do procedimento `run` do objecto da classe `Simulator`. 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()`;
- o agente `Setuper` invoca o agente `rtProtoLsSource` para obter um caminho até ao destino (isso é feito dentro do procedimento `Agent/Setuper get-source-route`{});
- o agente `Setuper`, conhecendo já a ligação física de saída para o primeiro salto, invoca o CAC (procedimento `Agent/Setuper CAC`{}) 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 (*link*) 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 efectuar 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 será dada por concluída quando o agente receber um pacote de confirmação ou de rejeição. É importante referir que a ligação só pode ser confirmada pelo último nó, mas

[1] Aqui poderíamos ter usado a palavra “conexão”.

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 acções:

- 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 actualizar quaisquer estruturas internas, uma vez que o agente receptor já 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 actualização 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 desactualizada;
- 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 já 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 já 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 já 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 arquitectura que temos vindo a descrever.

Este agente tem métodos e variáveis definidas quer em C++, quer em OTcl, dos quais 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-path{} 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 escrita 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 referir 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 desaparecer 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 (*link*) é iniciada/terminada. Além desta ocasião, a informação continua a ser trocada periodicamente.

[2] Neste contexto "ligação" significa "conexão".

`rtProtoLsSource-::intfChanged()`	Esta função vai actualizar 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 já 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-threshold{}` do agente `Setuper`). É esta função que permite manter as estruturas internas sempre actualizadas apesar das ligações que vão surgindo/terminando.
`rtProtoLsSource-::initialize()`	Diferente da função homónima 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 actualizada dos estado das ligações, quando aparece ou termina uma nova ligação (*conexão*). Eventualmente, o envio tem que ser adiado. Ver procedimento `send-periodic-update{}`.
`A./rt./LS/S. instproc send--periodic-update`	Envia actualizações 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 actualização, não podem haver duas actualizações 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.

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.

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}, t) \\ \mathbf{x}(t_0) &= \mathbf{a} \end{aligned} \tag{C.1}$$

$\mathbf{x}$ e $\mathbf{a}$ são vectores n-dimensionais e $\mathbf{f}$ é uma função de $\mathbb{R} \times \mathbb{R}^n \rightarrow \mathbb{R}^n$, possivelmente não linear. Estes vectores 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.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}, \mathbf{f} = \begin{bmatrix} f_1(t, x_1, x_2, \ldots, x_3) \\ f_2(t, x_1, x_2, \ldots, x_3) \\ \vdots \\ f_n(t, x_1, x_2, \ldots, x_3) \end{bmatrix} \tag{C.2}$$

O método de Runge-Kutta é iterativo e gera um conjunto de valores aproximados para $\mathbf{x}(t)$, $\mathbf{x_1}$, $\mathbf{x_2}$, $\mathbf{x_3}$, ..., 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 $\mathbf{x_n}$ no instante t_n, $\mathbf{x_{n+1}}$ e t_{n+1} são calculados de acordo com a expressão C.3.

$$\begin{aligned} \mathbf{x_{n+1}} &= \mathbf{x_n} + \mathbf{k} \\ t_{n+1} &= t_n + h \end{aligned} \tag{C.3}$$

O vector $\mathbf{k}$ é uma combinação linear de quatro vectores $\mathbf{k_1}$, $\mathbf{k_2}$, $\mathbf{k_3}$ e $\mathbf{k_4}$, calculados em C.4.

$$\begin{aligned} \mathbf{k_1} &= h \cdot \mathbf{f}\left(t_n, \mathbf{x_n}\right) \\ \mathbf{k_2} &= h \cdot \mathbf{f}\left(t_n + \frac{h}{2}, \mathbf{x_n} + \frac{\mathbf{k1}}{2}\right) \\ \mathbf{k_3} &= h \cdot \mathbf{f}\left(t_n + \frac{h}{2}, \mathbf{x_n} + \frac{\mathbf{k2}}{2}\right) \\ \mathbf{k_4} &= h \cdot \mathbf{f}\left(t_n + h, \mathbf{x_n} + \mathbf{k3}\right) \\ \mathbf{k} &= \frac{1}{6}\left(\mathbf{k_1} + 2 \cdot \mathbf{k_2} + 2 \cdot \mathbf{k_3} + \mathbf{k_4}\right) \end{aligned} \tag{C.4}$$

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 correcta de o utilizar seria:

```
AK entradaAK
```

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 * 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;
```

Figura C.1: Código para simulação da RN de Ali e Kamoun

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));
  atividade = (t_float *) malloc(dim * dim * sizeof(t_float));
  if (!saida || !atividade)
     erro("Falta memoria\n");
  for (X = 0; X < dim; X++) {
     for (i = 0; i < dim; i++) {
        res = *(atividade + 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

Bibliografia

ALI, MUSTAFA K. MEHMET & KAMOUN, FAOUZI. 1993. Neural Networks for Shortest Path Computation and Routing in Computer Networks. *IEEE Transactions on Neural Networks*, **4**(5), 941–953.

ARGON, E. & ARROWPOINT, R. & RAJAGOPALAN, B. 1998 (August). *A Framework for QoS-based Routing in the Internet*. Relatório Técnico. NEC, H. Sandick, Bay Networks.

BELLMAN, R.E. 1957. *Dynamic Programming*. Princeton, N.J.: Princeton University Press.

BRADEN, ED.R. & ZHANG, L. & BERSON, S. & HERZOG, S. & JAMIN, S. 1997 (September). *Resource ReSerVation Protocol (RSVP) — Version 1 Functional Specification*. Relatório Técnico. ISI, UCLA, IBM Reserach, Univ. of Michigan.

DAYHOFF, JUDITH. 1996. *Neural Network Architectures — An Introduction*. International Thomson Computer Press.

FALL, KEVIN & VARADHAN, KANNAN. 1999. ns *Notes and Documentation*.

G.APOSTOLOPOULOS & GUERIN, R. & KAMAT, S. & ORDA, A. & PRZYGIENDA, T. & WILLIAMS, D. 1998 (April). *QoS Routing Mechanisms and OSPF Extensions*. Relatório Técnico. IBM, UPenn, IBM, Technion, Lucent, IBM. draft-guerin-qos-routing-ospf-05.txt, trabalho em evolução.

GOTO, YUKINORI & OHTA, MASATAKA & ARAKI, KEIJIRO. 1997 (June). *Path QoS Collection for Stable Hop-by-Hop QoS Routing*. Relatório Técnico. Kyushu University, Tokyo Institute of Technology.

HAYKIN, SIMON. 1994. *Neural Networks — A Comprehensive Foundation*. Macmillan College Publishing Company, Inc.

HEDRICK. 1988 (June). *Routing Information Protocol, RFC-1058*. Relatório Técnico. Rutgers University.

HOPFIELD, J.J. & TANK, D.W. 1986. "Neural" Computation of Decisions in Optimization Problems. *Biological Cybernetics*, 533–541.

HOPFIELD, J.J. 1982. Neural Networks and Physical Systems with Emergent Collective Computational Abilities. *Páginas 2554–2558 de: Proceedings of the National Academy of Sciences of the U.S.A.*

HUITEMA, CHRISTIAN. 1995. *Routing in The Internet*. New Jersey: Prentice Hall PTR.

JR., L.R. FORD & FULKERSON, D.R. 1962. *Flows in Networks*. Princeton, N.J.: Princeton University Press.

KLEINROCK, L. 1964. *Communication Nets: Stochastic Message Flow and Delay*. New York: McGraw-Hill. Reimprimido por Dover Publication, New York, 1972.

KNUTH, DONALD. 1994. *The Stanford GraphBase: A Platform for Combinatorial Computing*. Addison-Wesley.

LAUBACH, M. 1994 (January). *Classical IP and ARP over ATM*. Relatório Técnico. Hewlett-Packard Laboratories.

MCCULLOCH, W. S. & PITTS, W. 1943. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 115–133.

MOY, J. 1994a (March). *Multicast Extensions to OSPF*. Relatório Técnico. Proteon, Inc.

MOY, J. 1994b (March). *OSPF Version 2*. Relatório Técnico. Proteon, Inc.

OGATA, KATSUHIKO. 1970. *Modern Control Engineering*. Englewood Cliffs, N.J.: Prentice-Hall.

PARK, DONG-CHUL & CHOI, SEUNG-EOK. 1998. A Neural Network Based Multi-Destination Routing Algorithm for Communication Network. *IEEE*, 1673–1678.

RAUCH, HERBERT E. & WINARSKE, THEO. 1988. Neural Networks for Routing Communication Traffic. *IEEE Control Systems Magazine*, Abril, 26–31.

SCHARWTZ, MISCHA. 1987. *Telecommunication Networks*. Addison-Wesley Publishing Company.

WEISS, M.A. 1993. *Data Structures and Algorithm Analysis in C*. The Benjamin/Cummings Pub. Co. Inc.

ZHANG, Z. & SANCHEZ, C. & SALKEWICZ, B. & CRAWLEY, E. 1997 (September). *Quality of Service Extensions to OSPF or Quality of Service Path First Routing*. Relatório Técnico. Bay Networks, Avici Systems, Redback Networks, Gigapacket Networks. draft-zhang-qos-ospf-01, trabalho em evolução.