UNIVERSIDADE DE BRASÍLIA - UNB
FACULDADE DE TECNOLOGIA - FT
DEPARTAMENTO DE ENGENHARIA ELÉTRICA - ENE

APLICAÇÃO DE APRENDIZADO POR REFORÇO NA ESCOLHA INTELIGENTE DE TÚNEIS ESTABELECIDOS POR MEIO DE CAMINHOS LSP/MPLS-TE

DAYAL MACHADO BRITO

ORIENTADOR: PAULO HENRIQUE PORTELA DE CARVALHO

DISSERTAÇÃO DE MESTRADO EM ENGENHARIA ELÉTRICA

BRASÍLIA/DF: AGOSTO – 2013

UNIVERSIDADE DE BRASÍLIA - UNB
FACULDADE DE TECNOLOGIA - FT
DEPARTAMENTO DE ENGENHARIA ELÉTRICA - ENE

APLICAÇÃO DE APRENDIZADO POR REFORÇO NA ESCOLHA INTELIGENTE DE TÚNEIS ESTABALECIDOS POR MEIO DE CAMINHOS LSP/MPLS-TE

DAYAL MACHADO BRITO

DISSERTAÇÃO DE MESTRADO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA ELÉTRICA.

Aprovada por:

_____

Prof. Paulo Henrique Portela de Carvalho, Dr. (ENE-UnB)

(Orientador)

_____

Prof. Paulo Roberto de Lira Gondim, Dr. (ENE-UnB)

(Examinador Interno)

_____

Profa. Priscila América Solís Mendez Barreto, Dra. (CIC-UnB)

(Examinador Externo)

_____

Prof. Marcelo Menezes de Carvalho, Dr. (ENE-UnB)

(Suplente)

Brasília-DF, agosto de 2013

FICHA CATALOGRÁFICA

BRITO, DAYAL MACHADO

APLICAÇÃO DE APRENDIZADO POR REFORÇO NA ESCOLHA INTELIGENTE DE TÚNEIS ESTABALECIDOS POR MEIO DE CAMINHOS LSP/MPLS-TE [Distrito Federal] 2013.

xvii, 99p., 210 x 297 mm (ENE/FT/UnB, Mestre, Dissertação de Mestrado – Universidade de Brasília. Faculdade de Tecnologia.

Departamento de Engenharia Elétrica

1.Engenharia de tráfego 2.Aprendizado por reforço

3.MPLS-TE 4.OMNeT++

I. ENE/FT/UnB II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

BRITO., D. M. (2013). APLICAÇÃO DE APRENDIZADO POR REFORÇO NA ESCOLHA INTELIGENTE DE TÚNEIS ESTABALECIDOS POR MEIO DE CAMINHOS LSP/MPLS-TE. Dissertação de Mestrado em Engenharia Elétrica, Publicação PPGEE.DM-542/2013, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 99p.

CESSÃO DE DIREITOS

AUTOR: Dayal Machado Brito.

TÍTULO: APLICAÇÃO DE APRENDIZADO POR REFORÇO NA ESCOLHA INTELIGENTE DE TÚNEIS ESTABALECIDOS POR MEIO DE CAMINHOS LSP/MPLS-TE.

GRAU: Mestre ANO: 2013

_____

Dayal Machado Brito

SQS 210, Bloco I, apto. 603. Asa Sul.

70.273-090 Brasília – DF – Brasil.

AGRADECIMENTOS

Agradeço a Deus pela saúde e por ter tido condições de chegar ao final do trabalho.

À minha mãe, à minha avó, ao meu padrinho Maurício pela ajuda, encorajamento e amorosa preocupação. À Alessandra pelo amor, apoio e compreensão. A todos os amigos do REMA, sempre ao meu lado. Ao meu pai pelo constante interesse.

Aos professores Márcio Augusto de Deus e João Paulo Leite, sempre disponíveis e solícitos, que atuaram como fundamentais co-orientadores deste trabalho. Ao professor Paulo Henrique Portela de Carvalho por ter acreditado no trabalho até sua concretização.

Aos meus chefes no Ministério das Comunicações, Frederico de Oliveira e Denise de Oliveira, que deram o necessário incentivo para a realização do trabalho.

A todos que, de alguma forma, me ajudaram na construção de mais esta etapa.

RESUMO

APLICAÇÃO DE APRENDIZADO POR REFORÇO NA ESCOLHA INTELIGENTE DE TÚNEIS ESTABALECIDOS POR MEIO DE CAMINHOS LSP/MPLS-TE

Autor: Dayal Machado Brito

Orientador: Paulo Henrique Portela de Carvalho

Programa de Pós-graduação em Engenharia Elétrica

Brasília, agosto de 2013

O presente trabalho se propõe a explorar técnicas que deem um caráter mais dinâmico e inteligente aos protocolos de roteamento ou engenharia de tráfego que existem e são amplamente utilizados no mercado, em razão de seu comportamento um tanto quanto estático frente a mudanças nas condições da rede. Geralmente é necessária a intervenção de um operador humano, o que nem sempre ocorre de forma rápida ou eficaz. Portanto, constitui-se um objeto de estudo válido o acréscimo de inteligência artificial a esses sistemas de modo a avançar nesse sentido. Este trabalho teve como objetivo aplicar um algoritmo de aprendizado por reforço ao protocolo RSVP-TE de modo a permitir a escolha dinâmica e automática do caminho com menor atraso até o roteador de borda de saída da sessão, dentre um grupo de caminhos pré-configurados explicitamente. A ferramenta desenvolvida foi testada em ambiente de simulação e comparada com um algoritmo clássico de roteamento adaptativo, com bons resultados. Mostrou-se que a solução é válida e promissora, podendo passar por ajuste fino de parâmetros para que se consiga um desempenho otimizado e condizente com cada cenário.

ABSTRACT

APLICAÇÃO DE APRENDIZADO POR REFORÇO NA ESCOLHA INTELIGENTE DE TÚNEIS ESTABALECIDOS POR MEIO DE CAMINHOS LSP/MPLS-TE

Author: Dayal Machado Brito

Supervisor: Paulo Henrique Portela de Carvalho

Programa de Pós-graduação em Engenharia Elétrica

Brasília, August 2013

The present work has the goal to explore techniques that may add adaptive and intelligent characteristics to the existing routing protocols and traffic engineering tools of today's commercial networks. As these protocols don't generally respond well to changes in the network's performance and quality of service, the intervention of an engineer is often needed. It may not happen, however, in the fastest or most efficient way possible. Thus an interesting subject rises to be studied: the adding of Artificial Intelligence in today's protocols to reach the aimed dynamic response. It was intended in this work to apply a reinforcement learning algorithm to the RSVP-TE protocol to implement a scheme that allows the automatic and dynamic choice of the best path to the edge router at the end of the session, among a pre-configured group of paths. The end-to-end delay in each path is the considered performance parameter. The solution was tested and presented good results, remaining a promising tool for problems of traffic engineering after a fine adjustment of the parameters involved.

SUMÁRIO

Lista de Figuras

Lista de Tabelas

Lista de Símbolos, Nomenclaturas e Abreviações

AS	*Autonomous System*
BGP	*Border Gateway Protocol*
CA-OSPF	*Cost Adaptive* OSPF
CIDR	*Classless InterDomain Routing*
CM	*Composite Metric*
CSPF	*Constrained Shortest Path First*
EGP	*Exterior Gateway Protocol*
FEC	*Forward Equivalence Class*
IETF	*Internet Engineering Task Force*
IGP	*Interior Gateway Protocol*
INET	Pacote de simulação de redes de comunicação do OMNeT++
IP	*Internet Protocol*
IS-IS	*Intermediate System-Intermediate System*
ISO	*International Standards Organization*
kbps	Quilobit por segundo
LER	*Label Edge Router*
LIB	*Label Information Base*
LMS	*Least Mean Squares*
LSA	*Link State Acknowledge*
LSP	*Label Switched Path*
LSR	*Label Switch Router*
Mbps	Megabit por segundo
MPLS	*Multi Protocol Label Switching*
MPLS-TE	MPLS *Traffic Engineering*
NED	*Network Description language* (OMNeT++)
OFDM	*Orthogonal Frequency-Division Multiplexing*
OMNeT++	*Framework* de simulação de redes
OSI	*Open Systems Interconnection*
OSPF	*Open Shortest Path First*
PDM	Processo de Decisão de Markov

PPP	*Point-to-Point Protocol*
$Q(s, a)$	Retorno esperado para o estado s depois de escolhida a ação a (aprendizado por reforço)
QoS	*Quality of Service*
r	Recompensa, ganho ou retorno (aprendizado por reforço)
RFC	*Request For Comments*
RIP	*Routing Information Protocol*
RL-RSVP-TE	*Reinforcement Learning* RSVP-TE
RSpec	*Resource Specification*
RSVP	*Resource Reservation Protocol*
RSVP-TE	RSVP *Traffic Engineering*
SARSA	*State-Action-Reward-State-Action*
TCP	*Transmission Control Protocol*
TCP/IP	Modelo de referência para representação de redes nomeado por dois de seus principais protocolos, TCP e IP
TD	*Temporal Difference*
TSpec	*Traffic Specification*
UDP	*User Datagram Protocol*
VPN	*Virtual Private Network*
α	Taxa de aprendizado (aprendizado por reforço)
γ	Fator de desconto (aprendizado por reforço)

1 INTRODUÇÃO

Atualmente, presencia-se um aumento cada vez maior do número de acessos à Internet no Brasil e no mundo. À medida que os avanços da tecnologia vão permitindo maiores taxas de transmissão, a demanda também aumenta, tanto pela crescente quantidade de usuários como por maior qualidade do conteúdo. Assim, quando os recursos disponíveis na rede não são suficientes para atender a todos os usuários com a mesma qualidade, os requisitos de QoS (*Quality of Service*), mais rigorosos, têm de ser respeitados a fim de se possibilitar preços mais competitivos e justos [1].

Embora estabelecidos no mercado, os protocolos comerciais de roteamento e engenharia de tráfego mais amplamente utilizados na Internet ainda possuem limitações pouco exploradas. Os protocolos IGP (*Interior Gateway Protocols*), na maioria das vezes não conseguem tomar decisões de maneira a adaptar rotas dos serviços à situação atual da rede, o que exige a configuração manual de algum engenheiro de tráfego. Um dos principais objetivos deste trabalho é explorar possibilidades de melhor utilização dinâmica dos recursos da rede de acordo com as variações de seus parâmetros de QoS.

Adicionar inteligência ao sistema pode ajudar no objetivo de substituir e otimizar as ações de um operador da rede, caso haja deterioração das condições desta. O sistema deve poder ajustar parâmetros da rede de forma autônoma e dinâmica. Se um sistema situado em um ambiente sujeito a alterações tem habilidade de aprender e se adaptar a elas, pode ser considerado inteligente. Desta forma, diminui-se sua dependência a um operador humano para contornar mudanças, que além de tudo também está sujeito a cometer erros. A inteligência artificial é uma área a ser explorada como possibilidade de suporte a este tipo de necessidade, com isto melhorando a efetividade em termos de estabilidade e ganhos operacionais. Algoritmos desse tipo vêm sendo aplicados com sucesso em diversos campos, de reconhecimento facial à aviação, passando pela medicina [2].

1.1 DEFINIÇÃO DO PROBLEMA

Fazer engenharia de tráfego é organizar o tráfego, por meio de análise ou predição, adequando-o a uma rede existente de forma a evitar congestionamentos e buscar atendimento a certos critérios de QoS [3]. O que se propõe atacar neste trabalho é um problema específico relacionado à engenharia de tráfego: um provedor de serviços de dados precisa realizar a entrega do tráfego de um grande cliente de forma satisfatória, ou seja, deve ser escolhido um caminho para os pacotes, dentro do sistema autônomo (intra-AS), que forneça a melhor qualidade de serviço possível.

Uma solução usual nas redes comerciais é a utilização de túneis fim a fim por meio de algum protocolo de engenharia de tráfego, de modo a separar o tráfego acrescentando a ele alguma marcação. Isso permite que o engenheiro de tráfego tenha um controle maior sobre os processos automatizados pelos protocolos de roteamento tradicionais. Neste caso os túneis devem ser muito bem dimensionados para que não ocorram surpresas que possam causar instabilidade na rede. Além disso, os túneis possuem caráter fixo, permanecendo com a mesma configuração mesmo que algum congestionamento na rede venha a trazer degradação em parâmetros de QoS.

Outra solução relativamente comum no dia-a-dia das redes é a adaptação dos custos dos enlaces por parte do engenheiro de tráfego. Após análise, se constatada alguma piora em parâmetros de QoS de um certo enlace, pode-se aumentar o respectivo custo para que as mudanças na rede sejam absorvidas pelo protocolo de roteamento utilizado.

Outras soluções utilizadas pelo engenheiro de tráfego sempre envolvem a medição do uso da rede e o controle de seus recursos. Essas soluções são sensíveis a mudanças no estado da rede, mas dependentes de um operador humano, que normalmente não testa todas as possibilidades e teria que despender todo o seu tempo para o acompanhamento adequado do desempenho. As redes carecem de uma solução mais dinâmica, que permita a realização automática de ajustes para adequação a essas mudanças.

1.2 OBJETIVOS DO TRABALHO

Neste trabalho, deseja-se contribuir para a solução do referido problema, apresentando uma ferramenta computacional que implemente dinamicamente a atividade de um engenheiro de tráfego, ainda que de forma simplificada em um primeiro momento. Para isso, pode-se adicionar inteligência aos protocolos utilizados para realizar engenharia de tráfego já existentes. O intuito é realizar uma prova de conceito, com um sistema que permita a escolha do caminho que ofereça a melhor qualidade de serviço – de acordo com determinado parâmetro – acompanhando as mudanças na rede e se adaptando a elas. Portanto, um bom instrumento para o fim almejado é a inteligência artificial, tendo em vista suas características anteriormente citadas.

O aprendizado por reforço é um ramo da inteligência artificial que se propõe a simular em um agente computacional o modelo de aprendizado cognitivo, ou seja, aquisição de conhecimento e adaptação ao meio. De forma autônoma, o agente obtém informações do ambiente por meio de uma sequência de ações usando tentativa e erro. Pela similaridade com as necessidades do problema proposto, um algoritmo de aprendizado por reforço será utilizado para adicionar inteligência ao estabelecimento e manutenção de um túnel pelo melhor caminho para prover engenharia de tráfego. O parâmetro que será principalmente contemplado para definir o melhor caminho é o atraso fim a fim entre origem e destino. Para a implementação da solução e testes, será utilizado o simulador de redes OMNeT++ [4] e seu pacote INET Framework, que contém modelos de vários protocolos de redes e telecomunicações.

1.3 ESTRUTURA DA DISSERTAÇÃO

O Capítulo 2 terá uma revisão de conceitos básicos de rede e roteamento, com aspectos teóricos e descrição dos principais protocolos de rede utilizados no mercado. Também será realizada uma breve descrição das tecnologias que compõem a solução MPLS-TE para engenharia de tráfego.

No Capítulo 3, encontram-se conceitos teóricos relativos à técnica de aprendizado por reforço: sua modelagem matemática como Processo de Decisão de Markov e respectiva solução, bem como algoritmos clássicos e outras informações.

O Capítulo 4 traz informações de diversos trabalhos com propostas similares à deste, que desenvolveram protocolos e algoritmos adaptativos de roteamento e estabelecimento de túneis, seja com novos paradigmas, seja com modificações de protocolos já difundidos.

No Capítulo 5 apresenta-se a ferramenta OMNeT++, bem como a solução desenvolvida neste trabalho para escolha inteligente de caminhos MPLS-TE para engenharia de tráfego adaptativa. Também são mostrados resultados de simulações realizadas para testar a ferramenta, revelando sua aplicabilidade, e promover ajuste dos parâmetros.

O Capítulo 6 traz as conclusões do trabalho, bem como as considerações finais e sugestões para trabalhos futuros.

2 CAMADA DE REDE E ROTEAMENTO

2.1 INTRODUÇÃO

Como o objetivo do trabalho é o desenvolvimento de uma ferramenta para escolher dinamicamente o melhor caminho para um determinado fluxo de dados, faz-se necessária uma revisão de conceitos básicos de redes a respeito da camada de rede, na qual se encontra o tratamento do problema do roteamento. Serão descritas as funções geralmente desempenhadas no escopo dessa camada, bem como os principais protocolos utilizados na Internet atualmente. Além disso, tratar-se-á da solução MPLS-TE para prover engenharia de tráfego, que envolve a combinação entre os protocolos MPLS e RSVP-TE. Este conjunto será importante como ferramenta também para a solução implementada por este trabalho.

2.2 MODELOS DE REPRESENTAÇÃO EM CAMADAS

Os modelos geralmente utilizados para descrever as redes de comunicação são paradigmas que separam as redes em diferentes camadas dispostas umas sobre as outras e são fundamentais para reduzir a complexidade do projeto dessas redes. Cada camada deve prestar serviços à camada superior, deixando ocultos para as demais os detalhes desse serviço e sua implementação. Este conceito recebe o nome de encapsulamento de dados. Para tal, existe uma interface de comunicação entre cada camada e as de nível imediatamente superior e inferior [5].

Cada camada de nível *n* de uma máquina, por exemplo, se comunica com a camada de mesmo nível de outra máquina, seguindo um conjunto de regras de um acordo firmado entre os nós que é chamado de protocolo da camada *n*. A Figura 2.1 mostra um exemplo genérico de modelagem de uma rede por meio de cinco camadas, evidenciando as interfaces e protocolos.

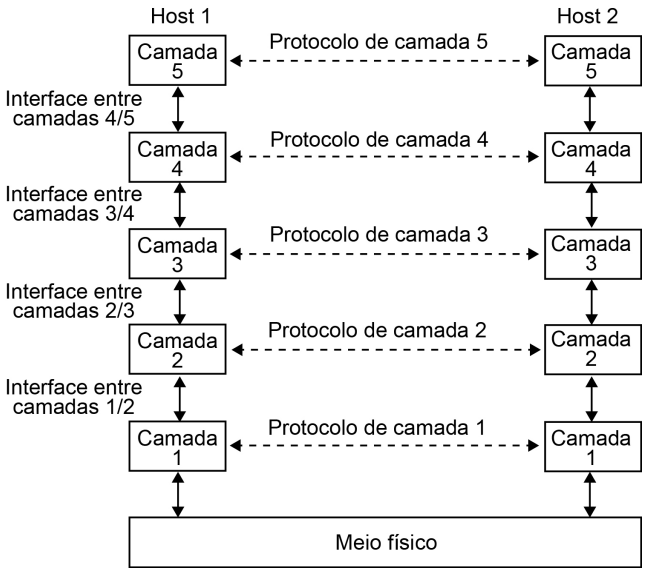

Figura 2.1 – Representação de uma rede em camadas, com protocolos e interfaces. Adaptada de [5].

Existem importantes modelos de referência para a representação de redes, como o modelo OSI (*Open Systems Interconnection*), desenvolvido pela ISO (*International Standards Organization*), que representou um avanço na padronização internacional na área. Constam do modelo OSI sete camadas com funções bem definidas, listadas abaixo da camada mais elevada até a mais baixa [5][6][7]:

- Camada 7 – Aplicação: onde são trocadas as mensagens das aplicações de interesse do usuário que são geralmente o objetivo final da comunicação.

- Camada 6 – Apresentação: responsável pela sintaxe e semântica das informações transmitidas para que máquinas com diferentes representações de dados possam se comunicar sem problemas.

- Camada 5 – Sessão: permite a criação de sessões entre máquinas que possibilitam diversos serviços relativos ao controle da comunicação.

- Camada 4 – Transporte: oferece o transporte fim a fim de mensagens das aplicações entre cliente e servidor, com a possibilidade de serviços para garantir ou

melhorar a qualidade dessa entrega. As mensagens da camada superior são encapsuladas em pacotes chamados de **segmentos**.

- Camada 3 – Rede: trata da transferência de pacotes da camada de rede de uma máquina de origem para a de destino na rede, cuidando, por exemplo, do roteamento dos pacotes e troca de possíveis mensagens de erro. Os segmentos da camada de transporte são encapsulados em **datagramas**, os pacotes cujos campos, na Internet, são definidos pelo protocolo IP (*Internet Protocol*).

- Camada 2 – Enlace: oferece à camada de rede a entrega dos pacotes para a interface de rede vizinha utilizando o canal de transmissão disponível. Os datagramas da camada de rede são encapsulados em **quadros**.

- Camada 1 – Física: encarregada da transmissão de bits brutos pelo canal de comunicação. Fazem parte desta camada questões como técnicas de modulação, codificação de linha e multiplexação.

Um modelo de referência também bastante utilizado na Internet é o TCP/IP, nomeado pelos seus dois principais protocolos, com suas quatro camadas: Aplicação, Transporte, Internet e Acesso à Rede. Ele foi criado a partir de um conjunto de protocolos específicos para descrevê-los e, por isso, não se encaixa tão bem com outros protocolos [5]. A camada de Aplicação da pilha TCP/IP é análoga à camada de mesmo nome do OSI, assim como a de Transporte. A camada Internet foi pensada para garantir que os pacotes trafeguem independentemente e cheguem corretamente ao destino, mesmo que passem por redes diferentes. Ou seja, desempenha função análoga à camada de Rede do modelo OSI. A camada de Acesso à Rede não é exatamente especificada no TCP/IP, apenas define-se que um protocolo deve ser usado para que o *host* acesse a rede e possa enviar pacotes IP. Contudo, esse protocolo varia de acordo com o tipo de *host* e de rede.

A camada de rede é responsável, entre outras funções, pelo roteamento dos datagramas, ou seja, o estabelecimento do melhor caminho que o pacote pode percorrer

até o seu destino. Trata-se de uma parte crítica para o desempenho das redes, uma vez que sua finalidade consiste em alocar da melhor forma possível os recursos disponíveis para o tráfego que se deseja entregar, de forma a otimizá-la segundo algum critério, como o tempo médio de entrega dos pacotes [8]. Por este motivo, este trabalho dará destaque especial a esse aspecto e, portanto, faz-se necessária a princípio uma descrição breve a respeito da camada de rede.

2.3 A CAMADA DE REDE NA INTERNET

Para atender a finalidade principal da camada de rede de promover o transporte de pacotes entre o nó de origem até o seu destino, duas funções fundamentais podem ser destacadas:

- O roteamento dos pacotes – a definição dos melhores caminhos a serem percorridos por meio de algoritmos alimentados por informações trocadas entre os roteadores;

- O encaminhamento dos pacotes – o tratamento dado aos pacotes por cada roteador da rede individualmente, tendo como base as rotas definidas pelo algoritmo de roteamento.

Cada roteador mantém uma **tabela de roteamento**, que armazena a interface de saída para a qual deve ser encaminhado um pacote de acordo com o seu endereço de destino. A tabela é atualizada periodicamente pelo algoritmo de roteamento escolhido para a rede.

As redes de computadores podem ser divididas em duas classes. Quando a camada de rede oferece à camada de transporte apenas um serviço orientado a conexão, tem-se uma rede de circuitos virtuais (CV), onde um caminho fixo é estabelecido entre origem e destino para o repasse dos pacotes e os roteadores têm de manter informações sobre o estado dessas conexões. Por outro lado, quando é oferecido apenas um serviço não orientado a conexão para a camada de transporte, tem-se uma rede de datagramas.

Como na Internet, interesse central deste trabalho, utiliza-se o paradigma de rede de datagramas do protocolo IP, este será objeto de estudo nesta seção.

2.3.1 O protocolo IP

O protocolo IP (*Internet Protocol*) foi desde o início projetado para interconectar as redes diversas que compõem a Internet. Ele promove a entrega de datagramas da origem para o destino por meio de um serviço sem garantias (*Best Effort*, ou da melhor maneira possível), independentemente de os *hosts* estarem na mesma rede ou não.

Um datagrama IP consiste em duas partes: um cabeçalho e uma área de dados. No cabeçalho, há uma parte de tamanho fixo de 20 bytes e uma parte opcional de tamanho variável. O formato do cabeçalho IP é mostrado na Figura 2.2 e sua descrição completa pode ser encontrada em [5] [6]. Por exemplo, o campo *Version* traz informação do número da versão do protocolo (IPv4 ou IPv6).Um campo importante é *Type of Service*, no qual se informa a classe de serviço à qual pertence o pacote. Esse campo teve algumas finalidades diferentes pelos anos, mas hoje é utilizado, por exemplo, pelo mecanismo *DiffServ* (serviços diferenciados) para garantir qualidade de serviço (QoS). No campo *Total Length*, observa-se o tamanho total em bytes do datagrama, incluindo o cabeçalho e os dados.

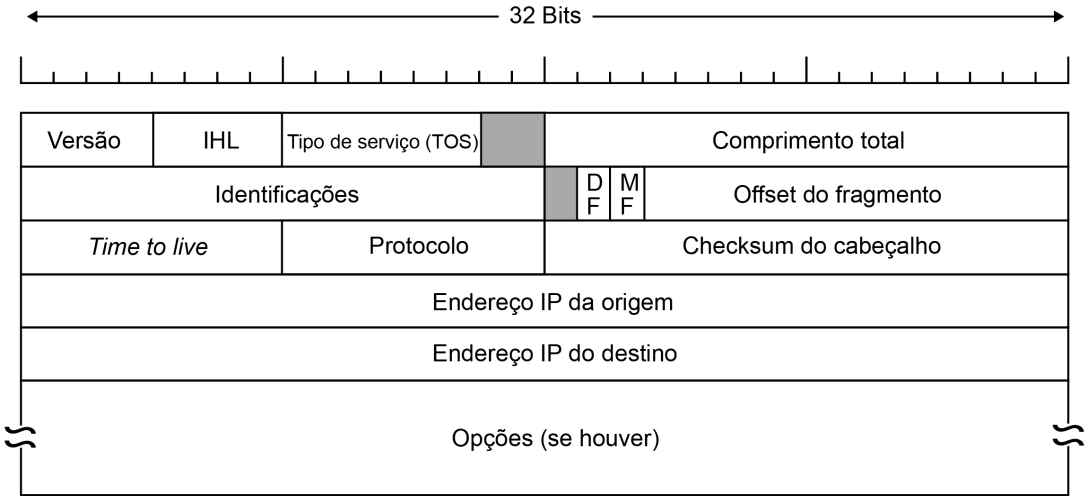

Figura 2.2 – Formato do cabeçalho IP. Adaptada de [5].

O campo *Time to Live* armazena o tempo de vida útil do pacote, com um contador que é decrementado a cada salto para outro nó. O datagrama é descartado quando esse contador chega a zero. Já o campo *Protocol* é utilizado para informar qual é o processo de transporte ao qual deve ser entregue o datagrama, por exemplo, TCP ou UDP. O campo de opções não é de uso obrigatório e permite uma ampliação do cabeçalho IP. Para verificar o conteúdo do cabeçalho e facilitar a detecção de erros, inicialmente é feita a soma de todas as meias palavras de 16 bits com aritmética de complemento de 1 e, ao final, calcula-se o complemento de 1 do resultado. O resultado final é escrito no campo *Header Checksum*.

Os campos *Source Address* e *Destination Address* contêm os endereços IP da origem e do destino, respectivamente, que são endereços únicos de 32 bits para cada interface de rede e possuem partes destinadas ao endereço da sub-rede e ao do *host*. Antigamente, sabia-se qual parte do endereço era referente à sub-rede pela classe a que pertencia o endereço, em um esquema fixo com 5 opções, de A a E, conhecido como *classful* [5]. Em razão do esgotamento inevitável de endereços IP e da rigidez desse esquema, um novo modelo e melhor solução encontrada foi o CIDR (*Classless InterDomain Routing* – RFC 1519), por meio do qual os endereços restantes são alocados, de forma não restrita a classes, em blocos de tamanho variável. Assim, um endereço é representado por ***a.b.c.d/x*** e tem os *x* bits mais significativos para indicar a parte destinada à sub-rede, formando o chamado prefixo de rede, enquanto os demais representam o endereço do *host*. Com isso, todos os endereços passaram a ser armazenados na tabela de encaminhamento juntamente com a respectiva máscara de rede. Essa tabela deve ser preenchida pelos algoritmos de roteamento após a realização de cálculos. Estes algoritmos serão o tema da próxima seção.

2.3.2 Roteamento

A Internet é formada pela interligação de vários sistemas autônomos (*Autonomous Systems* – AS's). Um AS é um conjunto de um ou mais prefixos IP mantidos por pelo menos uma operadora de rede e que possui uma única e bem definida política de roteamento [9]. E este tipo de política se destina a tratar um problema clássico que se

tem no escopo das redes: encontrar o melhor caminho entre uma origem e um determinado destino. A divisão em sistemas autônomos facilita o tratamento desse problema, permitindo que ele seja dividido em duas esferas independentes: roteamento intra-AS, ou seja, dentro de um mesmo sistema autônomo, realizado por um protocolo IGP (*Interior Gateway Protocol* – Protocolo de Roteador Interno); e roteamento inter-AS, entre sistemas autônomos distintos, realizado por um protocolo EGP (*Exterior Gateway Protocol* – Protocolo de Roteador Externo). Caso esses domínios não fossem separados, os algoritmos de roteamento teriam de lidar com um número excessivo de *hosts*, o que traria prejuízos a sua eficiência.

Os dados disponíveis para que os algoritmos tratem a tarefa de roteamento são o conjunto de roteadores e os enlaces que existem entre estes. O objetivo final é a obtenção do "melhor" caminho entre origem e destino, ou seja, do caminho que possui o menor custo. A definição de custo de um enlace em uma rede específica geralmente é determinada pelo administrador e pode assumir a figura de alguns parâmetros de qualidade de serviço, o tamanho físico dos enlaces, o custo monetário associado ao estabelecimento destes ou ainda a taxa de transmissão suportada por eles.

Uma ferramenta matemática fundamental para este tipo de assunto é a teoria de grafos, usada para a formulação do problema. Um grafo $G = (N, E)$ é um conjunto de N nós e E arestas, em que uma aresta representa um par de nós (x, y) do conjunto N. A cada uma das arestas existentes está associado um custo $c(x, y)$. O custo total de um caminho é a soma do custo individual de cada enlace. Um exemplo de grafo é mostrado na Figura 2.4, no caso com 6 nós (A, B, C, D, E e F) e 10 enlaces entre eles. O número exibido sobre o enlace representa o seu custo, por exemplo, o custo do enlace entre os nós A e C é $c(A, C) = 5$.

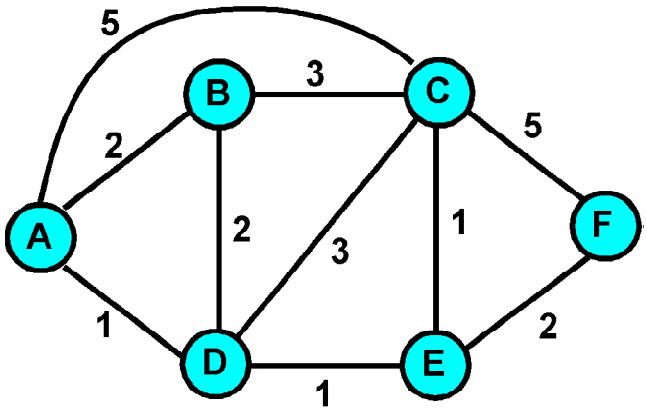

Figura 2.3 – Exemplo de grafo com 6 nós e 10 enlaces com pesos variados. Baseado em [6].

Uma vez que a rede está descrita matematicamente por meio de um grafo, os algoritmos de roteamento têm a finalidade de encontrar os melhores caminhos. Estes algoritmos podem ser classificados de acordo com diferentes aspectos, por exemplo: quanto à sensibilidade a carga, quanto à dinamicidade e quanto ao conhecimento da rede por parte dos nós.

Em algoritmos sensíveis à carga, o custo de cada enlace varia de acordo com o seu nível de congestionamento, ao passo que quando o custo do enlace é indiferente ao congestionamento, diz-se que o algoritmo não apresenta sensibilidade à carga. Se os caminhos estabelecidos pelo algoritmo mudam de acordo com mudanças no tráfego ou na topologia da rede, o algoritmo é dito dinâmico. Caso não houver nenhuma mudança nos custos em razão dos referidos fatores, o algoritmo é classificado como estático.

Quanto ao conhecimento da rede por parte dos nós, os algoritmos são classificados em globais e descentralizados. No caso de algoritmos de roteamento globais, todos os nós possuem informação completa sobre a conectividade da rede e o custo dos enlaces, como ocorre com os algoritmos de estado de enlace (*link state*). Em algoritmos descentralizados, ocorre um processo iterativo de troca de informações entre vizinhos, não havendo ciência do custo de todos os enlaces por parte de cada nó. Exemplos de algoritmos descentralizados são os algoritmos de vetor de distâncias (*distance-vector*).

Nos algoritmos de estado de enlace, para que todos os nós tenham uma visão idêntica e completa da rede, cada nó deve transmitir periodicamente por *broadcast* (ou seja, para todos os nós da rede) pacotes informando sua identidade e o custo dos enlaces a ele conectados. Um exemplo de algoritmo de estado de enlace é o de Dijkstra, um algoritmo iterativo que calcula o caminho de menor custo de um nó de referência até todos os demais da rede.

Os algoritmos do tipo vetor de distâncias são iterativos, assíncronos e distribuídos – cada nó recebe informação de seus vizinhos, realiza cálculos e envia resultados apenas aos vizinhos. Cada nó mantém uma série de informações: o custo dos enlaces de ligação com seus vizinhos $c(x, y)$; o seu vetor de distâncias D_x, que contém a estimativa do custo dos melhores caminhos até cada destino; o vetor de distância D_v de cada vizinho v. Assim, um nó repassa a seus vizinhos uma cópia de seu vetor de distâncias e utiliza o vetor dos vizinhos para atualizar sua tabela de repasse por meio da equação (2.1), conhecida como equação de Bellman-Ford:

$$D_x(y) = \min_v\{c(x, v) + D_v(y)\} \tag{2.1}$$

A equação de Bellman-Ford indica que o nó vai escolher o vizinho com a menor estimativa de custo do caminho até o destino, sendo que a estimativa é calculada pela soma entre o custo do enlace até o vizinho e a estimativa do caminho entre o referido vizinho e o destino.

2.3.3 Protocolos de roteamento na Internet

IGP

Originalmente, o protocolo IGP da Internet, ou seja, no interior de sistemas autônomos, era o RIP (*Routing Information Protocol* – Protocolo de Informação de Roteamento. O RIP é um protocolo de roteamento que tem funcionamento muito similar ao algoritmo geral de vetor de distâncias descrito anteriormente e utiliza o número de saltos como métrica para escolha do melhor caminho. Para evitar *loops*, o protocolo

limita a 15 o número de saltos até qualquer destino. Essa estratégia também acaba por restringir o uso do RIP a sistemas autônomos com 15 ou menos saltos de diâmetro. Como se trata de um algoritmo de vetor de distâncias, um nó em que roda o RIP deve trocar tabelas de roteamento com seus vizinhos a cada 30 segundos, aproximadamente, por meio de mensagens conhecidas como anúncios RIP. Se um nó não recebe notícias de um determinado vizinho por mais de 180 segundos, este vizinho passa a ser considerado como inalcançável pelo nó em questão, que atualiza sua tabela de roteamento com esse fato e divulga para os demais vizinhos por meio de anúncios. O RIP apresenta prejuízos em sua eficiência quando aplicado a sistemas autônomos grandes, além de sofrer com convergência lenta e problemas de contagem até o infinito – quando atualizações inválidas continuam circulando mesmo após a falha em um enlace, causando um *loop*.

Em razão dos problemas, o RIP foi amplamente substituído pelo OSPF (*Open Shortest Path First*), que começou a ser desenvolvido pela IETF (*Internet Engineering Task Force*) em 1988 e virou um padrão em 1990. Sua versão mais recente, a 2, foi definida no RFC 2328 [10], que é um documento aberto. Como protocolo de roteamento de estado de enlace, o OSPF utiliza o algoritmo de Dijkstra para achar os melhores caminhos, utilizando a topologia completa da rede obtida por meio de *broadcast* de informações de cada nó. Os custos de cada enlace, nesse caso, são configurados pelo administrador da rede. Como regra do fabricante Cisco [11], por exemplo, o custo C de cada enlace deve ser calculado de modo a ser inversamente proporcional à taxa de transmissão deste, tendo como referência uma taxa de 10^8 bps, ou 100 Mbps. Assim:

$$C = \frac{10^8}{taxa\ de\ transmissão\ do\ enlace\ em\ \text{bps}} \tag{2.2}$$

Atualmente, como as taxas muitas vezes são superiores a 100 Mbps, a capacidade de referência adotada na prática tem sido maior.

O OSPF descobre automaticamente os roteadores vizinhos, com o estabelecimento e manutenção do relacionamento de vizinhança entre eles. Com isso,

cada roteador sabe o estado de cada vizinho. Essa descoberta é feita por meio dos Pacotes *Hello*, que são enviados para cada interface de um roteador periodicamente, a cada *Hello Interval* segundos. Uma vez com as informações do estado de seus vizinhos, um roteador OSPF pode distribuí-las para toda a rede por meio das LSAs (*Link State Adverstisements*), unidades de dados que são carregadas por pacotes *Link State Update* e devem ser necessariamente confirmadas por meio de *Link State Acknowledgements*. Com as informações recebidas, cada roteador monta seu *Link State Database*, base de dados que descreve a topologia completa da área em questão e deve ser igual em cada roteador que nela se encontra [10]. Assim, por meio do algoritmo de Dijkstra [12], pode-se encontrar o melhor caminho a cada outro roteador da área.

Um mesmo sistema autônomo em que se usa OSPF pode ainda ser dividido em áreas menores, para que em cada uma rode um algoritmo de estado de enlace diferente. Uma área deve ser separada para ser a área de *backbone*, que tem a incumbência de rotear o tráfego entre as demais áreas. Essa capacidade de estruturar hierarquicamente o sistema autônomo é um dos maiores avanços trazidos pelo OSPF.

Outros pontos positivos do OSPF são: uma maior segurança disponível em razão da autenticação das mensagens trocadas entre os roteadores; a possibilidade de utilizar mais de um caminho com o mesmo custo total, permitindo balanceamento de carga; e suporte integrado para roteamento *multicast* [6]. O fato de no OSPF os custos dos enlaces serem estabelecidos pelo administrador e permanecerem quase sempre fixos, sem nenhuma alteração que acompanhe mudanças em parâmetros de qualidade de serviço, representa uma limitação do protocolo, que deixa de aproveitar essa possibilidade de melhoria de desempenho, como é feito em [13], trabalho que propõe uma solução em que os pesos do algoritmo de Dijkstra são alterados dinamicamente com uma composição de diversas métricas de QoS, e que obtém bons resultados.

Algumas funcionalidades do OSPF foram herdadas do protocolo IS-IS (*Intermediate System-Intermediate System*), que também implementa roteamento de estado de enlace. As diferenças entre os dois protocolos não são muitas, portanto. Uma importante vantagem do IS-IS é o fato de permitir de forma natural e relativamente

simples o transporte simultâneo de informações sobre diferentes protocolos da camada de rede [5].

EGP

O protocolo EGP padrão da Internet, ou seja, para estabelecimento de caminhos entre diferentes sistemas autônomos, é o BGP (*Border Gateway Protocol* – Protocolo de Roteador de Borda), cuja versão 4 foi originalmente definida no RFC 1771 e atualizada desde 2006 no RFC 4271. Trata-se de um protocolo de roteamento de vetor de distâncias, porém um pouco diferente, pois os roteadores rodando BGP têm ciência do caminho completo que utilizam. Para escolha do melhor caminho até o sistema autônomo pretendido, os roteadores BGP dão importância fundamental a políticas definidas pelos administradores, geralmente por questões negociais. Os destinos no caso do BGP não são *hosts*, mas sim prefixos de sub-redes(ou conjunto destas) baseados no CIDR. Juntando-se um prefixo com os seus chamados atributos BGP, tem-se uma rota BGP, que constitui a informação básica trocada pelos nós BGP na Internet.

Alguns atributos BGP são extremamente relevantes para o estabelecimento das rotas. Entre os principais estão o *local preference*, o AS-PATH (“caminho de sistema autônomo”) e o NEXT-HOP (“próximo salto”). O *local preference* é um atributo de decisão essencialmente política por parte do administrador da rede e que tem prioridade na escolha de rotas: a rota com o valor de *local preference* mais elevado é escolhida, caso não haja outra com valor igual. Em caso de empate, o atributo utilizado para determinar a rota é o AS-PATH, que constitui, para cada prefixo, uma lista de todos os AS pelos quais o anúncio passou. Se esse atributo fosse o primeiro e único a ser considerado, a escolha para melhor rota seria reduzida à execução de um algoritmo clássico de vetor de distâncias. Já o atributo NEXT-HOP é o terceiro na lista de prioridades: em caso de mesmo AS-PATH, é escolhida a rota com o roteador de próximo salto mais próximo. Em caso de novo empate, os roteadores têm de utilizar identificadores BGP para escolher a melhor rota.

2.4 MPLS E RSVP-TE

O MPLS (*Multiprotocol Label Switching*) é uma arquitetura de comutação de pacotes IP com base em um rótulo (*label*), que geralmente é composto por alguns bytes no começo de cada pacote. O rótulo é atribuído ao pacote na interface da rede externa com a rede MPLS por roteadores de rótulo de borda LER (*Label Edge Router*). Ao adentrar na rede MPLS, o pacote percorre caminhos LSP (*Label Switched Path*) formados pelos roteadores comuns de rótulo LSR (*Label Switch Router*), com cada rótulo correspondendo a um caminho. Inicialmente concebido para tentar aumentar o desempenho de redes IP com maior velocidade de processamento das informações da camada de rede, o MPLS é caracterizado pela transferência da maior parte do processamento referente à obtenção do caminho dos pacotes para as bordas, já que os LERs fazem o trabalho mais complexo computacionalmente de atribuir rótulos, ao passo que os LSRs comutam o pacote com base nestes pequenos rótulos – mais simples que os cabeçalhos IP – e na tabela LIB (*Label Information Base*). A tabela LIB contém a correspondência entre um rótulo e a sua classe de envio equivalente (*Forward Equivalence Class* – FEC), ou seja, o tipo dos pacotes com mesmos requisitos de QoS que devem recebê-lo [14]. A arquitetura MPLS foi definida na RFC 3031 [15].

No MPLS, é possível selecionar o caminho LSP para uma determinada FEC de duas maneiras. Com a seleção implícita de caminhos, ou *hop by hop*, cada LSR do caminho pode escolher o próximo nó de forma independente. Isso deve ser feito por meio de um algoritmo de roteamento com restrições (*constraints*) como o CSPF (*Constrained Shortest Path First*) que considere algumas métricas adicionais dos enlaces além de seu custo, como a taxa de transmissão, custo administrativo ou outros atributos do enlace. Já no modo explícito de seleção, a totalidade do caminho, ou grande parte dele, é escolhida por apenas um LSR, geralmente o roteador de entrada ou de saída do caminho, por configuração manual. Se todos os nós do caminho forem definidos por apenas um LSR, diz-se que o LSP em questão foi selecionado de forma estritamente explícita [15]. Em ambos os tipos de seleção, uma vez estabelecido o caminho MPLS, este permanece fixo, não importando as alterações e degradações que possam acontecer no desempenho da rede.

Com a evolução da tecnologia de processamento dos roteadores IP, houve um aumento na capacidade destes, e a maior velocidade de comutação das redes MPLS deixou de ser sua principal vantagem. Outros aspectos dessa tecnologia ganharam destaque, de modo que o MPLS se transformou em uma alternativa importante para soluções que envolvam redes virtuais privadas (VPN), suporte a QoS e engenharia de tráfego [16].

O MPLS se encontra, na pilha de protocolos, entre as camadas de enlace e a camada de rede, sendo independente de ambas, e é transparente ao protocolo IGP utilizado. Feito especificamente para o MPLS, o LDP (*Label Distribution Protocol*) realiza a distribuição de rótulos, o estabelecimento de sessões e outras funções de controle e manutenção destas. Contudo, é possível a utilização de outros protocolos para as mesmas tarefas, como o OSPF e o RSVP (*Resource Reservation Protocol*), protocolo definido na arquitetura *IntServ* (*Integrated Services*).

Com o RSVP, solicita-se a reserva de um determinado recurso na rede, ou mais especificamente, em cada nó. Para isso o protocolo, que ocupa o lugar de um protocolo de transporte na pilha, estabelece sessões para fluxos *simplex* (apenas em uma direção), com criação e manutenção de caminhos (subsidiadas por informações de um IGP) e sinalização de eventuais erros. A partir de parâmetros pré-estabelecidos, são definidos o *TSpec* (*Traffic Specification*), um especificador do fluxo que se quer transmitir em um túnel, e o *RSpec* (*Resource Specification*), um especificador dos recursos e requisitos de qualidade de serviço necessários para ele. Caso não haja recursos necessários, a requisição de reserva é barrada no controle de admissão do protocolo, que também permite classificação de pacotes para tratamento diferenciado.

A simplicidade do RSVP é uma vantagem, pois utiliza basicamente dois tipos principais de mensagens: PATH e RESV. A mensagem PATH é enviada do LER *sender* (ou roteador de borda de entrada) com destino ao LER *receiver* (ou roteador de borda de saída) de modo a estabelecer a sessão em cada LSR dos caminhos, com informações de classificação e *TSpec*. Já a mensagem RESV percorre o caminho contrário, começando

no LER *receiver*, carregando a requisição de reserva já aprovada pelo controle de admissão e o respectivo *RSpec* também por cada roteador intermediário até o LER *sender*, onde funciona como confirmação do estabelecimento do túnel e permite definição de parâmetros adequados para controle de tráfego [17][18]. A Figura 2.4 mostra um exemplo de transmissões dos pacotes PATH e RESV num túnel RSVP, com estabelecimento da sessão e posterior reserva de recursos combinada com definição de etiquetas para cada salto.

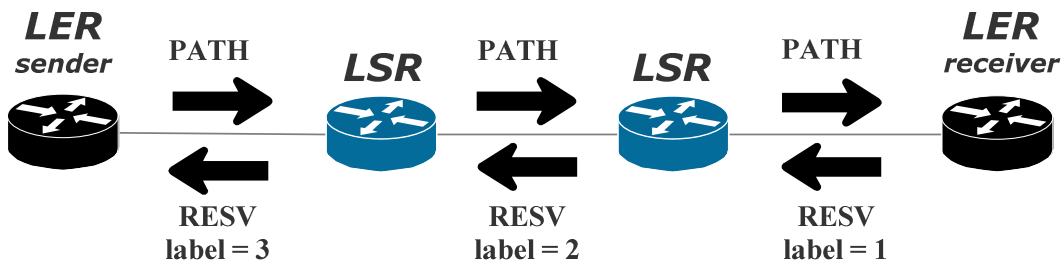

Figura 2.4 – Exemplo de estabelecimento de sessão RSVP com reserva. Baseado em [17].

A engenharia de tráfego é uma área em que o MPLS é bastante utilizado como ferramenta. O RSVP-TE (RSVP-*Traffic Engineering*) [19] é uma extensão do RSVP desenvolvida para prover suporte de engenharia de tráfego ao MPLS. Essa solução permite o estabelecimento de caminhos LSP explícitos com ou sem reserva de recursos, além do chaveamento suave para outros LSPs e detecção de *loops*. Cada sessão RSVP-TE, ou túnel, é caracterizado por uma identificação de túnel e pode conter vários caminhos LSPs entre os LERs *sender* e *receiver* com diferentes prioridades configuradas. Com o RSVP-TE, é possível configurar um caminho LSP *backup* para prover redundância a cada LSP principal. Caso haja uma falha envolvendo um caminho principal, o tráfego é encaminhado para o novo caminho por meio dessa solução de nome *MPLS Fast Reroute*.

As funcionalidades citadas do RSVP-TE podem servir como base de um sistema que permita a escolha do melhor LSP dentre um conjunto de caminhos previamente configurados. Um algoritmo de inteligência artificial pode ser explorado para a escolha, enquanto o RSVP-TE fornece a infraestrutura necessária.

2.5 CONCLUSÃO DO CAPÍTULO

Neste capítulo foram apresentadas as soluções difundidas atualmente nas redes para a obtenção dos caminhos mais adequados para um fluxo de pacotes. Foram evidenciados alguns pontos dos protocolos que explicam a dependência destes à atuação constante de um operador de rede. Para a implementação de um sistema que implemente de forma automática essas intervenções e interprete da melhor forma as alterações no desempenho das redes, será estudada no próximo capítulo a estratégia de aprendizado por reforço.

3 APRENDIZADO POR REFORÇO

3.1 INTRODUÇÃO

O aprendizado por reforço consiste em uma estratégia focada na interação do aprendiz com o ambiente ao seu redor de forma a mapear suas ações a cada situação. Aplicado em problemas de computação, é um ramo da inteligência artificial cada vez mais aplicado em diversas áreas de engenharia [20] e que une conhecimentos de estatística, psicologia, neurociência e ciência da computação [21].

Este capítulo traz a descrição do ambiente de aprendizado por reforço: um tomador de decisões, chamado de agente, escolhendo ações para alcançar um objetivo e medindo a eficácia destas com base no reforço que obtiver do ambiente, de modo a encontrar qual a melhor sequência de ações. Também serão expostos a modelagem matemática do problema, a solução para se obter maior ganho acumulado e um par de algoritmos comuns na área. Uma pequena revisão bibliográfica ao final do capítulo mostra alguns exemplos de aplicação desse tipo de algoritmos em redes de comunicação.

3.2 DEFINIÇÃO E MODELAGEM

No modelo de aprendizado por reforço, um agente realiza ações para atingir determinado objetivo e, para cada ação ou sequência delas, obtém um valor escalar como *ganho* do processo (*recompensa* ou *retorno*). As ações podem mudar o estado do ambiente em que se encontra o agente e o que se deseja é aprender a melhor sequência de ações, ou seja, aquela que resulte no melhor ganho acumulado. A sequência de tentativa e erro feita pelo agente pode ser guiada por uma série de algoritmos diferentes. A Figura 3.1 traz uma representação desse modelo, em que o agente modifica o ambiente em razão de uma ação nele aplicada e escolhida com base em um conjunto de regras chamada *política* [2]. O modelo de um ambiente descreve o comportamento deste

em face às diferentes ações em diferentes estados. Uma sequência completa de ações, desde o começo até o estado final, é chamada de episódio.

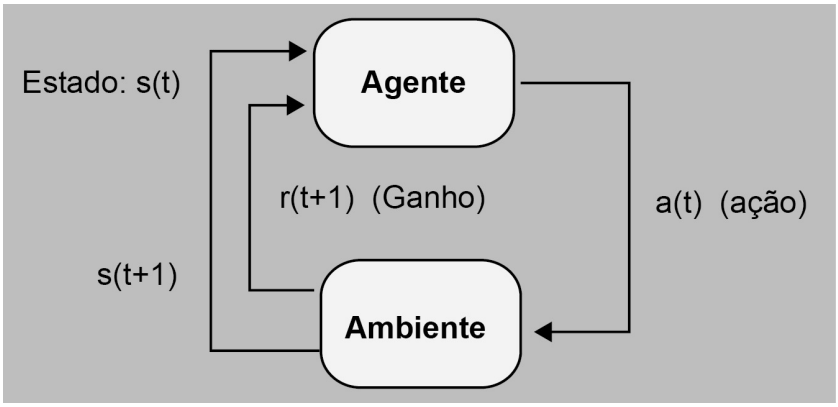

Figura 3.1 – Ambiente de aprendizado por reforço.

É possível que um ambiente tenha um número maior de agentes, que possam tomar ações diferentes interagindo entre si. Nesse caso, tem-se um problema de aprendizado por reforço com múltiplos agentes, com uma maior dificuldade para definir o objetivo de aprendizado coletivo. Este trabalho terá como foco o caso de aprendizado por reforço com agente único por sua relativa simplicidade e melhor adequação ao problema de referência.

3.2.1 Modelagem matemática do caso com agente único

Formalmente, o ambiente de aprendizado por reforço é definido como um processo de decisão de Markov (PDM) $\langle S, A, f, \rho \rangle$, em que: S é o conjunto de estados possíveis; A é o conjunto de ações possíveis para um agente; $f(s, a, s')$ é a probabilidade de se ir para o estado $s' \in S$ quando a ação $a \in A$ é tomada no estado $s \in S$; e $\rho(s, a, s')$ é a função que descreve qual o ganho obtido após a ação ser tomada e o estado mudar [22]. Pode-se dizer também que o ganho é dado por:

$$r_{k+1} = \rho(s_k, a_k, s_{k+1}), \tag{3.1}$$

em que k representa o instante discreto de tempo. Assim, na equação (3.1), a ação a_k foi escolhida pelo agente no instante k, quando o ambiente se encontrava no estado s_k. Após a ação, no próximo instante de tempo, o agente recebe como recompensa r_{k+1}, e o ambiente passa ao estado s_{k+1},

O comportamento do agente é definido pela política Π. Para o caso de uma política determinística, para cada estado há uma ação específica, ou seja $\Pi: S \rightarrow A$. Quando a política é estocástica, em cada estado há uma probabilidade de se executar cada uma das ações $a \in A$, ou $\Pi: S \times A \rightarrow [0,1]$. O valor V_k^{Π} de uma política é definido como o retorno esperado para o agente quando este segue Π a partir do estado s_k. O objetivo do agente é maximizar V_k^{Π}, que no modelo de horizonte infinito é dado por:

$$V^{\Pi}(s_k) = E\left\{\sum_{j=0}^{\infty} \gamma^j \cdot r_{k+j+1}\right\}, \tag{3.2}$$

em que $\gamma \in [0,1)$ é o fator de desconto. Quando γ é nulo, somente o ganho mais atual conta para o retorno esperado. A medida que γ se aproxima de 1, o agente considera cada vez mais ganhos futuros. O valor de γ é usualmente inferior a 1 para se evitarem problemas de convergência. Para cada política Π que pode ser seguida pelo agente, existe um retorno esperado $V^{\Pi}(s_k)$ e se quer encontrar a política ótima Π^* que maximiza esse valor:

$$V^*(s_k) = \max_{\Pi} V^{\Pi}(s_k), \forall\, s_k \tag{3.3}$$

3.2.2 Solução de um Processo de Decisão de Markov e algoritmos

Geralmente prefere-se trabalhar com o retorno esperado para uma ação específica em um determinado estado: $Q(s_k, a_k)$, que representa o quão bom é para o agente tomar a ação a_k quando se encontra no estado s_k. Se o agente segue a política Π, tem-se:

$$Q^{\Pi}(s_k, a_k) = E\left\{\sum_{j=0}^{\infty} \gamma^j \cdot r_{k+j+1} \middle| s_k = s, a_k = a \ , \Pi\right\} \tag{3.4}$$

A política ótima Π^* é aquela que acarreta maior valor de Q e corresponde àquela em que se tomam as melhores ações:

$$Q^*(s, a) = \max_{\Pi} Q^{\Pi}(s, a) \tag{3.5}$$

$$Q^*(s, a) = \max_{a'} Q^{\Pi}(s, a') \tag{3.6}$$

A solução da equação (3.6) é conhecida como equação de Bellman e pode ser escrita da seguinte maneira [23]:

$$Q^*(s, a) = \sum_{s' \in S} \left(f(s, a, s') \cdot \left[\rho(s, a, s') + \gamma \cdot \max_{a'} Q^*(s', a') \ \right] \right) \tag{3.7}$$

Uma vez definidos os retornos esperados $Q^*(s, a)$ para cada estado e ação, pode-se utilizar a política *greedy* (gulosa), que significa escolher, para cada estado, a ação a^* com maior valor Q correspondente. Contudo, quando se trata de aplicações realísticas de aprendizagem por reforço, não se tem tanto conhecimento acerca do ambiente, ou seja, da distribuição de probabilidades $f(s, a, s')$ e da função $\rho(s, a, s')$. Assim, é necessário exploração (*exploration*) do ambiente para investigar o modelo.

Algoritmos que funcionam sem necessidade de conhecimento completo do modelo matemático do ambiente são chamados de *model-free.* Um exemplo amplamente conhecido é o *Q-learning*, que implementa uma estratégia de estimação iterativa dos valores $Q^*(s, a)$baseada na equação (3.7). Por meio dela, a estimativa corrente de Q^* é atualizada por meio de estimativas de expressões do lado direito da equação (3.7), calculadas utilizando a experiência do próprio algoritmo, como a última recompensa obtida r_{k+1} e estimativas Q dos estados s_k e s_{k+1}:

$$Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha_k \left[r_{k+1} + \gamma \cdot \max_{a'} Q_k(s_{k+1}, a') - Q_k(s_k, a_k) \right] \quad (3.8)$$

Na equação (3.8), o termo $\alpha_k \in (0,1]$ é a taxa de aprendizado, que define o peso que a nova recompensa e o retorno esperado do próximo estado devem exercer no cálculo da estimativa atual do valor Q. Como se pode ver, o *Q-learning* independe de política específica: não importa qual ação será escolhida, a componente referente ao estado futuro sempre representa aquele o valor Q vinculado à ação com maior ganho.

Foi demonstrado em [24] que o *Q-learning* converge com probabilidade 1 se os valores $Q(s, a)$ forem atualizados infinitas vezes para cada par possível de estados e ações, com α gradualmente decrescente com o tempo. Definindo-se $k^i(s, a)$ como o índice de tempo da i-ésima vez que a ação a é escolhida no estado s, podem-se destacar as condições para convergência nas expressões (3.9) a (3.12):

- A recompensa deve ser limitada em módulo.

$$|r_k| \leq R \in \mathbb{R} \quad (3.9)$$

- A taxa de aprendizado deve ser não negativa e menor do que 1.

$$0 \leq \alpha_k < 1 \quad (3.10)$$

- As duas expressões abaixo indicam que a taxa de aprendizado deve diminuir conforme cada uma das ações vai sendo escolhida novamente em cada estado.

$$\sum_{i=1}^{\infty} \alpha_{k^i(s,a)} = \infty, \qquad \forall s, a \quad (3.11)$$

$$\sum_{i=1}^{\infty} \left[\alpha_{k^i(s,a)}\right]^2 < \infty, \qquad \forall s, a \quad (3.12)$$

Com as condições acima, pode-se provar que, com probabilidade 1:

$$\lim_{k\to\infty} Q_k(s,a) = Q^*(s,a), \qquad \forall s,a \tag{3.13}$$

O *Q-learning* possui complexidade computacional quadrática com relação ao número de estados e linear com relação ao número de ações possíveis por iteração [21], o que pode ser tornar um problema de escalabilidade [23].

Por outro lado, existe também um algoritmo análogo ao *Q-learning*, porém dependente da política específica utilizada para definir a próxima ação. Trata-se do SARSA (*State-Action-Reward-State-Action*), em que se utiliza como componente do próximo estado na atualização o valor Q do par estado-ação obtido aplicando-se a política escolhida. A expressão (3.14) representa a atualização do SARSA. Este, como é possível mostrar, converge quando os pares estado-ação possíveis são visitados infinitamente e se a política convergir, no limite, para a política gulosa [2].

$$Q_{k+1}(s_k,a_k) = Q_k(s_k,a_k) + \alpha_k[r_{k+1} + \gamma \cdot Q_k(s_{k+1},a_{k+1}) - Q_k(s_k,a_k)] \tag{3.14}$$

Algoritmos baseados na diferença entre estimativas dos valores Q^* de tempos diferentes são chamados de algoritmos de diferença temporal (*temporal difference* - TD). Este tipo de método, apesar de exigir análise relativamente complexa, não necessita de um modelo conhecido para o ambiente de aprendizado. Contudo, não é o único disponível para a solução de um PDM. A programação dinâmica, por exemplo, apresenta métodos bem desenvolvidos matematicamente, porém com a desvantagem de precisarem de um modelo completo para o ambiente. Por esse motivo, ela não é aplicada usualmente em casos mais realistas, nos quais é muito difícil ter esse tipo de informação.

Existem também os métodos de Monte Carlo, que não necessitam de um modelo e tem aprendizado baseado unicamente em experiência. Com este último tipo, apesar de conceitualmente simples, não se trabalha com computação incremental passo a passo.

Em comparação com os demais, o método de diferença temporal tem a vantagem de ser totalmente incremental e geralmente convergir mais rapidamente com tarefas estocásticas [20].

Por essas razões, este trabalho manterá o foco em algoritmos de TD. Como na maioria das vezes não se tem informações acerca do modelo do ambiente, este deve ser explorado de modo a mapeá-lo o máximo possível. Portanto, no momento de tomar uma ação, o agente deve decidir se vai realizar *exploitation*, ou seja, escolher aquela com maior valor Q para o estado atual, utilizando o que já foi aprendido, ou *exploration*, momento em que uma ação qualquer é escolhida de acordo com alguma distribuição de probabilidade com o objetivo de conhecer o ambiente.

Há diversas técnicas disponíveis para tal, sendo uma das mais simples e difundidas a chamada ε-*greedy*. Nesta estratégia, com probabilidade ε o agente escolhe uma ação aleatória para realizar (*exploration*), ao passo que com probabilidade $1 - \varepsilon$, realiza-se *exploitation*. Pode-se fazer ε variável de modo que se tenha mais *exploration* inicialmente e, com o avançar do tempo e concomitante diminuição de ε, gradualmente se aumente o índice de *exploitation* [2].

Uma breve revisão bibliográfica sobre aplicações do aprendizado por reforço em redes de comunicação é apresentada na Seção 4.2. Mais informações sobre aprendizado por reforço com agente único podem ser encontradas em [2] e [25]. Já um exame abrangente e detalhado sobre aprendizado por reforço com múltiplos agentes pode ser encontrado em [23].

3.3 CONCLUSÃO DO CAPÍTULO

Neste capítulo foi apresentada a modelagem de aprendizado por reforço para problemas computacionais, que pode ser uma ferramenta importante para soluções em várias áreas da engenharia. Particularmente, neste trabalho será feito um esforço para adequar o problema prático de roteamento adaptativo a este modelo de ambiente. O capítulo a seguir fará uma revisão bibliográfica sobre outras soluções de roteamento

adaptativo que tiveram relativo sucesso e que deram uma base teórica e de entendimento do problema.

4 ESTRATÉGIAS DE ROTEAMENTO ADAPTATIVO

4.1 INTRODUÇÃO

O roteamento com algoritmos de menor caminho está baseado na definição de parâmetros muito importantes: os custos dos enlaces da rede. O custo de um enlace deve representar alguma métrica de desempenho da rede. A mais utilizada e recomendada por fabricantes, como abordado no Capítulo 2, é a que atribui como custo de cada enlace um valor proporcional ao inverso de sua taxa de transmissão, o que significa que *links* com maior taxa possuem maior chance de serem utilizados. Essa estratégia implica em enlaces com custos fixos, uma definição estática de custos [10].

Podem-se também utilizar como métrica parâmetros que variam com o tempo, o que abre a possibilidade de existência de um método de definição dinâmica do caminho dos pacotes. Este tipo de roteamento em que as rotas são alteradas em decorrência de mudança de condições da rede é chamado de roteamento adaptativo [5] e neste capítulo serão apresentados exemplos nessa área.

As abordagens serão dividas em três categorias principais. Na Seção 4.2, serão mostradas técnicas de roteamento adaptativa baseadas em aprendizado por reforço, fugindo do paradigma dos algoritmos de menor caminho comerciais. Na Seção 4.3, serão apresentadas propostas de melhoria feitas ao protocolo OSPF com o objetivo de torná-lo mais sensível a mudanças no desempenho das redes. Já na última seção será descrita uma estratégia para se basear o roteamento em várias métricas conjuntamente.

4.2 ROTEAMENTO ADAPTATIVO COM APRENDIZADO POR REFORÇO

Vários trabalhos em redes de comunicação em geral vêm aplicando algoritmos de aprendizado por reforço para melhorar o desempenho de redes de comunicação, em escopos diferentes. Por exemplo, em [26], o problema de modulação e codificação adaptativas em sistemas OFDM foi adaptado ao modelo de aprendizado por reforço: o estado é descrito pela relação sinal-ruído média de um conjunto de subportadoras, ao

passo que cada combinação possível de taxa de modulação e codificação é uma ação que pode ser tomada. No caso, a recompensa definida é uma função que representa a vazão alcançada pelo sistema depois de escolhida uma dada ação. No trabalho, foi comparado o desempenho do algoritmo proposto com o da difundida solução baseada em tabelas de consulta, obtendo-se uma eficiência espectral bastante próxima no caso de canal com ruído branco gaussiano. Contudo, quando o canal em que os testes foram feitos possuía ruído branco gaussiano e interferência colorida, a eficiência espectral da solução de aprendizado por reforço se mostrou superior, provando ser um método adequado para aprendizado em tempo real.

Uma aplicação clássica de algoritmos de aprendizado por reforço pode ser destacada no campo do roteamento, como apontado em [23]. No caso deste problema, o roteador pode assumir o papel do agente, sendo o ambiente a própria rede. A ação do modelo corresponde ao ato do roteador enviar um pacote por uma das interfaces disponíveis e, para o ganho resultante, pode-se escolher um parâmetro de QoS medido no caminho total até o destino em questão, por exemplo. Tem-se assim um novo paradigma, uma estratégia de roteamento sem relação com algoritmos de melhor caminho ou com os protocolos de roteamento usuais na Internet, já que cada transmissão de pacote de dados deve ser receber uma recompensa à qual se deve associar.

Um caso em que se aplicaram esses princípios foi o do algoritmo *Q-routing*, um algoritmo de roteamento proposto inicialmente em 1993 em [27] e com modificações dos mesmos autores em 1994 em [28]. Embora baseado nos elementos do aprendizado por reforço, não se trata exatamente de uma aplicação deste, já que no *Q-routing* não há a representação do ambiente por estados.

Com o *Q-routing*, cada roteador mantém, para cada endereço de destino, uma estimativa do tempo de entrega de um pacote para cada interface de saída do nó. A essas estimativas dá-se o nome de valores Q, e $Q_x(d, y)$ representa a estimativa de tempo de entrega de um pacote do roteador x ao destino d, saindo pela interface conectada ao vizinho y. É escolhida para repasse de um pacote a interface que tem a menor estimativa

para o tempo de entrega do pacote ao destino em questão. O vizinho conectado à interface escolhida (por exemplo, roteador *y*) envia de volta ao primeiro nó (roteador *x*), após receber o pacote, um pequeno pacote contendo sua melhor estimativa para tempo de entrega até o destino, além de informações como o instante em que o pacote original nele chegou.

No presente trabalho, chama-se o pequeno pacote usado para informar o atraso de QAck, para simplificação do entendimento. Com essas informações, o roteador A pode atualizar sua estimativa da interface utilizada para certo destino da seguinte maneira:

$$\Delta Q = \alpha\big(q + s + t - Q_x(d, y)\big) \tag{4.1}$$

$$t = \min_z Q_y(d, z) \tag{4.2}$$

$$Q_x(d, y) \leftarrow Q_x(d, y) + \Delta Q, \tag{4.3}$$

em que q é o tempo que o pacote passou na fila no roteador *x*, s é o tempo de transmissão e propagação do roteador *x* ao roteador *y*, t é a menor estimativa que y tem para a transmissão até o destino *d*e α é a taxa de aprendizado. Assim, podem-se atualizar as estimativas locais com informações de nós vizinhos. O *Q-routing* apresentou resultados promissores com respeito ao tempo médio de entrega de pacotes, sobretudo em cenários com elevada carga. Na Figura 4.1 é possível verificar o funcionamento do protocolo, com a interação entre roteadores vizinhos. Após o recebimento de pacote de dados por um roteador, este devolve à origem um pacote QAck com informações de atraso para atualização da tabela.

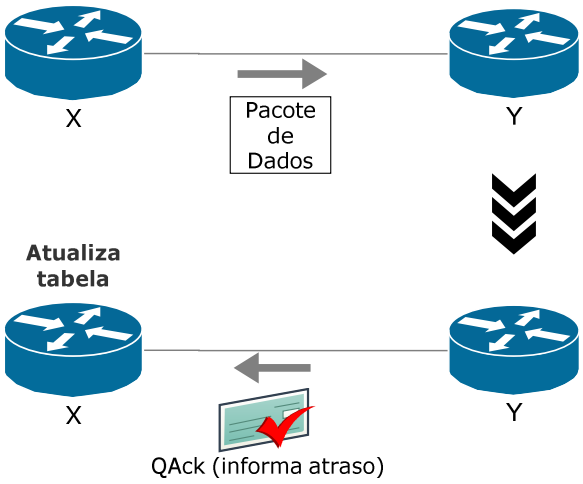

Figura 4.1 – Funcionamento do protocolo *Q-routing*.

O *Q-routing* apresenta uma falha grave. Caso a estimativa do atraso em uma interface sofra um drástico aumento, elevando seu correspondente valor *Q* a níveis muito acima dos demais, corre-se o risco dessa interface nunca mais ser testada. Para que uma interface drasticamente congestionada volte a ser escolhida, é preciso que sua estimativa de atraso seja ultrapassada por todos os demais. Isso ocorre mesmo se a situação de estresse para as respectivas partes rede se finde rapidamente. Essa característica deve-se ao fato de que, nesse algoritmo, não existe estratégia de *exploration*, como a que foi descrita no Capítulo 4. Os autores que propuseram o *Q-routing* justificam a falta desse tipo de estratégia para o protocolo pelo receio de se causar instabilidade na rede com alternância excessiva de caminhos.

Várias adaptações desse algoritmo foram feitas, em busca de uma maior otimização da rede em relação à métrica do atraso dos pacotes, inclusive para utilização em redes móveis [29]. Por exemplo, em [30] apresenta-se o *Predictive Q-routing*, um esquema para, eventualmente, se escolher interfaces preteridas por muito tempo para serem testadas novamente se já tiveram boas estimativas no passado. Em [31], foi adicionada atualização de estimativas no sentido reverso para diminuir o tempo de adaptação com carga baixa e melhorar o desempenho com carga alta, com o protocolo chamado de *Dual Reinforcement Q-routing*. Já [32] traz uma modificação para renovar estimativas que ficam inalteradas por muito tempo por meio de medida de confiança: o *Confidence-Based Q-routing*, uma tentativa de sanar a falha referida anteriormente.

Por outro lado, em [33] apresentou-se um algoritmo de roteamento baseado em busca por política via gradiente de subida, ou seja, ajuste dos parâmetros na direção do gradiente empiricamente estimado da recompensa agregada. Dessa maneira, definiram-se as ações identicamente às do *Q-routing*, porém a recompensa passa a ser o número médio de pacotes entregues por unidade de tempo (pacotes cujo recebimento tenha sido confirmado). Esta nova abordagem rendeu melhor desempenho que o *Q-routing*, notadamente com alto fluxo de pacotes na rede.

4.3 ROTEAMENTO ADAPTATIVO COM OSPF

Embora a estratégia usual de se configurar o peso do enlace como o inverso de sua taxa de transmissão resulte em um desempenho satisfatório de uma rede OSPF com baixo fluxo de pacotes [34], alguns trabalhos trouxeram novas maneiras de se definirem dinamicamente esses pesos de modo a tentar otimizar a rede para determinados parâmetros de QoS.

Em [35], por exemplo, foi apresentado o *Cost Adaptive OSPF* (CA-OSPF) uma solução em que os pesos das interfaces no OSPF são configurados dinamicamente com base na utilização dos *links*, com o objetivo de que os menores caminhos sejam também os melhores, ou seja, sem congestionamentos. Essa estratégia envolve a definição de dois limiares no nível de utilização U da interface: limite superior $U_a = 95\%$ e limite inferior $U_b = 45\%$. Cada interface tem seu custo inicial C_0 como referência e custo mínimo.

O nível de utilização de cada interface é medido periodicamente e o algoritmo do CA-OSPF procede da seguinte maneira: se o nível de utilização ficar acima do limite superior U_a por três medições consecutivas, o custo da interface deve ser incrementado de Δ, o valor mínimo de incremento. Caso o nível de utilização da interface esteja abaixo do limiar inferior U_b, o custo relacionado a ela deve ter seu valor reduzido de Δ unidades. Os custos devem ficar obrigatoriamente dentro do intervalo $[C_0\ , C_0 + \Delta_{MAX}]$, sendo Δ_{MAX} o incremento máximo permitido para evitar aumentos indefinidos quando

toda a rede estiver congestionada. Se o custo da interface for de fato alterado, deve haver geração e distribuição de nova *Router-LSA* do roteador que a contém para que o novo atributo seja conhecido pelos outros roteadores da rede.

Mostrou-se que o CA-OSPF é efetivo lidando com problemas locais de congestionamento, mas, quando este se alastra na rede em geral, não há diferenças com o desempenho da rede com OSPF puro. Uma vantagem do algoritmo é que os roteadores que rodam CA-OSPF são compatíveis com os roteadores com OSPF convencional.

Em [36], apresenta-se uma estratégia também baseada na taxa de utilização das interfaces, porém com estimação da banda efetiva de cada *link* calculada levando-se em conta o tipo de tráfego: de Poisson ou autossimilar. Com o algoritmo desenvolvido, o roteador somente adapta o custo de sua interface se identificar alto nível de ocupação ($U > 70\%$) ou de perdas ($P > 1\%$) no enlace conectado a ela. Se este for o caso, passa-se ao cálculo da estimativa da banda efetiva no enlace, inicialmente com a estimação do parâmetro de Hurst. Caso sua estimativa seja igual a 0,5 ou maior, a banda efetiva é calculada com o modelo de tráfego autossimilar. Caso contrário, é utilizado o modelo de tráfego de Poisson. Os modelos utilizados para o cálculo da banda efetiva não levam em conta o atraso como parâmetro, e a probabilidade de estouro do *buffer* é a restrição de QoS considerada. Assim, no momento da adaptação do algoritmo, o novo custo C_i da interface *i* é calculado da seguinte maneira:

$$C_i = 10 \cdot \frac{B_{ef}}{Taxa\ de\ transmissão_i}, \tag{4.4}$$

em que B_{ef} representa a banda efetiva calculada e $Taxa\ de\ transmissão_i$ diz respeito à taxa de transmissão da interface. Esse método foi testado em uma rede com matriz de tráfego já conhecida e a sua atualização adaptativa dos custos dos enlaces propiciou melhor balanceamento de carga quando comparado com o OSPF com custos fixos, o que melhorou para quase todos os fluxos parâmetros de QoS como atraso médio e perda de pacotes. Contudo, foi constatado que o processo adaptativo proposto não pode atuar

diretamente na rede alterando os custos em todas as iterações em razão da instabilidade causada pelo número elevado de cenários de custos visitados.

Em 2011, o trabalho [37] apresentou uma revisão ao protocolo OSPF para considerar, dinamicamente, o atraso em cada *link* e sua variação. Essa nova versão foi batizada de *Dynamic OSPF*. Os autores ressaltaram que um algoritmo de roteamento que utilize unicamente um parâmetro dinâmico de QoS como métrica costuma gerar instabilidade na rede, com o pior cenário possível sendo tráfego oscilando entre dois ou mais caminhos até o destino. Assim, foi desenvolvida uma estratégia para que o custo de um determinado enlace dependa não só de sua capacidade, mas também do atraso presente nele e sua variação. Assim, o custo C_i de cada enlace i segue a seguinte expressão:

$$C_i = \frac{10^8}{Taxa\ de\ transmissão_i} + \left(\frac{t' + m}{3{,}5\ \mu\ \text{s}}\right), \tag{4.5}$$

em que t' representa a estimativa de atraso médio no enlace e m, a estimativa de variação média no mesmo. Verifica-se que a solução utiliza o valor de $3{,}5\ \mu$ s como uma espécie de atraso de referência previamente determinado, independentemente da rede. Esse fato pode não se apresentar adequado para todo tipo de rede, uma vez que os perfis de atraso nos *links* das redes variam de acordo com suas características. Na solução em questão, para cada enlace as referidas estimativas são calculadas de acordo com as equações (4.6) e (4.7). Assim, depois de uma medida de atraso t_k no instante de tempo k:

$$t'_k = \alpha \cdot t'_{k-1} + (1 - \alpha) \cdot t_k \tag{4.6}$$

$$m_k = \alpha \cdot m_{k-1} + (1 - \alpha) \cdot |(t_k - t'_k)| \tag{4.7}$$

Para o trabalho em questão, o atraso em um determinado enlace é medido com uma aproximação que utiliza o número de pacotes no *buffer* de transmissão, expressa na seguinte equação:

$$t = num_{pacotes} \cdot \left(\frac{4 \cdot 10^6}{Taxa\ de\ transmissão} + 0{,}1\ \text{ms} \right) \quad (4.8)$$

Pode-se observar que a equação (4.8), utilizada em [37], não depende do tamanho médio do pacote, ou outros parâmetros que normalmente seriam considerados ao se calcular atraso médio em filas. Possivelmente os autores definiram algum valor médio geral que resultou nos números da equação (4.8), porém não foi explicitado se esses valores englobariam todos, ou ao menos a maior parte dos casos de redes.

Ainda em [37], a solução foi testada em uma topologia específica em contraponto com o OSPF puro e um protocolo que leva somente em conta o atraso como métrica, chamado de HELLO. Para cargas baixas injetadas na rede, o OSPF original apresentava um desempenho superior aos dois outros, com valores inferiores de atraso. Contudo, o desempenho da rede com OSPF original se deteriorava consideravelmente com cada passo de aumento da carga, enquanto o *Dynamic OSPF* indicou uma piora até o nível mediano de carga e a partir desse ponto começou a se estabilizar, apresentando o melhor nível de atraso médio em cargas altas. O protocolo HELLO apresentou o melhor desempenho em carga mediana, porém com cargas altas, apresentou atrasos maiores que o *Dynamic OSPF* em razão da instabilidade entre dois caminhos.

Apesar dos resultados positivos, os testes para a solução apresentada foram feitos em apenas um cenário, o que suscita dúvidas a respeito da aplicabilidade desta em redes reais, com uma vasta possibilidade de topologias e cenários.

4.4 ROTEAMENTO COM MÚLTIPLAS RESTRIÇÕES

De acordo com [12], com a consolidação da Internet e advento das redes NGN (*next generation networks*), hoje é necessário que haja conectividade entre redes heterogêneas de operadores distintos. O mesmo trabalho propôs então um esquema relativamente simples para estabelecimento de melhores caminhos entre esses diferentes domínios, ou AS's, assumindo que existem túneis (por exemplo *Label-Switched Paths*

do MPLS) configurados dentro dos domínios e entre eles. Em cada túnel são feitas medidas das seguintes métricas de QoS: atraso (soma do atraso de fila na porta de saída com o tempo de transmissão e tempo de propagação), *jitter*, perda de pacotes e disponibilidade (probabilidade de que um túnel esteja disponível). Para cada métrica, os túneis são classificados em ordem crescente (decrescente para a disponibilidade), cada um ficando com índice *i*, de um total de *N*. É calculado um *fitness value* para cada túnel e cada métrica da seguinte maneira:

$$\frac{2 \cdot i}{\mathrm{N}(\mathrm{N}+1)} \tag{4.9}$$

Uma métrica composta (*CompositeMetric* - CM) é então calculada para um determinado túnel somando-se os *fitness values* de cada uma das quatro métricas consideradas. O melhor caminho entre dois pontos em domínios diferentes é calculado por meio do algoritmo de Dijkstra em que os túneis fazem o papel de arestas e tem como peso a respectiva CM. Por isso, essa estratégia foi batizada de Dijkstra(CM).

Embora não se trate de uma solução automatizada de algoritmo de roteamento adaptativo, o fato de permitir o cálculo de pesos de enlaces utilizando uma métrica composta que mude com o desempenho da rede abre possibilidades de novas soluções na área que envolvam mais de um parâmetro de QoS.

Ainda em [12], a solução foi confrontada em testes com outros dois métodos conhecidos na literatura para múltiplas restrições: min-max e minSS. Dado um caminho P, o valor total de uma métrica k associado a ele é dado por $\omega_k(P)$. Se S_k for a restrição (ou limite superior) da métrica k, temos $\omega_k(P) < S_k$. Com o método chamado de *min-max*, De Neve e Van Mieghem [38] usaram a seguinte expressão para descrever o custo total de um caminho levando em consideração várias métricas:

$$\max\left[\frac{\omega_1(P)}{S_1}, \frac{\omega_2(P)}{S_2}, \ldots, \frac{\omega_k(P)}{S_k}\right] \tag{4.10}$$

Ou seja, o maior peso relativo de um caminho de acordo com uma determinada restrição é escolhido para representar todo o caminho. Já em [39], foi proposto um algoritmo heurístico para encontrar um caminho viável que satisfaça restrições de k métricas diferentes minimizando a seguinte expressão:

$$\left(\frac{\omega_1(P)}{S_1}\right)^{\lambda} + \left(\frac{\omega_2(P)}{S_2}\right)^{\lambda} + \cdots + \left(\frac{\omega_k(P)}{S_k}\right)^{\lambda} \tag{4.11}$$

Como em [39] utilizou-se principalmente $\lambda = 2$, o método ficou conhecido como *minimum sum of squares* (*minSS*). Provou-se que, ao se minimizar a expressão (4.11), pelo menos uma das restrições é respeitada, ao passo que as outras métricas podem ficar sujeitas a uma restrição igual à original multiplicada por $\sqrt[\lambda]{k}$.

Para os testes comparativos, são geradas 300 matrizes aleatórias com informações de atraso, *jitter*, perda e disponibilidade para cada túnel em uma topologia com 10 domínios e 60 roteadores. São escolhidos 5 pares de origem e destino cuja distância mínima é 2, 4, 6, 8 e 10 saltos. Para cada par e para cada matriz, é calculado o melhor caminho com as três técnicas citadas. Também é calculado o melhor caminho utilizando Dijkstra e cada métrica individualmente para comparação. São colhidos resultados das métricas em cada uma das 300 ocorrências de cada par e é feita a média aritmética. A estratégia Dijkstra(CM), quando comparada a essas duas técnicas, obteve desempenho superior, inclusive chegando perto dos resultados ótimos individuais de cada métrica.

4.5 CONCLUSÃO DO CAPÍTULO

Os algoritmos de roteamento adaptativo estudados neste capítulo possuem um aspecto em comum: o seu uso em ambientes intra-AS de redes comerciais reais pode não ser o mais adequado. Dentre as razões, os algoritmos baseados em aprendizado por reforço e descendentes do *Q-routing* não são compatíveis com os protocolos comerciais e necessitariam de um novo paradigma de roteamento que pode não ser eficiente em todos os casos. Por outro lado, os algoritmos que resultaram de tentativas de tornar o

OSPF mais sensível às alterações no desempenho da rede devem mostrar um cuidado especial para evitar instabilidades na rede provocadas pela mudança nos pesos dos enlaces. Estes muitas vezes se mostraram benéficos em situações mais específicas.

Com o fim de apresentar uma alternativa que possibilite a aplicação prática de um esquema adaptativo, neste trabalho foi desenvolvida uma ferramenta computacional por meio do *software* OMNeT++, que será apresentada no próximo capítulo.

5 APRENDIZADO POR REFORÇO APLICADO A ENGENHARIA DE TRÁFEGO

5.1 INTRODUÇÃO

Conhecendo as limitações dos protocolos de roteamento utilizados largamente na Internet, como visto no Capítulo 2, é possível inferir que existem campos não estudados, havendo possibilidades para adaptações com algoritmos de inteligência artificial e, mais especificamente, aprendizado por reforço para possível melhoria no desempenho da rede. Este capítulo mostra o desenvolvimento da solução encontrada com este trabalho para estabelecimento, no caso de um determinado fluxo de dados, do caminho MPLS-TE com o menor atraso fim a fim.

A implementação foi realizada utilizando o *software* simulador de redes OMNeT++, cuja descrição sucinta é realizada na próxima seção. Na Seção 5.3 é descrita a reprodução do protocolo *Q-routin*g no ambiente OMNeT++.

O pacote INET Framework já dispõe de modelos para vários protocolos de variadas camadas, inclusive para MPLS e RSVP-TE. Foi aproveitada a base desses modelos e, com as devidas modificações, foi adicionado suporte à configuração de vários caminhos por cada sessão RSVP-TE, que vão ter seu desempenho medido pelo atraso fim a fim para que o caminho cuja utilização para transmissão resulte no melhor atraso possível. Os detalhes serão contados na Seção 5.5. Na última seção, serão mostrados os resultados obtidos após os testes com a solução proposta.

5.2 O SIMULADOR OMNET++

O *software* OMNeT++ [3] é um *framework* de simulação gratuito para usos acadêmicos que foi desenvolvido para ser o mais geral possível, tendo em mente a simulação de redes de comunicação e outros sistemas distribuídos. Com o tempo, foi sendo utilizado para diversos fins com o lançamento de *frameworks* de modelos específicos, como o MiXiM para redes sem fio fixas e móveis e o mais geral INET,

contando com modelos de vários dos protocolos mais usados. Modelos presentes no INET incluem: Ethernet, PPP, IP, UDP, TCP, IP, MPLS, OSPF, entre outros.

O OMNeT++ é uma plataforma de simulação de eventos discretos modular e baseado em C++, portanto, orientado a objetos. A tela principal do OMNeT pode ser vista na Figura 5.1, com as três principais áreas do ambiente de simulação: a área (1) corresponde ao *packet explorer*, local em que se navega pelos diretórios dos diferentes projetos de simulação, contendo tanto os arquivos descritores de topologia (.NED) como arquivos descritores de pacotes (.msg), além de arquivos em C++ que realmente implementam as funcionalidades por trás dos módulos ou arquivos de configuração dos cenários; a área (2) representa a área de trabalho do ambiente de simulação, em que se manipulam topologias em formato gráfico ou códigos tanto de componentes quanto de arquivos de configuração de cenários, além de visualização e manipulação de resultados; por fim, a área (3) possui várias funções, dependendo das atividades que se realiza, incluindo exibição de resultados de pesquisas, resultados de compilação, console ou mesmo os dados de pontos em gráficos de resultados, entre outras.

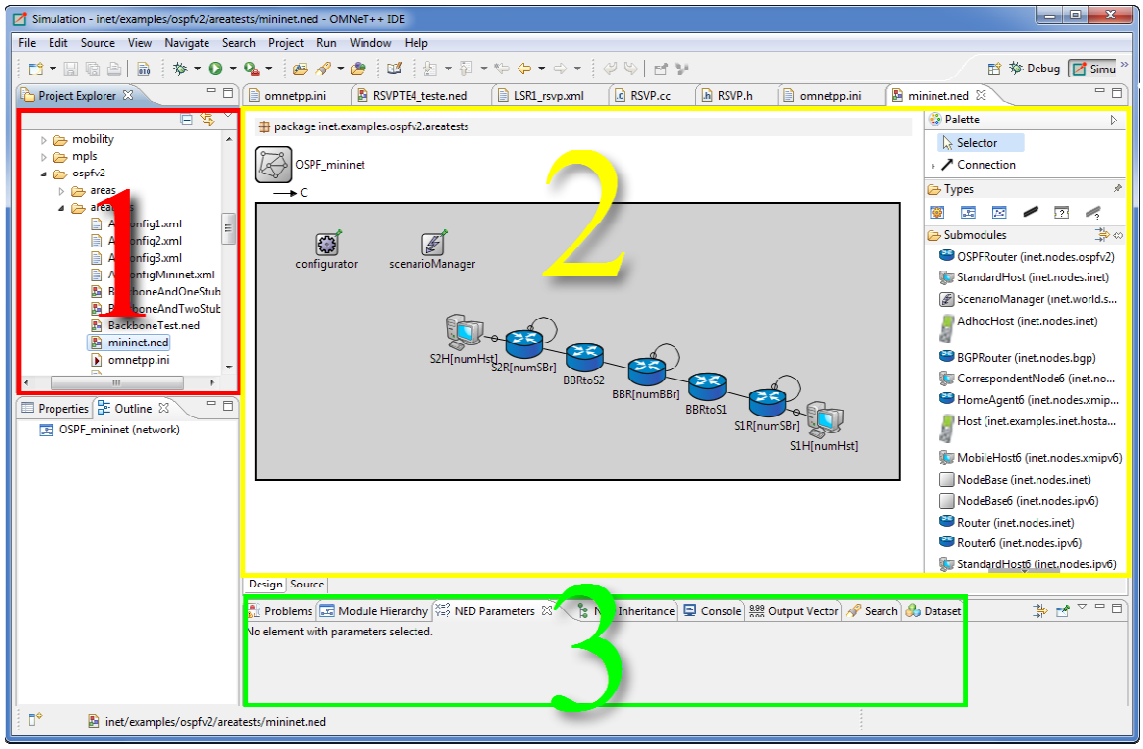

Figura 5.1 – Tela principal do OMNeT++ com suas três áreas principais.

Cada componente de uma topologia é formado por diferentes módulos e sub-módulos em vários níveis, que devem ser vinculados a seus correspondentes códigos C++ que descrevem sua funcionalidade.

5.3 REPRODUÇÃO DO PROTOCOLO Q-ROUTING NO OMNET++

O pacote INET (1ª versão) do OMNeT++ (versão 4.2) foi utilizado para a implementação do protocolo *Q-routing* [28], apresentado na Seção 4.2. O modelo de nó *Router*, um roteador genérico que não roda nenhum protocolo de roteamento específico e basicamente encaminha pacotes segundo o que está registrado na tabela de roteamento, foi modificado para que cada interface tenha uma estimativa de atraso específica para cada destino conhecido. Cada estimativa deve ser atualizada por meio da equação (4.3) quando um vizinho recebe um pacote de dados e envia o valor do atraso medido para o roteador imediatamente anterior por meio de um pequeno pacote chamado QAck. Para a alteração pretendida, foi necessário alterar a classe “IP.cc” para que o encaminhamento de pacotes seguisse a chamada Tabela Q, que registra um conjunto de valores Q para cada destino, ao invés da tabela de roteamento comum.

A Figura 5.2 mostra um esquema de funcionamento do encaminhamento de pacotes no caso do roteador original, em cujo módulo o algoritmo de roteamento disponível (ou rotas estáticas) alimenta a tabela de roteamento, que por sua vez é consultada pelo módulo IP no momento para se realizar o encaminhamento. A situação pretendida com o *Q-routing* pode ser vista na Figura 5.3, em que se vê um novo processo funcionando dentro do âmbito do IP no roteador, o que se chamou de Processo Q. Esse processo é responsável por atualizar a Tabela Q e avaliá-la para informar no momento do encaminhamento qual é a interface mais vantajosa a se utilizar para a transmissão a um determinado destino. A Figura 5.4 traz o fluxograma completo do protocolo *Q-routing*, que é implementado pelo referido Processo Q.

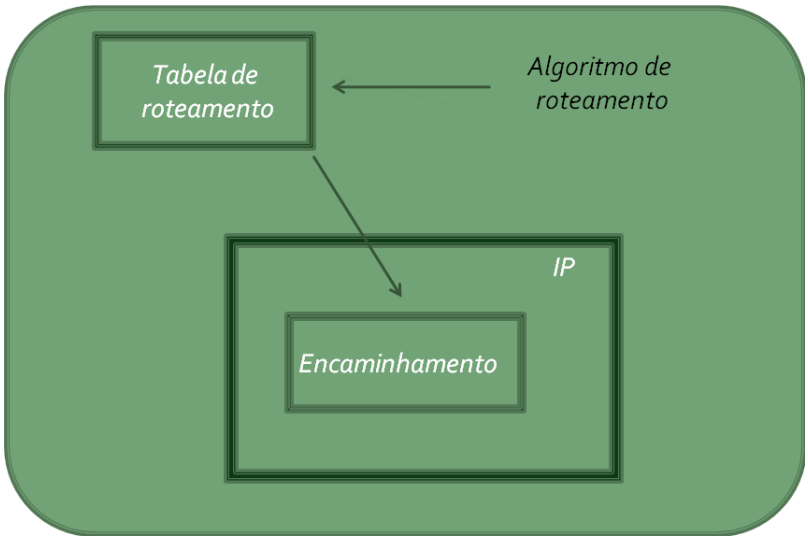

Figura 5.2– Esquema de funcionamento do encaminhamento de pacotes no roteador original do INET.

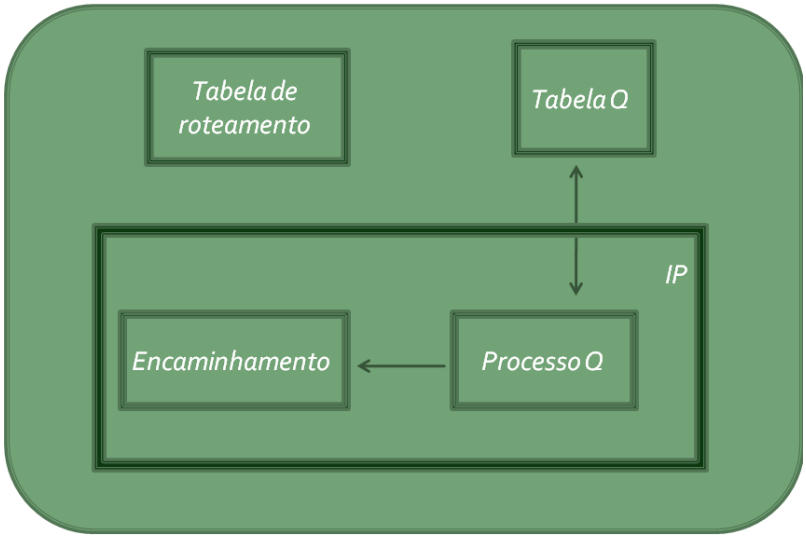

Figura 5.3 – Esquema de funcionamento do encaminhamento de pacotes no roteador com *Q-routing*.

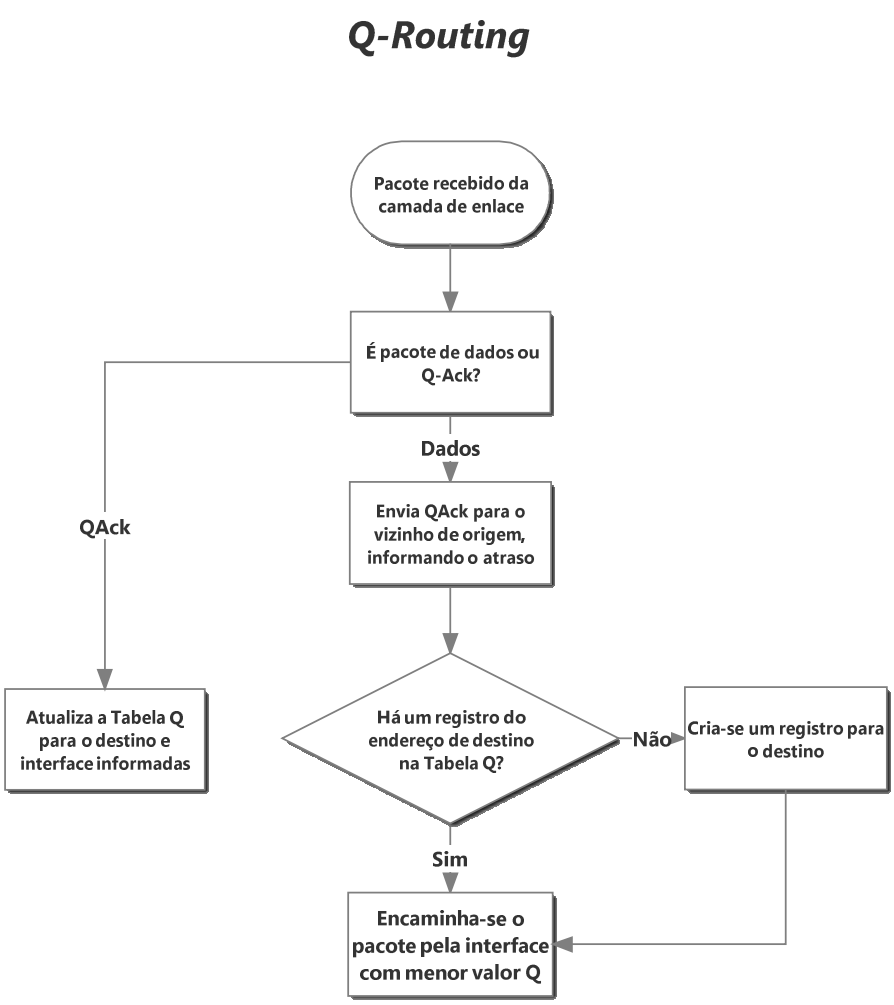

Figura 5.4 – Fluxograma do Processo Q, que implementa o *Q-routing*.

O protocolo foi, então, testado em uma topologia bastante simples, com baixa carga. O cenário, mostrado na Figura 5.5, contém três caminhos limitados por capacidades diferentes do *host 0*, a origem do tráfego, até *host 1*, o destino. O caminho de cima possui um enlace de 10 Mbps, o de menor capacidade da rede. O caminho de baixo contém um enlace de 100 Mbps, uma capacidade intermediária. Entre eles, um enlace com capacidade de 512 Mbps, igual a todos os outros demais da rede. O resultado esperado é que o *Q-routing* inicialmente teste as possibilidades e convirja para a escolha do caminho do meio, uma vez que, como não há congestionamento, não atraso considerável de fila que diferencie os caminhos, ficando a diferença entre eles por conta do atraso de transmissão. Como o caminho do meio possui maior capacidade, o

esperado era que ele fosse o escolhido e assim o foi, com variações de acordo com o valor escolhido para o fator de aprendizado α.

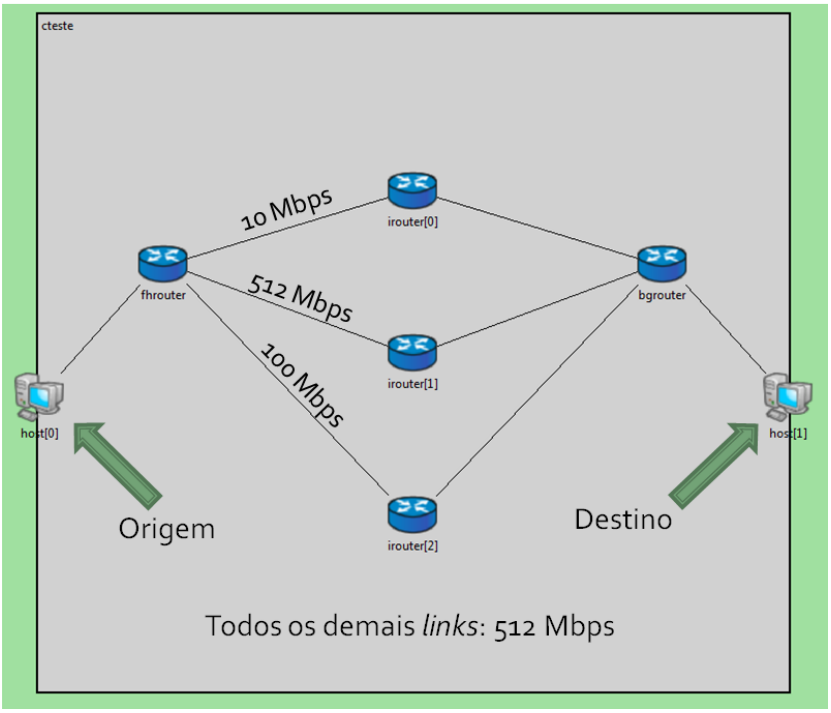

Figura 5.5 – Cenário simples para teste do *Q-routing*.

Foram feitas simulações com valores diferentes do parâmetro α para se verificar se o valor 0,7, sugerido pelos criadores do protocolo, é realmente o melhor. Os resultados desses testes encontram-se na Tabela 5.1. É possível observar que o valor 0,7 ofereceu muito melhor tempo de convergência em comparação a valores menores. Até $\alpha = 1{,}5$, os resultados foram muito parecidos com o caso de $\alpha = 0{,}7$, porém acima disso o valor da estimativa cresce tanto que não há convergência.

Tabela 5.1 – Testes para o valor de $\boldsymbol{\alpha}$

	Q-routing		
α	0,05	0,7	1,5
Atraso médio (s)	0,31	0,32	Overflow
Tempo de convergência (s)	51,88	4,57	Overflow
Pacotes enviados	39998	39998	Overflow
Perdas (%)	0,05	0,06	Overflow

5.4 PROPOSTA DE ESTRATÉGIA PARA ESCOLHA INTELIGENTE DE TÚNEIS

Como definido no Capítulo 1, neste trabalho se tem um problema específico a tratar: como encontrar, dentro de um AS, um caminho que proveja qualidade de serviço a um determinado cliente, com melhor adaptação a mudanças no desempenho da rede? A estratégia proposta consiste em uma alteração no protocolo RSVP-TE de modo a lhe acrescentar inteligência no roteador de borda de entrada da sessão, criando o que se chamou de RL-RSVP-TE (*Reinforcement Learning* RSVP-TE). O objetivo é que, de um grupo de caminhos previamente escolhidos e configurados explicitamente, seja escolhido automática e dinamicamente para a transmissão aquele que apresentar melhor desempenho de acordo com um determinado parâmetro de QoS, numa solução centralizada. No caso deste trabalho, o parâmetro escolhido foi o atraso fim a fim no túnel.

Desta forma, se quis desenvolver uma ferramenta computacional que receba como entrada uma sessão RSVP-TE com N caminhos diferentes entre uma origem e um determinado destino, e que consiga manter o tráfego no caminho com melhor medida de atraso, adaptando-o de acordo com as mudanças na rede. O levantamento dos caminhos a serem considerados pode ser realizado, por exemplo, por meio do protocolo CSPF, descrito no Capítulo 2, ou uma solução que faça a tarefa dinamicamente. Neste trabalho, contudo, o foco será mantido na parte de escolha inteligente dos caminhos já definidos.

O problema em questão pode ser descrito segundo a modelagem do aprendizado por reforço. No presente caso, fizeram-se as seguintes definições:

- Ambiente: o AS em que o provedor de serviço deve encontrar um caminho para o tráfego do cliente.

- Agente: o roteador de borda da entrada da sessão RSVP-TE.

- Ações possíveis: a cada período de iteração do algoritmo, o roteador agente deve escolher se continua a transmissão por meio do caminho atual ou se muda para outro caminho. Portanto, com N caminhos disponíveis, tem-se N ações.

- Descrição dos estados: cada estado é definido por uma tupla de $N + 1$ números inteiros, sendo N números representando o nível de atraso de cada caminho possível, dentre L valores quantizados; e um número $c \in [0, \dots, N-1]$, representando o caminho atualmente utilizado para transmissão. A Figura 5.6 ilustra uma tupla geral representando um estado do ambiente. Desta forma, tem-se $N \cdot L^N$ estados possíveis.

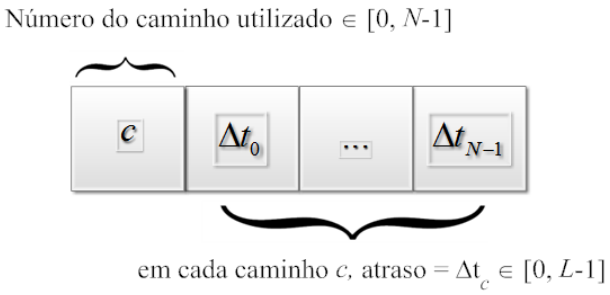

Figura 5.6 – Representação da tupla que define um estado.

- Recompensa: deve ser uma função inversamente proporcional à medida de atraso no último caminho escolhido para que atrasos menores gerem maiores ganhos. A ideia é que caminhos com atrasos menores sejam privilegiados.

No sistema descrito, o agente escolhe um dos N caminhos LSP para transmissão e é definido um espaço de tempo fixo para que a decisão seja avaliada com uma iteração periódica do algoritmo de aprendizado por reforço. O período de iteração deste algoritmo será chamado de T3, por ser o maior dos três importantes parâmetros de tempo. O atraso será medido por meio de pacotes de teste enviados periodicamente por cada um dos N caminhos considerados, do LER de entrada até o LER de saída da sessão RSVP-TE. O período de envio dos referidos pacotes será denominado de T1 < T3. Para

T3 deve ser escolhido um período grande o suficiente de forma a evitar instabilidade na rede, com muitas mudanças de caminho de transmissão em pouco tempo. Por outro lado, um valor grande demais pode fazer a convergência demorar excessivamente.

De acordo com o descritor de estados do ambiente apresentado anteriormente, é possível concluir que o estado pode se alterar mesmo que a ação permaneça fixa. Isso ocorre porque o nível de atraso em cada caminho se altera, o que faz com que N inteiros dos $N + 1$ da tupla de estado fiquem suscetíveis a mudanças. Já que a ação permanece fixa por um período que pode ser muito longo, introduz-se o período intermediário T2 (T1<T2<T3). A ideia é que a cada T2 segundos o algoritmo de aprendizado seja atualizado, não podendo, porém, ser alterado o caminho utilizado para transmissão do fluxo de dados.

Assim, pode-se dizer que a cada T2 segundos, o agente segue uma política fixa, em que se tem que obrigatoriamente repetir a ação anterior. Além de evitar trocas excessivamente frequentes, essa estratégia possibilita a atualização de alguns valores $Q(s, a_{\text{fixa}})$ dos estados s pelos quais o agente passar para não interromper o aprendizado. O uso dessas atualizações intermediárias em T2 será justificado por meio de um teste posteriormente.

Para a implementação adequada da solução proposta, um período T2 deve conter um número inteiro de períodos T1, e um período T3 precisa conter um número inteiro de períodos T2. Quando terminar o período T3, o algoritmo deve escolher se realizará *exploration* ou *exploitation*, segundo uma política *ε-greedy*, ou seja, escolhendo a ação que represente um maior ganho de acordo com o respectivo valor Q. Tanto o algoritmo *Q-learning* como o SARSA podem ser utilizados na solução. Posteriormente serão apresentados testes para verificar qual deles é o mais adequado.

5.4.1 Implementação em ambiente de simulação computacional

A implementação da solução foi realizada por meio do *software* OMNeT++, em sua versão 4.2.2. Para tal, foi necessário fazer modificações no código de alguns

módulos e classes do pacote INET 2.0 do *framework* de simulação. Como a estratégia proposta por este trabalho consiste em adaptação ao RSVP-TE, acrescentando-lhe um processo de inteligência artificial, as modificações realizadas se concentraram em módulos pertencentes ao modelo de roteador compatível com RSVP, o módulo "RSVP_LSR", mostrado na Figura 5.7. O controle de todas as sessões RSVP fica no submódulo "rsvp", e nele se encontra o processo responsável pelo algoritmo de aprendizado, que será chamado de processo RL. Este conta com *timers* periódicos para realizar as referidas ações a cada T1, T2 e T3 segundos.

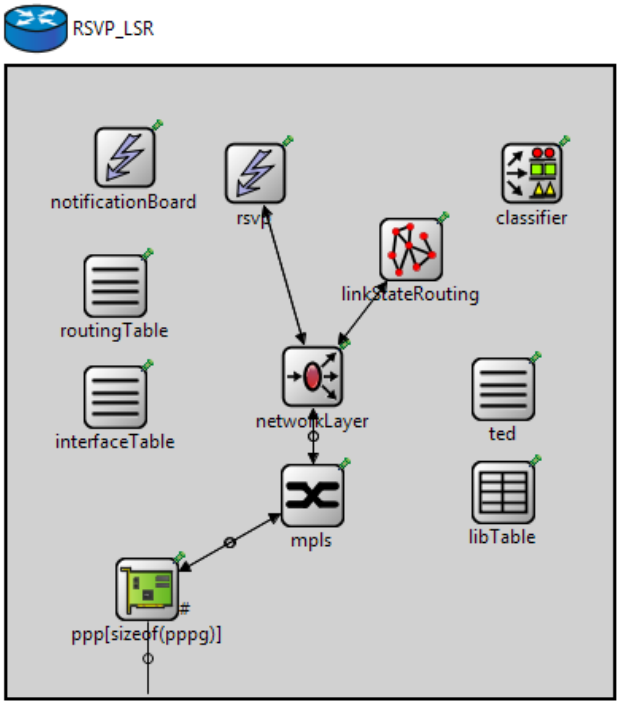

Figura 5.7 – Modelo de roteador RSVP do INET 2.0.

Cada sessão RL-RSVP-TE mantém um processo RL, que inicia os *timers* T1, T2 e T3 ao mesmo tempo. Após cada período T1, um pacote "RLtimer" é enviado para o módulo "networkLayer" para dar início ao envio dos pacotes de teste. O "RLtimer" carrega informações da sessão, como seu número de identificação e os de seus caminhos LSP configurados. Quando o submódulo "ip" – dentro do módulo "networkLayer" – recebe o "RLtimer", procede à criação de um datagrama de teste para cada caminho da sessão, contendo poucos bytes. Cada um desses datagramas é repassado ao módulo "mpls" juntamente com a identificação do respectivo caminho pelo qual será enviado.

Neste instante a tabela LIB é consultada para marcar os pacotes com o rótulo MPLS correspondente a cada caminho. A partir daí, os pacotes estão prontos para serem transmitidos e medir o atraso.

Após a chegada de um pacote de teste no LER *receiver* da sessão, o tempo entre o seu envio e seu recebimento é registrado e utilizado para atualização da estimativa de atraso do respectivo caminho. Foi utilizado um filtro adaptativo LMS (*Least Mean Squares*) [40] para manter a estimativa do atraso de cada caminho:

$$W(n+1) = W(n) + \mu\big(\Delta t(n+1) - W(n)\big), \tag{5.1}$$

em que $W(n)$ é a estimativa de atraso no instante n, $\Delta t(n)$é o atraso observado no instante n e μ é o passo de atualização do filtro. Essas estimativas W permanecem sendo atualizadas a cada T1 segundos, até que quando se passam T2 segundos e o LER *receiver* faz uma compilação da estimativa atual de todos os caminhos em um pacote de controle do RSVP-TE, que é então enviado ao LER *sender*. Na implementação com OMNeT++, o modelo desse pacote foi construído a partir do modelo do pacote "path_notify" e, por essa razão, será chamado de "RL_notify". Após esse processo, o LER *receiver* zera as estimativas de atraso para renovar a leitura do estado do ambiente.

Quando um LER *sender* recebe o "RL_notify", registra os novos valores de estimativa do atraso de cada caminho e procede à atualização do *Q-learning*. Se for o fim de um período de T2 segundos, esta atualização se dá com a política limitada de modo que a ação escolhida seja obrigatoriamente a de permanecer transmitindo pelo mesmo caminho. Caso o *timer* esgotado no momento seja o de T3, escolhe-se entre *exploration* e *exploitation*: com probabilidade ε, escolhe-se uma ação aleatoriamente, ou seja, um caminho qualquer para a transmissão. Com probabilidade $1 - \varepsilon$, a melhor ação deve ser escolhida e o caminho com maior valor Q para o estado atual passa a ser utilizado para transmissão.

A estratégia foi ilustrada por meio de duas figuras, explicitando cada um dos instantes de referência T1, T2 e T3. A Figura 5.8 representa os acontecimentos após o

instante T1, o envio dos pacotes de teste até o LER *receiver*, mostrando a trajetória em cada caminho até a atualização das estimativas de atraso. Já na Figura 5.9 pode-se ver a o envio do LER *receiver* até o LER *sender* de um pacote informando a estimativa de atraso de cada um dos caminhos disponíveis. Quando da chegada destes pacotes ao LER *sender*, a referida figura mostra a atualização do algoritmo de aprendizado por reforço, ocorrendo de forma diferente para o caso do timer estourado ser T2 ou T3.

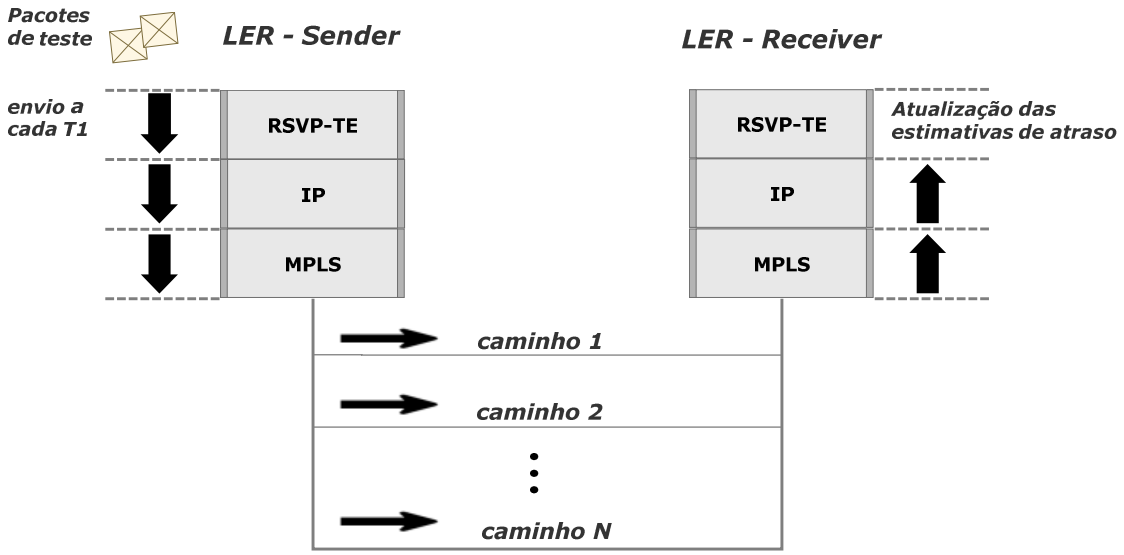

Figura 5.8 – Envio dos pacotes de teste por cada caminho para atualização das estimativas de atraso.

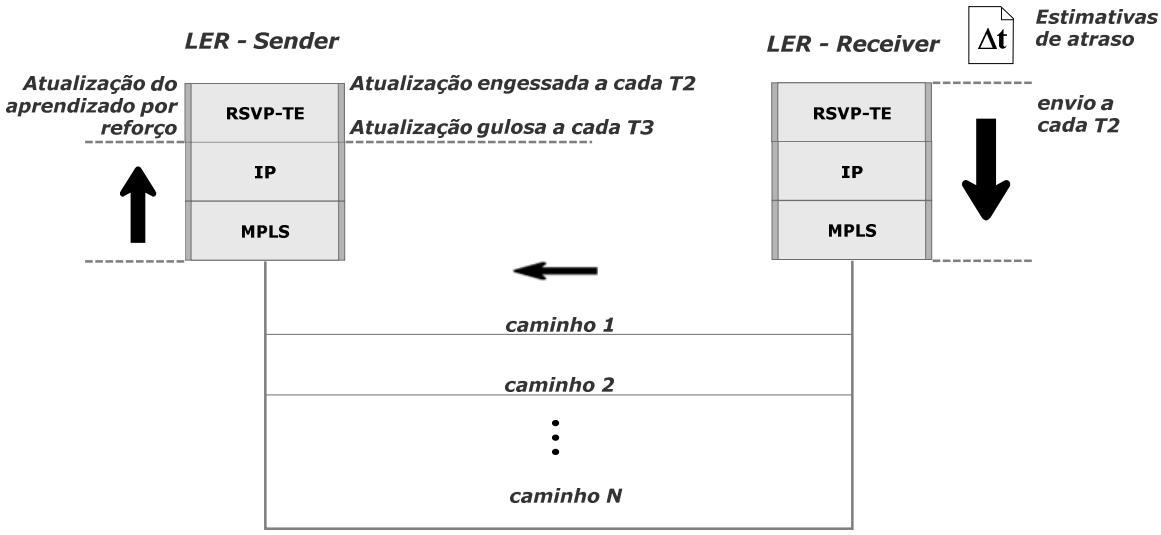

Figura 5.9 – Retorno do pacote com as estimativas de atraso para o LER *sender*.

5.4.2 Níveis de atraso e definição das recompensas

Para a quantização do atraso em L níveis foi escolhida uma estratégia com dois parâmetros: Δt_{min} e Δt_{max}. Se a estimativa de atraso W_c, do caminho c, for menor que Δt_{min}, o atraso quantizado Δt_c fica no menor nível possível. Caso a estimativa esteja acima de Δt_{max}, o atraso quantizado recebe o valor referente ao maior nível possível. Assim, esses parâmetros devem ser escolhidos de para delimitar o intervalo em que há maior variação do atraso, de forma que atrasos menores que Δt_{min} sejam os menos prejudiciais possíveis ao desempenho do sistema e atrasos maiores que Δt_{max} sejam os mais indesejados. As expressões (5.2) e (5.3) explicitam os valores definidos para o nível de atraso quando a estimativa não fica no intervalo entre os dois atrasos de referência.

$$0 \leq W_c \leq \Delta t_{min} \Rightarrow \Delta t_c = 0 \tag{5.2}$$

$$W_c > \Delta t_{max} \Rightarrow \Delta t_c = L - 1 \tag{5.3}$$

A Figura 5.10 mostra a ideia de divisão genérica do atraso em níveis discretos entre um valor máximo e um mínimo. A divisão do atraso em níveis discretos ocorre em razão da necessidade de definir estados finitos e discretos para utilização dos algoritmos de aprendizado por reforço de diferença temporal expostos no Capítulo 3, relativamente simples [20].

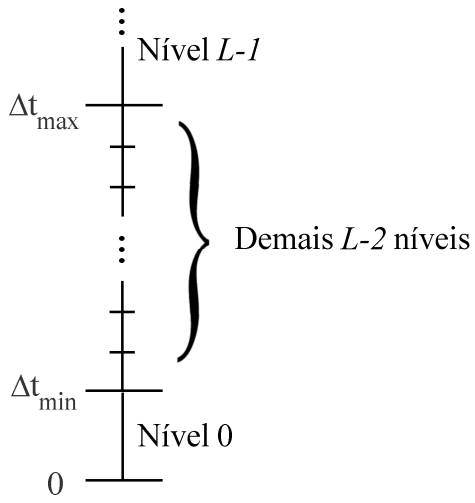

Figura 5.10 – Modelo para quantização da estimativa de atraso.

Com as diretrizes estabelecidas pelas expressões (5.2) e (5.3), os $L-2$ níveis restantes são distribuídos dentro do intervalo $(\Delta t_{min}, \Delta t_{max}]$. Pode-se pensar em várias maneiras de fazê-lo. Uma delas é a divisão uniforme do intervalo $\Delta t_{min} < W_c \leq \Delta t_{max}$. Para isso, o atraso quantizado pode ser calculado da seguinte maneira:

$$\Delta t_c = \left\lfloor 1 + (L-2) \cdot \left(\frac{W_c - \Delta t_{min}}{\Delta t_{max} - \Delta t_{min}} \right) \right\rfloor , \tag{5.4}$$

em que o operador $\lfloor \quad \rfloor$ representa a função *chão*, para obtenção da parte inteira da expressão. No intervalo intermediário considerado, a fração presente na equação (5.4) varia de 0 a 1. Multiplicando-se a referida fração pelo termo $(L-2)$, tem-se um resultado variando no intervalo $(0, L-2]$. Após a soma com 1, o resultado pretendido é obtido, variando dentro de $(1, L-1]$.

Pode ser conveniente realizar a divisão dos níveis de forma não uniforme, se a rede tiver maior variação de atraso em níveis mais altos ou baixos. Neste trabalho serão testados dois métodos alternativos: quantização logarítmica e exponencial, com as expressões respectivas (5.5) e (5.6), onde f é parâmetro de compressão. A Figura 5.11 exibe três intervalos quantizados de atraso com $L = 9$ para exemplificação: (a) de maneira uniforme; (b) escala logarítmica com $f = 12$; (c) escala exponencial com

$f = 2{,}5$. Os respectivos valores de f foram escolhidos de forma a que os níveis tenham uma gradação semelhante para o mesmo L.

$$\Delta t_c = \left\lfloor 1 + (L-2) \cdot \frac{1}{ln\,(1+f)} \cdot ln\left(1 + f \cdot \left(\frac{W_c - \Delta t_{min}}{\Delta t_{max} - \Delta t_{min}}\right)\right)\right\rfloor \tag{5.5}$$

$$\Delta t_c = \left\lfloor 1 + (L-2) \cdot \left(\frac{e^{f \cdot \left(\frac{W_c - \Delta t_{min}}{\Delta t_{max} - \Delta t_{min}}\right)} - 1}{e^f - 1}\right)\right\rfloor \tag{5.6}$$

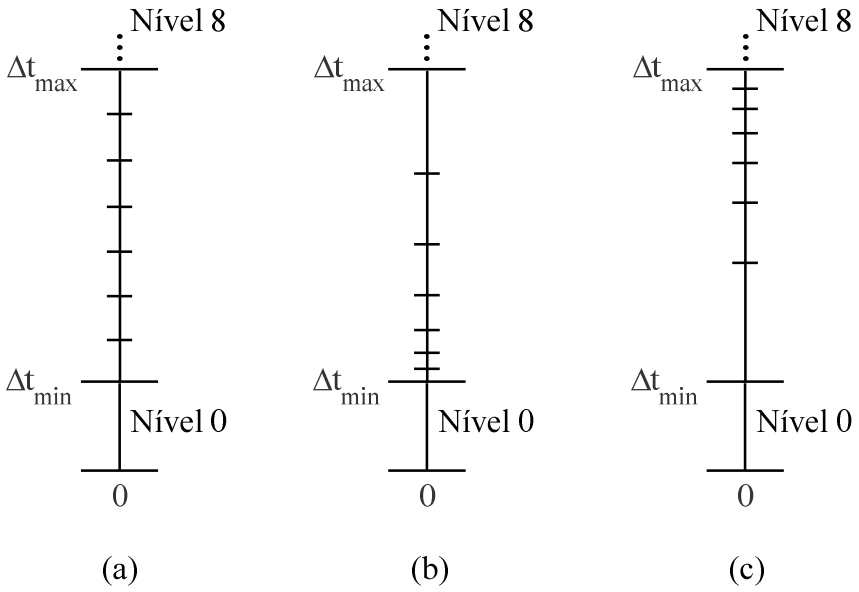

Figura 5.11 – Tipos de quantização do atraso: (a) uniforme; (b) logarítmica; (c) exponencial.

Quanto maior o valor L de níveis na divisão, mais estados se tem no ambiente. Com mais estados, tem-se um ambiente mais bem descrito, porém mais complexo. Isso quer dizer que o tempo de convergência do algoritmo de aprendizado por reforço tende a aumentar, uma vez que, com mais estados, dificulta-se a condição de convergência segundo a qual deve haver atualização dos valores $Q(s, a)$ por um número grande de vezes [24].

Tendo realizado a quantização dos níveis de atraso, é necessário escolher como será atribuída a recompensa às ações. Uma primeira ideia, que será testada, é

simplesmente representar a recompensa de forma contínua, como o oposto da estimativa do atraso do caminho utilizado para transmissão:

$$r_k = -W_{c,k}\ , \tag{5.7}$$

em que $W_{c,k}$ representa a estimativa de atraso do caminho *c* no passo k do algoritmo de aprendizado por reforço. Desta maneira, quanto menor o atraso, maior a recompensa. Como o os valores $Q(s,a)$ são nulos no início, acontece uma pequena *exploration* cada vez que um estado novo é visitado.

Pode-se testar também um esquema de quantização análogo ao do nível do atraso, atribuindo uma recompensa inversamente proporcional a cada nível Δt_c. Mantendo a ideia de recompensa negativa, esta pode ser definida após ser dado um passo de *P* unidades abaixo do 0 para cada nível, a medida que este aumentar. Ou seja:

$$r_k = -P \cdot (\Delta t_{c,k} + 1) \tag{5.8}$$

Por exemplo, supõe-se a verificação do nível $\Delta t_c = 0$ após ter sido escolhido o caminho c. Isso acarretaria a maior recompensa quantizada $r = -P$. No caso do maior nível de atraso, $\Delta t_c = L - 1$, seria obtida a recompensa mais negativa $r = -P \cdot L$. A definição de um passo é importante pois pode facilitar a convergência, evitando valores $Q(s,a)$ diferentes excessivamente próximos. Para uma quantização análoga de recompensas com valores positivos, pode-se seguir a equação (5.9), que também será objeto de teste.

$$r_k = P \cdot (L - \Delta t_{c,k}) \tag{5.9}$$

Sutton e Barto explicitam em [20] que uma boa condição inicial para encorajar a exploração é que os valores $Q(s,a)$ iniciais sejam razoavelmente maiores do que os valores ótimos $Q^*(s,a)$. Assim, sempre que um novo estado for visitado por *N* vezes, há uma garantia de que todas as *N* ações serão escolhidas ao menos uma vez nesse estado. Para adequar a solução proposta a essas condições, foram escolhidos preferencialmente valores $Q(s,a)$ iniciais nulos e recompensas negativas.

5.5 TESTES E RESULTADOS

Nesta seção são apresentados os resultados de testes realizados para ajuste de boa parte dos muitos parâmetros envolvidos na solução. Para isso foi montado no OMNeT++ um cenário simples com três caminhos previamente configurados e a possibilidade de injetar tráfego no decorrer da simulação.

5.5.1 Cenário para testes

Uma rede simples foi modelada por meio da linguagem descritora de topologia no OMNeT++: são 7 roteadores LSR que implementam a solução RL-RSVP-TE e 7 *hosts* capazes de gerar tráfego UDP com algumas possibilidades de distribuições para o tempo entre pacotes. A Figura 5.12 mostra o cenário com uma sessão RSVP configurada, com LSR1 como LER *sender* e LSR5 como LER *receiver*. Entre eles, outros 5 LSRs que formam os nós de 3 caminhos já configurados explicitamente, como indicado: o caminho 1, formado pelos nós LSR1, LSR3, LSR7 e LSR5; o caminho 2, formado pelos nós LSR1, LSR4 e LSR5; e o caminho 3, formado pelos nós LSR1, LSR2, LSR6 e LSR5. Posteriormente, a escolha automática de caminhos seguindo certos critérios de desempenho pode ser explorada, o que traria maior complexidade à solução.

Todos os *links* possuem a mesma capacidade de 1 Mbps e atraso médio de propagação de 10 μs. São utilizados enlaces com relativamente baixa capacidade e fluxos com taxas não muito alta como abstração de modo que as simulações não durem um tempo excessivo. Os *hosts* 1 e 2 devem enviar um fluxo com destino ao *host* 3, ao passo que gerador 1 e gerador 2 permitem a injeção de tráfego adicional nos caminhos 2 e 3 em algum instante da simulação.

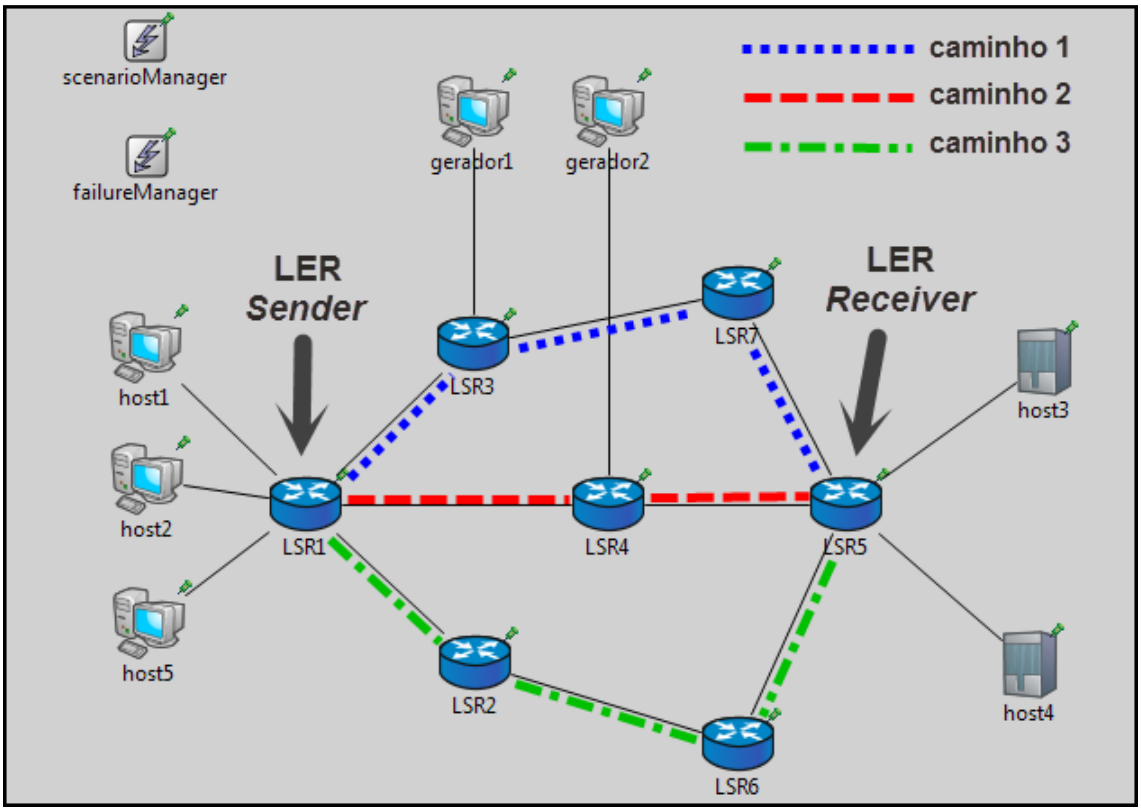

Figura 5.12 – Cenário de testes com 3 caminhos configurados.

5.5.2 Teste 1: algoritmo de aprendizado por reforço

O primeiro teste a ser mostrado serve para uma escolha fundamental ao desempenho da solução: qual o algoritmo de diferença temporal mais adequado para a atualização do aprendizado nos períodos T2 e T3? Os candidatos são *Q-learning* e SARSA. Primeiramente escolhem-se alguns parâmetros: T1 = 5 ms , T2 = 1 s e T3= 10 s. Para a escolha de μ, o passo do filtro adaptativo que estima os atrasos, foram feitas medidas das estimativas para verificar a convergência destas para um valor de $\mu = 0{,}7$. O resultado pode ser observado na Figura 5.13, comprovando que o valor escolhido permite uma convergência confortável a cada ciclo de T2 segundos.

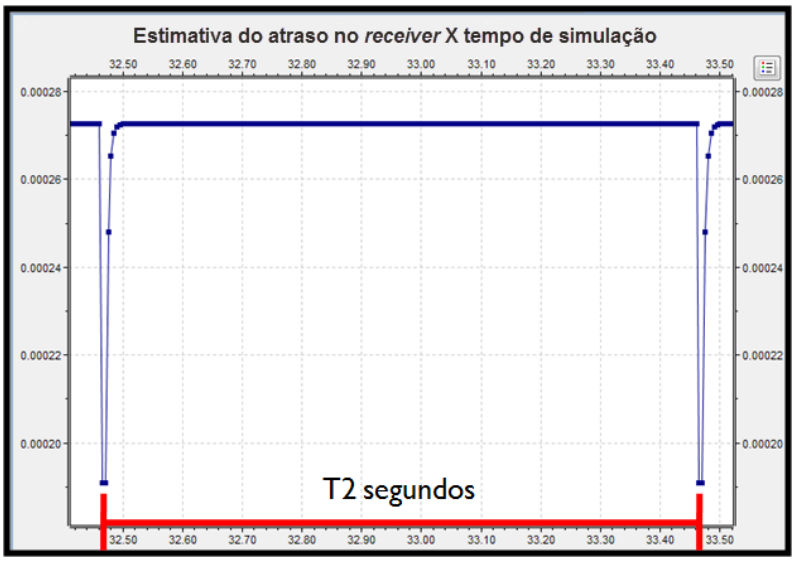

Figura 5.13 – Teste de convergência da estimativa de atraso no *receiver* com $\mu = 0,7$.

Para o algoritmo de aprendizado por reforço, foram escolhidos $\alpha = 0,7$ e $\gamma = 0,5$, para que se tenha, respectivamente, uma convergência melhor com um bom peso para novas recompensas e um desconto razoável para os ganhos futuros. O valor para o parâmetro ε será 7 %, para que se realize uma mínima *exploration* ε-*greedy*.

Foi definido um fluxo UDP de aproximadamente 200 kbps com destino ao *host* 3 partindo de cada um dos *hosts* 1 e 2, com tempo entre pacotes com distribuição normal. Estes serão os fluxos de referência do teste. Programou-se também o começo de fluxos UDP partindo de gerador 1 e gerador 2 com destino a *host* 4 para os 2000 segundos de simulação, de modo a congestionar os caminhos 1 e 2 para os nossos fluxos de referência. Foram feitas medidas com estas configurações de tráfego de modo a permitir saber quais os melhores valores de Δt_{max} e Δt_{min}. O resultado pode ser visto na Figura 5.14, que mostra o atraso efetivo (com transmissão do fluxo de dados) em cada caminho em momentos próximos do início da transmissão dos geradores 1 e 2.

Neste trabalho define-se o **atraso efetivo**: medida que representa a estimativa de atraso fim a fim de um caminho somente quando ele é escolhido para carregar o tráfego de referência. Espera-se que a solução desenvolvida neste trabalho reaja principalmente à variação do atraso efetivo e não à do atraso instantâneo em cada caminho. Se este

fosse o caso, seria possível cair em casos de instabilidade como ocorre em soluções que alteram o peso dos enlaces OSPF de acordo com o atraso, por exemplo.

Pela Figura 5.14, parece ser razoável a escolha de $\Delta t_{min} = 1{,}5\ ms$ e $\Delta t_{max} = 7{,}5\ ms$, uma vez que as grandes variações ocorrem neste intervalo. Será utilizada a quantização uniforme do intervalo entre os dois valores, bem como a recompensa será quantizada de acordo com a expressão (5.6). Inicialmente, será utilizada uma divisão dos valores de atraso em $L = 5$ níveis. Essa quantidade significa um espaço de $3 \cdot 5^3 = 375$ estados possíveis (vide Figura 5.6).

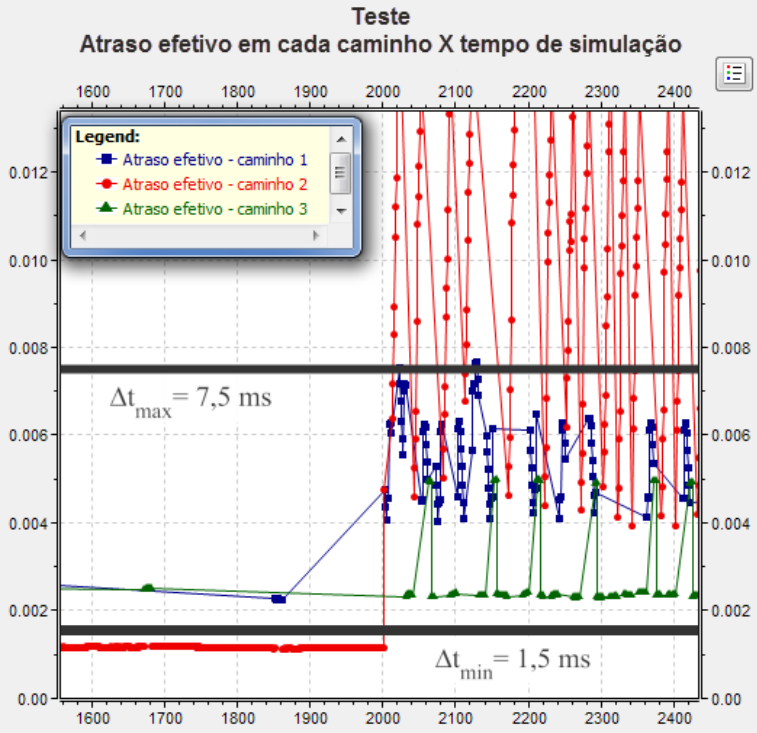

Figura 5.14 – Teste para averiguar os níveis de atraso.

Todos os referidos parâmetros foram compilados na Tabela 5.2 para melhor compressão geral do sistema em que o teste foi realizado. Os fluxos de pacotes presentes no teste, suas características e tempo de transmissão estão registrados na Tabela 5.3. Após a configuração desses parâmetros no ambiente de simulação, é possível iniciar o teste comparativo entre o uso dos algoritmos *Q-learning* e SARSA

nos momentos de atualização T2 e T3. Os dois algoritmos utilizam os mesmos parâmetros, como estudado na Seção 3.2.

Tabela 5.2 – Parâmetros do sistema para o primeiro teste

Parâmetro	**Valor**
T1	5 *ms*
T2	1 *s*
T3	10 *s*
$\boldsymbol{\mu}$	0,7
$\boldsymbol{\alpha}$	0,7
$\boldsymbol{\gamma}$	0,5
$\boldsymbol{\varepsilon}$	7 %
$\Delta\boldsymbol{t_{min}}$	1,5 *ms*
$\Delta\boldsymbol{t_{max}}$	7,5 *ms*
$\boldsymbol{L}$	5 níveis
Observações	
Atraso quantizado uniformemente	
Recompensa quantizada e negativa nos moldes da equação $(\mathbf{5.6})$, com $\boldsymbol{P} = 20$	

Tabela 5.3 – Fluxos de pacotes configurados para o primeiro teste

Fluxo	**Origem**	**Destino**	**Descrição**	**Distribuição do tempo entre pacotes**	**Tempo de transmissão**	
					Início	**Fim**
Tráfego de referência	*Hosts* 1 e 2	*Host* 3	Dois fluxos de ~200 kbps	Normal (0.01 *s*, 0.000001 *s*)	1,5 *s*	–
Tráfego injetado 1	Gerador 1	*Host* 4	Fluxo de ~400 kbps	Normal (0.0049 *s*, 0.000001 *s*)	2000 *s*	–
Tráfego injetado 2	Gerador 2	*Host* 4	Fluxo de ~460 kbps	Normal (0.00429 *s*, 0.000001 *s*)	2000 *s*	–

***Q-learning* nas atualizações de T2 e T3 segundos**

Primeiramente foi feita a simulação do cenário descrito pelas Tabelas 5.2 e 5.3 e pela Figura 5.12 com o algoritmo *Q-learning* (equação (3.8)) sendo utilizado tanto para as atualizações ocorridas após fim do *timer* T2 quanto para após T3. Após a simulação de 10000 segundos, temos como resultado a Figura 5.15, que indica o caminho escolhido após cada período de T3 segundos no início da simulação, ou seja, antes da entrada dos fluxos injetados. Já a Figura 5.16 mostra o atraso efetivo de cada caminho no mesmo período e explica a anterior. Como o atraso efetivo do caminho 2 é muito inferior ao dos outros, ele dominou as escolhas para transmissão, após um pequeno período inicial de aprendizado. Após algum tempo, na mesma Figura 5.15, podem-se ver resultados de *exploration*, com algumas mudanças isoladas de caminho.

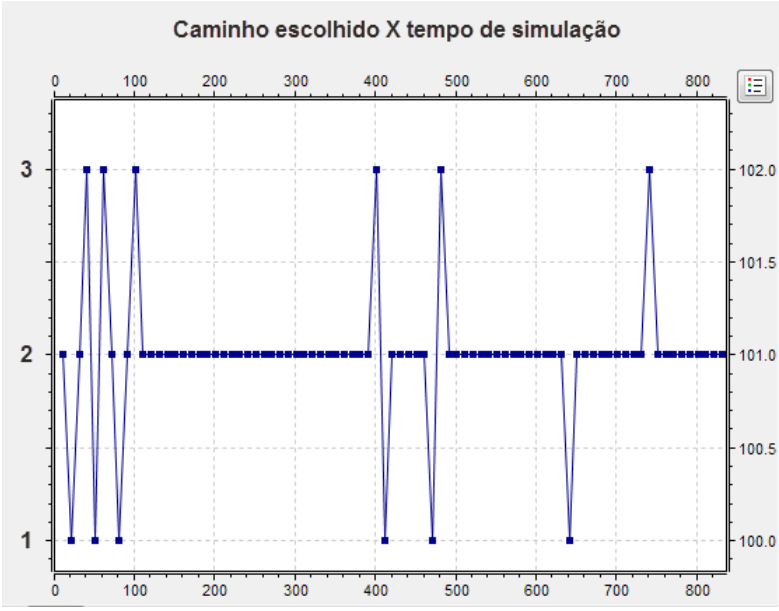

Figura 5.15 – Escolha dos caminhos ao longo do início do primeiro teste.

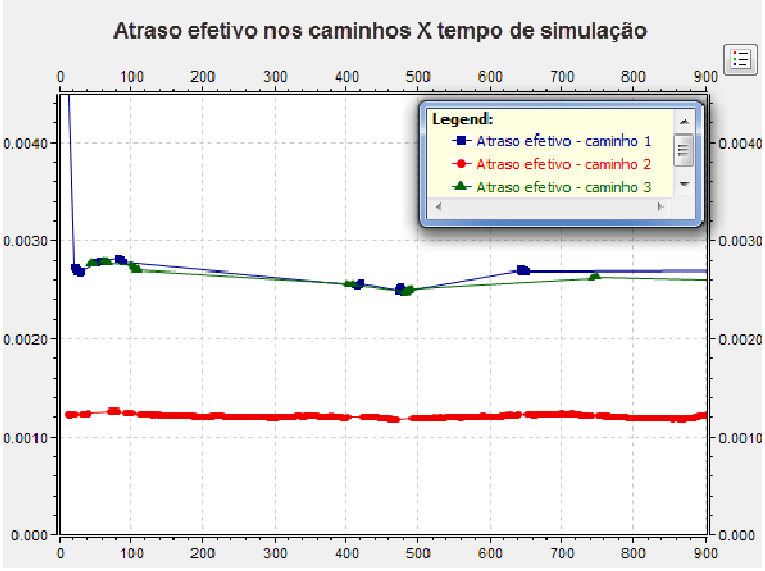

Figura 5.16 – Atraso efetivo nos caminhos no início do primeiro teste.

O estado com identificação $s = 125$, que representa o menor nível de atraso possível nos três caminhos após ter escolhido o caminho 1 para transmissão, foi o mais visitado nos primeiros 2000 segundos de simulação, identificado em 91 % das verificações desse período. A Figura 5.17 mostra os valores $Q(s, a)$ para $s = 125$, evidenciando a superioridade do valor $Q(125, \text{caminho } 2)$ a partir de pouco mais de

110 segundos de simulação. Essa é a razão de a ação mais escolhida nesse período inicial ser a transmissão por meio do caminho 2.

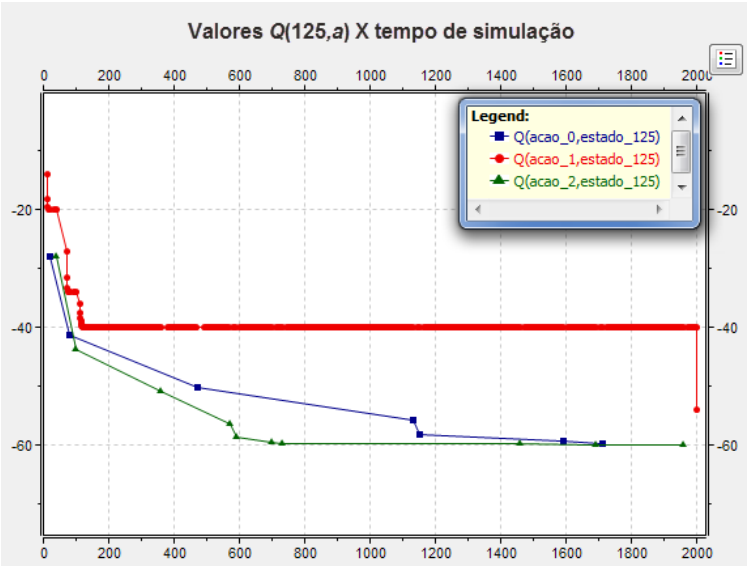

Figura 5.17 – Evolução dos valores Q do estado 125 para cada ação até tempo 2000s.

Após a entrada dos fluxos de pacotes injetados pelos geradores 1 e 2, espera-se que o perfil de atraso observado mude radicalmente. As ações tomadas pelo agente podem ser observadas na Figura 5.18, indicando que, com a entrada de mais tráfego, num período inicial de aproximadamente 500 segundos, a escolha oscila entre os três caminhos. Depois disso, a oscilação fica limitada aos caminhos 2 e 3 até o restante da simulação, o que não era o comportamento desejado.

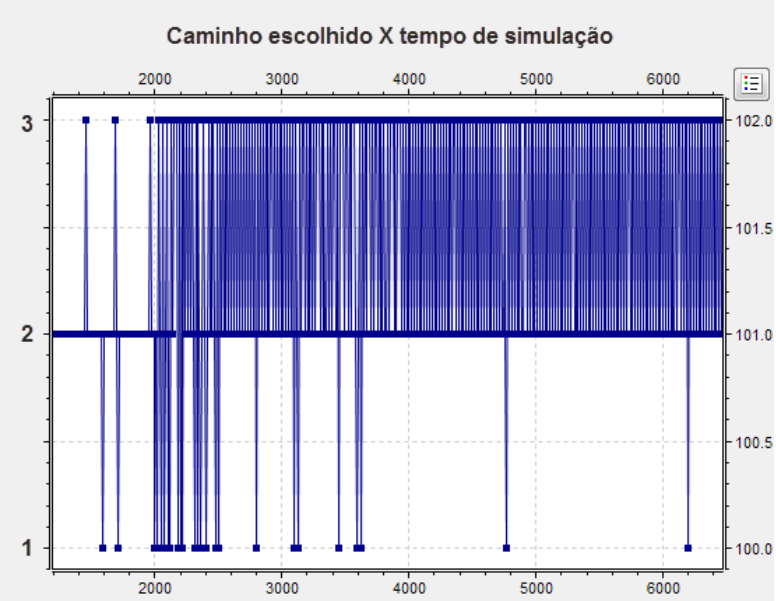

Figura 5.18 – Escolha dos caminhos após novos fluxos injetados na rede.

Pode-se ver o atraso efetivo nos 3 caminhos nesses instantes na Figura 5.19. Embora o atraso efetivo do caminho 3 se mostre menor que o dos outros a partir dos 2000 segundos de simulação, este caminho não foi o priorizado. Como foi verificado anteriormente que a quantização escolhida é satisfatória para representar o ambiente no caso deste teste, o motivo desse resultado deve ser investigado.

É possível verificar o comportamento geral dos valores $Q(s, a)$ na Figura 5.20, pela qual notam-se os dois momentos distintos, antes e depois dos 2000 segundos de simulação. Antes desse instante, é vista uma convergência suave dos valores $Q(s, a)$ envolvidos, porém depois, observa-se até certa instabilidade entre grande parte deles. Em razão da maior variação de tráfego, um maior número de estados passa a ser contemplado. Isso ocorre em razão da característica do *Q-learning* de sempre levar em conta como componente do estado futuro no cálculo das atualizações um valor $Q(s', a')$ referente à melhor ação a' possível, e não àquela de fato escolhida.

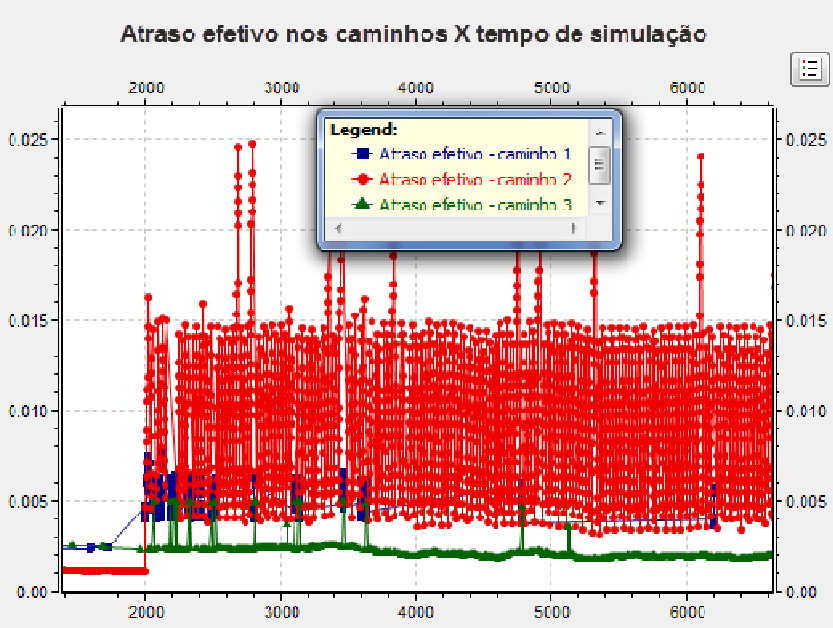

Figura 5.19 – Atraso efetivo nos caminhos após injeção de tráfego.

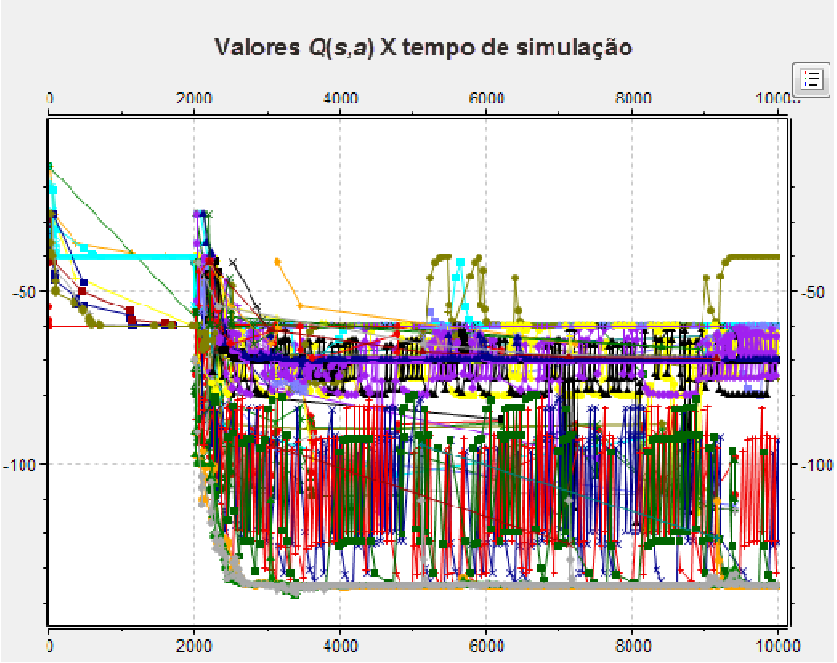

Figura 5.20 – Evolução de todos os valores $Q(s, a)$.

Na solução proposta, o agente fica preso na mesma ação durante as atualizações de T2 segundos, ou seja, segue uma política diferente da gulosa. O trabalho [41], que propôs o algoritmo SARSA, mostra que nessas situações se superestima o retorno esperado, o que gera problemas de convergência mesmo em estados avançados. No caso

específico, essas atualizações são $\frac{T3}{T2} = 10$ vezes mais comuns. Para verificar esse aspecto, será feito o mesmo teste utilizando o algoritmo SARSA nas atualizações de T2 segundos, pois ele leva em conta a política adotada pelo agente.

SARSA nas atualizações de T2 segundos

Foi mantido o cenário descrito pela Figura 5.12 e pelas Tabelas 5.2 e 5.3. Rodou-se a simulação durante o mesmo período de antes, 10000 segundos, porém com a utilização do algoritmo SARSA (equação (3.14)) nas atualizações dos valores $Q(s, a)$ a cada T2 segundos, com a política engessada. Porém, manteve-se o *Q-learning* para o aprendizado dos períodos T3. Até os 2000 segundos de simulação, com os níveis de tráfego iniciais, o comportamento do agente foi bastante parecido com o do caso anterior, levando aproximadamente 120 segundos para se estabilizar. Porém, a partir da injeção de tráfego após esse instante, é possível ver na Figura 5.21 que o caminho escolhido passou a ser majoritariamente o caminho 3, como desejado, já que neste não há nenhum outro fluxo de pacotes sendo injetado. As mudanças esporádicas de caminhos, depois da estabilização inicial de 360 segundos, ocorrem principalmente por *exploration*, encontrando-se dentro do número esperado de 7 %. A Figura 5.22 mostra o atraso efetivo em cada caminho em momentos antes e após a injeção dos novos fluxos de tráfego aos 2000 segundos de simulação. De fato o atraso efetivo do caminho 3 se mostra mais vantajoso para o agente, que logo aprende a vincular esse melhor caminho aos estados em que se encontra.

Observando a Figura 5.23, pode-se comparar a evolução dos valores $Q(s, a)$ no caso de utilização do SARSA para atualizações em T2 com o uso do *Q-learning* para o mesmo fim (Figura 5.20). No caso do SARSA não há oscilação tão grande nos valores, e se observa uma convergência mais rápida e suave.

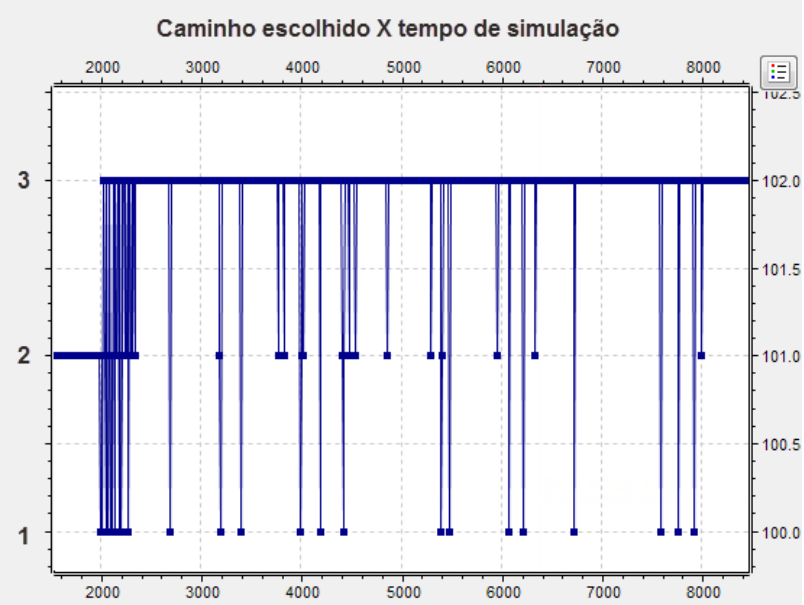

Figura 5.21 – Escolha do caminho após tráfego injetado, com SARSA nas atualizações T2.

Figura 5.22 – Atraso efetivo nos caminhos, com SARSA nas atualizações T2.

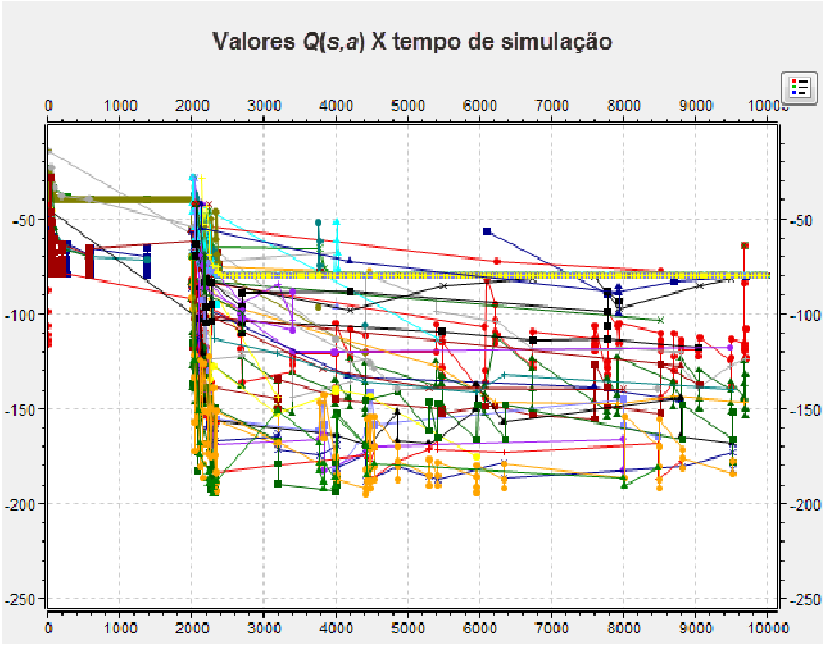

Figura 5.23 – Evolução de todos os valores $Q(s,a)$, SARSA nas atualizações T2.

Adicionalmente, foi comparado o desempenho da solução com três combinações de algoritmos:

1. *Q-learning* tanto nas atualizações de período T2 quanto nas de T3, caso referido como *Q-learning* puro;

2. SARSA nas atualizações de período T2 e Q-learning nas de T3, combinação referida como híbrida;

3. SARSA atualizações de período T2 quanto nas de T3, caso referido como SARSA puro.

A Tabela 5.4 traz o resultado da confrontação dessas três combinações quanto à escolha da ação ótima após a injeção do tráfego na rede. A atualização híbrida apresenta desempenho superior aos outros, embora semelhante ao caso de SARSA puro. Por essa razão, a combinação híbrida será utilizada preferencialmente neste trabalho para as demais simulações. Ressalta-se que o índice de ocorrência de *exploration* é aproximadamente 7 % em razão do parâmetro ε, portanto é esperado que algo em torno

dessa porcentagem das ocorrências não seja da ação ótima. Portanto, um índice de 92,5 % de escolha da ação ótima é bastante próximo do teórico esperado de 93 %.

Tabela 5.4 – Comparação entre combinações de algoritmos em T2 e T3

Combinação	**Porcentagem de escolha da ação ótima após entrada do tráfego**
Q-learning puro	49,3 %
Híbrida	92,5 %
SARSA puro	91,3 %

5.5.3 Teste 2: importância das atualizações em T2

Realizar atualizações com a política diferente em T2, sem alteração de caminho escolhido, é importante porque, mesmo com a ação fixa durante um tempo maior, os estados podem mudar. Portanto podem se atualizar alguns valores $Q(s,a)$ de estados diferentes**,** ajudando a convergência. Para comprovar o ganho no desempenho da solução com o uso dessas atualizações engessadas intermediárias, foi feita uma simulação sem elas para comparação. Nas simulações, foram utilizados o cenário representado na Figura 5.12, os mesmos fluxos de tráfego da Tabela 5.3 e os parâmetros da Tabela 5.2, com exceção do valor de T2, que não é aplicado no caso.

A Figura 5.24 ilustra o resultado do caso sem o uso das atualizações em T2. Podem-se observar tempos de estabilização maiores: sem as atualizações em T2, a adequação inicial dura 280 segundos, e com elas, 120 segundos. Para se adequar ao tráfego entrante aos 2000 segundos, a versão sem atualizações em T2 levou 480 segundos, em contraponto aos 360 segundos apresentados pelo caso com atualizações em T2. A Tabela 5.5 traz, além dos referidos tempos, a porcentagem de vezes em que a ação ótima (transmissão pelo caminho 3) foi escolhida após a entrada do tráfego, comprovando que traz benefícios a realização de atualizações intermediárias. Novamente, ressalta-se que o índice de 92,5 % de escolha da ação ótima é bom

resultado comparado aos 93 % esperados – já que aproximadamente 7 % da escolhas são *exploration*.

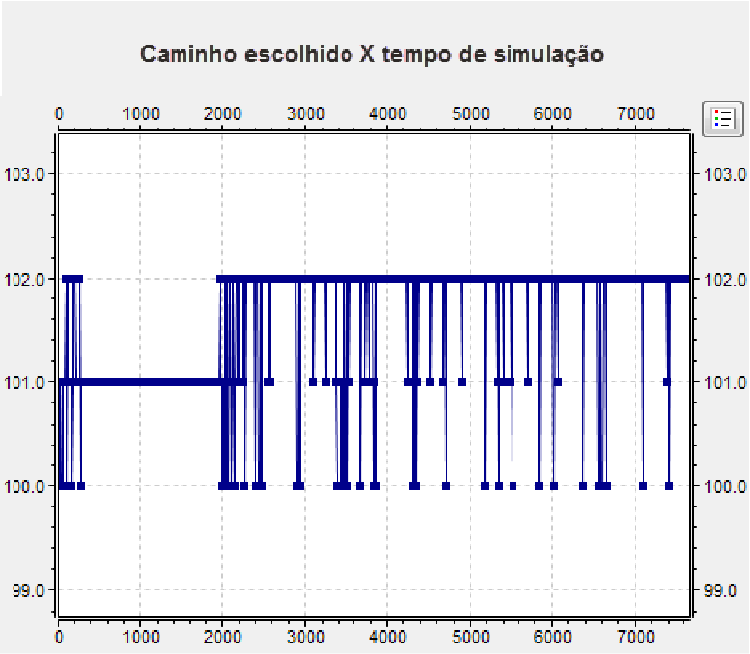

Figura 5.24 – Caminho escolhido durante a simulação sem atualizações engessadas em T2.

Tabela 5.5 – Comparação entre realização ou não de atualizações engessadas em T2

Realização de atualizações intermediárias em T2	**Porcentagem de escolha da ação ótima após entrada do tráfego**	**Tempo de estabilização inicial**	**Tempo de estabilização após entrada do tráfego**
Sim	92,5 %	120 *s*	360 *s*
Não	88,6 %	280 *s*	480 *s*

5.5.4 Teste 3: influência do parâmetro α

O terceiro teste tem o objetivo de verificar a influência do parâmetro α no desempenho da solução. Para tal, foi utilizado novamente o cenário descrito pela Figura 5.12 e com os fluxos configurados de acordo com a Tabela 5.3. Os demais parâmetros

utilizados neste teste são mostrados na Tabela 5.6, com a única diferença para o caso anterior para o valor $\alpha = 0{,}3$.

Tabela 5.6 – Parâmetros do sistema para verificar a influência da taxa de aprendizado

Parâmetro	**Valor**
T1	5 *ms*
T2	1 *s*
T3	10 *s*
$\boldsymbol{\mu}$	0,7
$\boldsymbol{\alpha}$	0,3
$\boldsymbol{\gamma}$	0,5
$\boldsymbol{\varepsilon}$	7 %
$\Delta \boldsymbol{t_{min}}$	1,5 *ms*
$\Delta \boldsymbol{t_{max}}$	7,5 *ms*
$\boldsymbol{L}$	5 níveis
Observações	
Atraso quantizado uniformemente	
Recompensa quantizada e negativa nos moldes da equação $(\mathbf{5.6})$, com $\boldsymbol{P} = 20$	

A Figura 5.25 mostra os caminhos escolhidos para transmissão a cada T3 segundos nos instantes iniciais da simulação. Percebe-se um tempo inicial de exploração de aproximadamente 200 segundos, o dobro do caso anterior (Figura 5.14). Já após a injeção de tráfego na rede, pode-se observar na Figura 5.26 que a escolha do caminho somente se estabiliza aproximadamente 1000 segundos depois, quase o triplo do caso com $\alpha = 0{,}7$.

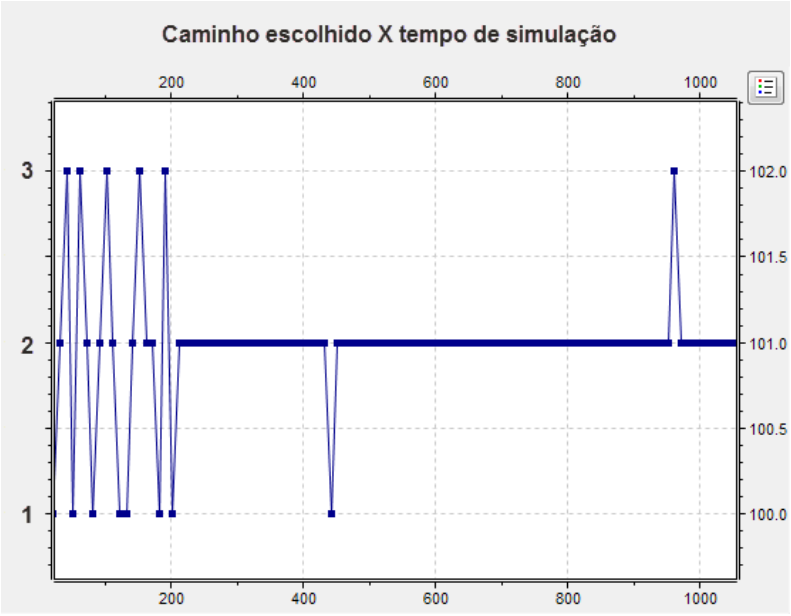

Figura 5.25 – Caminho escolhido no início da simulação, $\alpha = 0,3$.

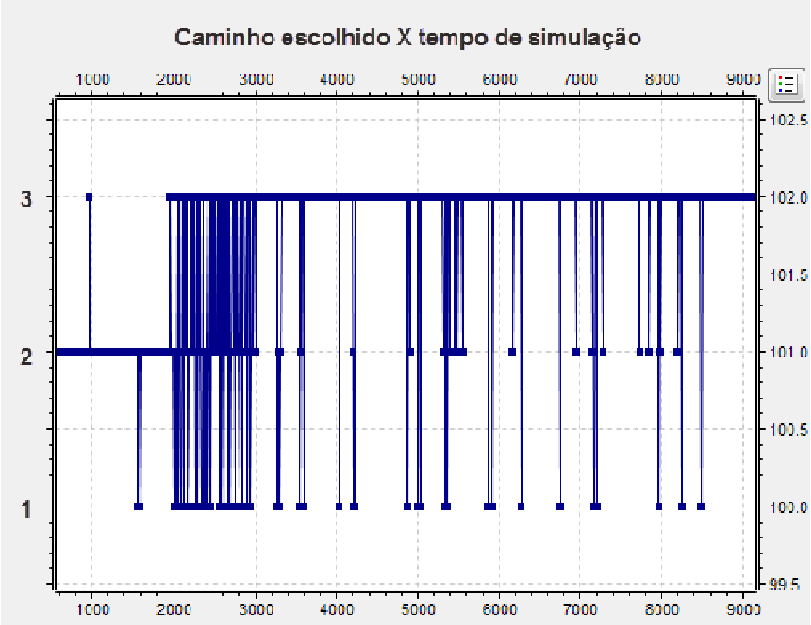

Figura 5.26 – Caminho escolhido após a entrada de mais tráfego na rede, $\alpha = 0,3$.

Já era esperada uma convergência mais rápida com maior α, já que representa o peso que as novas recompensas e estimativa do ganho futuro têm no cálculo dos valores $Q(s,a)$. A Tabela 5.7 mostra o desempenho da solução com os dois valores para a taxa

de aprendizado, comparando tanto os tempos de estabilização já comentados como a porcentagem de escolha do melhor caminho após os 2000 segundos de simulação.

Tabela 5.7 – Comparação entre simulações com diferente taxa de aprendizado

Taxa de aprendizado (α)	**Porcentagem de escolha da ação ótima após entrada do tráfego**	**Tempo de estabilização inicial**	**Tempo de estabilização após entrada do tráfego**
0,7	92,5 %	120 s	360 s
0,3	87 %	200 s	1000 s

5.5.5 Teste 4: formas de definição da recompensa

Para este teste será usado novamente o cenário ilustrado na Figura 5.12 com os parâmetros e configurações listados nas tabelas 5.2 e 5.3, exceto quanto à forma de representar a recompensa para cada ação. Nos resultados mostrados anteriormente, foram utilizados valores discretos para a recompensa, um inteiro negativo para cada nível, como expresso na equação (5.8).

Primeiramente foi feita uma simulação com a recompensa quantizada positiva de acordo com a expressão (5.9) com duração de 10000 segundos. A Figura 5.27 mostra os caminhos escolhidos durante toda a duração da simulação. Percebe-se que, até nos momentos iniciais da simulação, quando o caso anterior não enfrenta muitos problemas para convergência para o caminho com menor número de saltos, o caso com recompensa positiva e valores $Q(s,a)$ iniciais nulos é bem mais lento e ineficiente. Com a entrada de mais tráfego aos 2000 segundos, é possível ver uma estabilização somente cerca de 2400 segundos depois. Esse resultado é explicado pelo fato de, no caso, não se ter seguido à diretriz indicada por Sutton e Barto em [20] para que se aproxime das condições iniciais ótimas escolhendo valores $Q(s,a)$ iniciais maiores do que os valores de convergência. Como as recompensas são positivas e os valores iniciais são nulos, claramente a *exploration* não será privilegiada em estados novos, deixando o aprendizado dependente de ε.

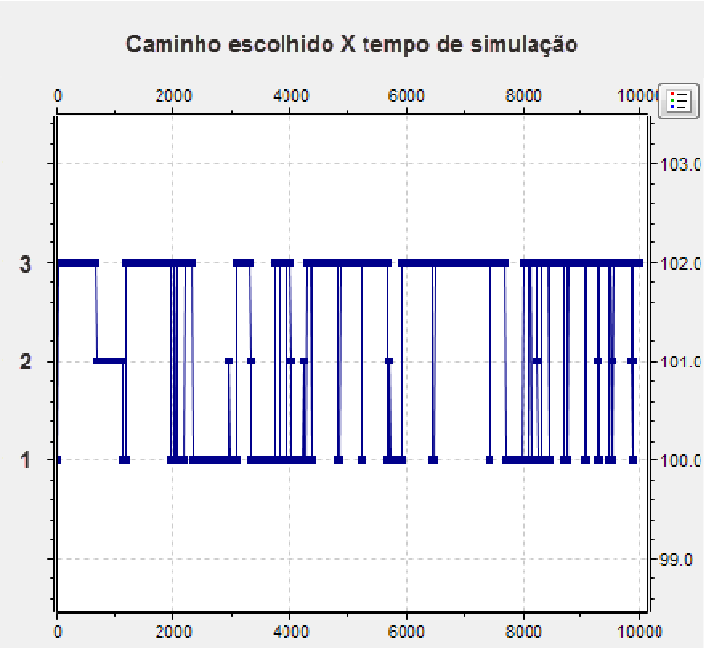

Figura 5.27 – Escolha do caminho para o caso de recompensa positiva.

Posteriormente foi testado o uso de recompensa negativa assumindo valores contínuos, de acordo com a expressão (5.7). A Figura 5.28 mostra o caminho selecionado ao longo do início da simulação com recompensa contínua. A estabilização inicial das escolhas dura, neste caso, aproximadamente 120 segundos, mesmo tempo do caso com recompensa quantizada. Já a Figura 5.29 mostra os caminhos escolhidos a partir da entrada dos novos fluxos de tráfego na rede, mostrando um tempo de adequação de aproximadamente 515 segundos, período maior que o observado no caso de recompensa quantizada negativa (360 segundos). O resultado indica que a quantização adequada da recompensa pode facilitar a convergência.

A Tabela 5.8 compila os referidos resultados e indica a porcentagem de escolha da ação ótima a partir dos 2000 segundos nos três casos, confirmando a superioridade da definição discreta e negativa para a recompensa.

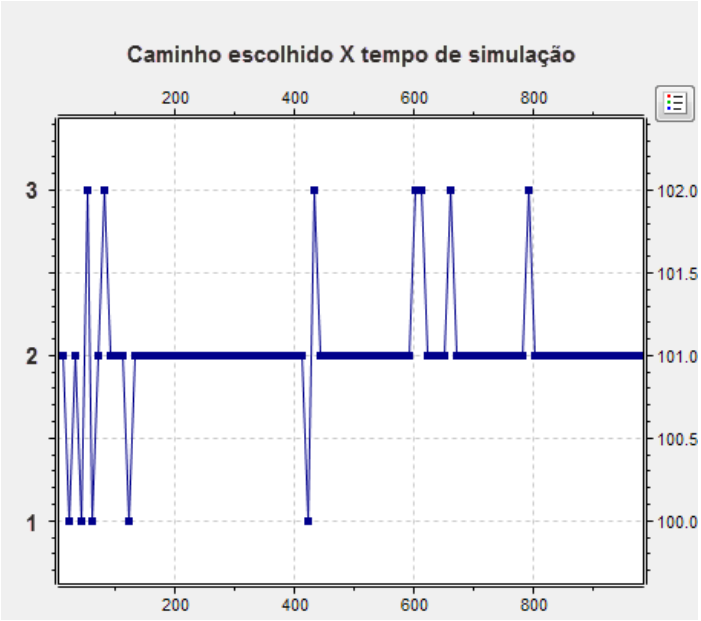

Figura 5.28 – Caminhos utilizados no começo da simulação com recompensa contínua.

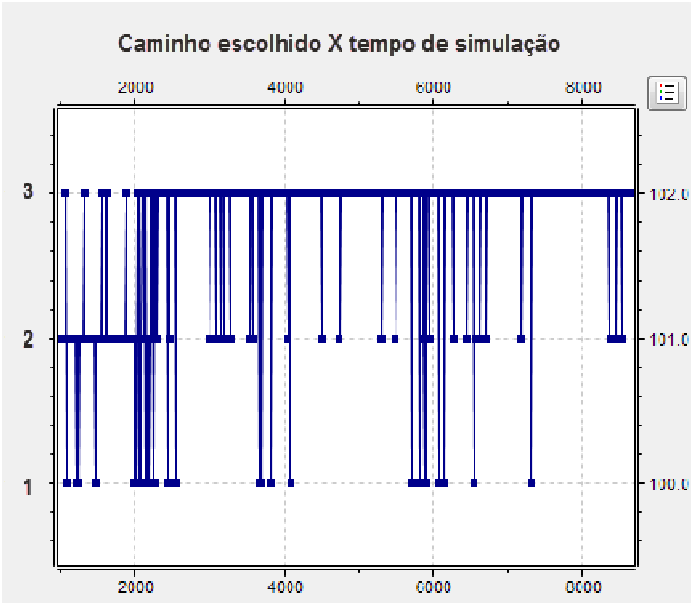

Figura 5.29 – Escolha dos caminhos após a entrada de mais tráfego com recompensa contínua.

Tabela 5.8 – Comparação entre simulações com formas diferentes de se definir a recompensa

Tipo de recompensa	**Porcentagem de escolha da ação ótima após entrada do tráfego**	**Tempo de estabilização inicial**	**Tempo de estabilização após entrada do tráfego**
Quantizada negativa	92,5 %	120 *s*	360 *s*
Contínua negativa	91,75 %	120 *s*	515 *s*
Quantizada positiva	66,25 %	–	2400 *s*

5.5.6 Teste 5: formas de quantização do atraso

Todos os testes realizados até esta subseção foram feitos com um número $L = 5$ de níveis, divididos uniformemente. Foi preparado, então, um cenário para se testar essa forma de divisão quando os caminhos diferentes possuem atrasos próximos. Para tal, foi utilizada a mesma rede mostrada na Figura 5.12, porém com um detalhe diferente: o atraso de propagação médio dos links que compõem o caminho 3 foi aumentado para $230\,\mu s$. Além disso, somente será adicionado tráfego por meio do gerador 2. Desta forma, os caminhos 1 e 3, que possuem igual quantidade de saltos e *links* com iguais taxas de transmissão, terão atraso semelhante. O caminho 3 terá um atraso um pouco maior em razão da propagação, mas ainda assim maior, o que faz que o resultado desejado seja a preferência pelo caminho 1. Os fluxos configurados para o presente teste estão descritos na Tabela 5.9.

Tabela 5.9 – Fluxos de pacotes configurados para o quinto teste

Fluxo	**Origem**	**Destino**	**Descrição**	**Distribuição do tempo entre pacotes**	**Tempo de transmissão**	
					Início	**Fim**
Tráfego de referência	*Hosts* 1 e 2	*Host* 3	Dois fluxos de ~200 kbps	Normal (0.01 *s*, 0.000001 *s*)	1,5 *s*	–
Tráfego injetado 2	Gerador 2	*Host* 4	Fluxo de ~460 kbps	Normal (0.00429 *s*, 0.000001 *s*)	2000 *s*	–

Os níveis de atraso encontrados neste caso podem ser observados na Figura 5.30, na qual fica evidente que o atraso efetivo dos caminhos 1 e 3 permanecem no mesmo nível de atraso. De fato, ao se iniciar o tráfego injetado pelo gerador 2 aos 2000 segundos de simulação, pode-se ver na Figura 5.31 que a solução não encontrou o melhor caminho, com escolhas oscilando entres os caminhos 1 e 3.

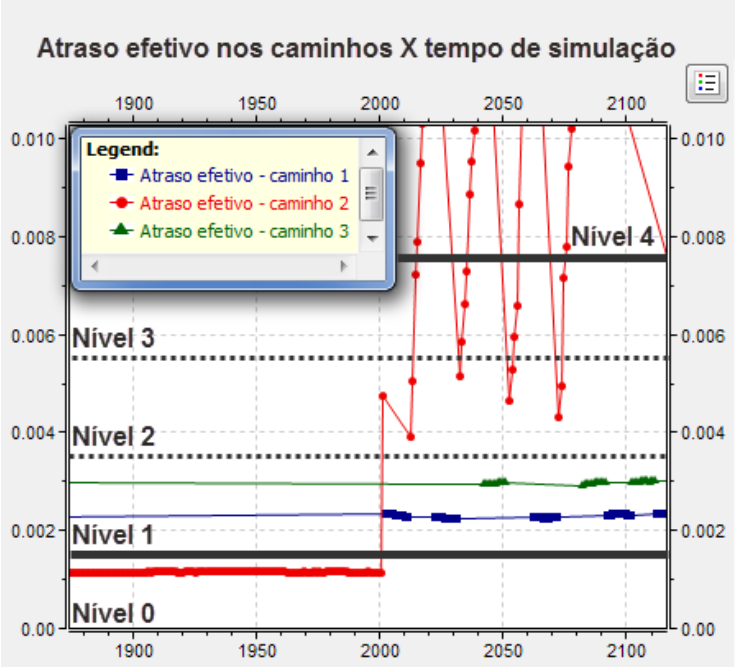

Figura 5.30 – Níveis de atraso efetivo no quinto teste com $L = 5$.

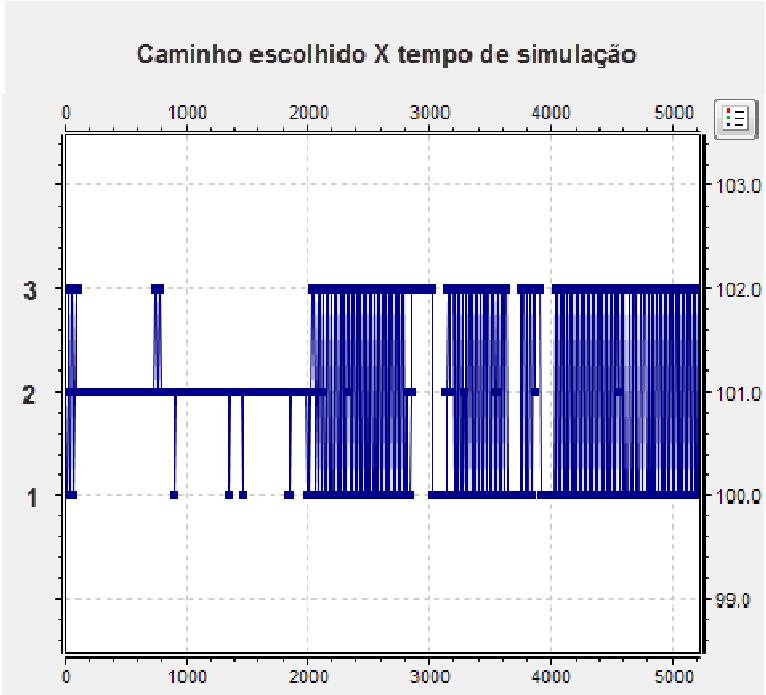

Figura 5.31 – Caminho escolhido com atrasos próximos e $L = 5$.

Espera-se que com um maior nível de divisões, o algoritmo se torne mais sensível a diferenças pequenas no atraso dos caminhos, como no caso. Portanto foi feita nova simulação utilizando-se uma divisão uniforme com $L = 10$ níveis, o que aumenta o número de estados para $3 \cdot 10^3 = 3000$ (vide Figura 5.6). Na Figura 5.32, pode-se ver o caminho escolhido pelo sistema durante toda a simulação nesta nova situação. Verifica-se que, com a entrada do tráfego injetado, o caminho 1 é o mais utilizado após um período inicial de aproximadamente 1030 segundos. Podem ser testados também outros tipos de quantização para verificar sua adequação ao cenário. A seguir serão apresentados resultados dos testes com a quantização de atraso de forma logarítmica, como descrito pela equação (5.5), com $f = 12$, e quantização exponencial, conforme a equação (5.6), com $f = 2{,}5$.

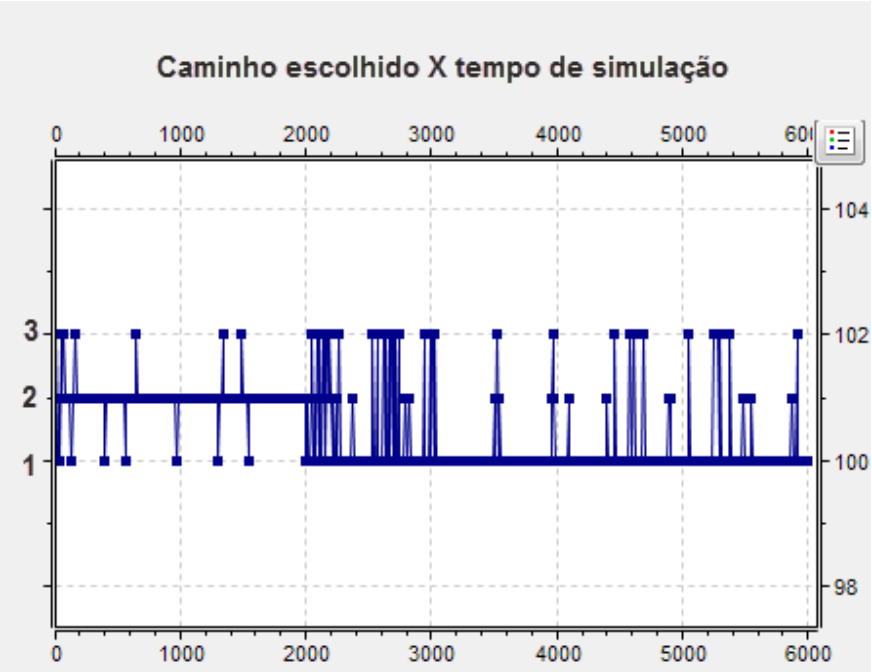

Figura 5.32 – Escolha do caminho para quantização uniforme com $L = 10$.

A escolha dos caminhos foi otimizada com a quantização logarítmica, como pode ser visto na Figura 5.33. Além de um menor tempo de estabilização inicial (60 segundos), a convergência após a entrada do tráfego adicional foi consideravelmente menor: 220 segundos. Já com a divisão exponencial, Figura 5.34, a convergência após a entrada de tráfego piorou, em relação à quantização uniforme, apesar de alguma melhora para as escolhas no início da simulação. Uma comparação geral entre as três formas de quantização é apresentada na Tabela 5.10, com posterior análise.

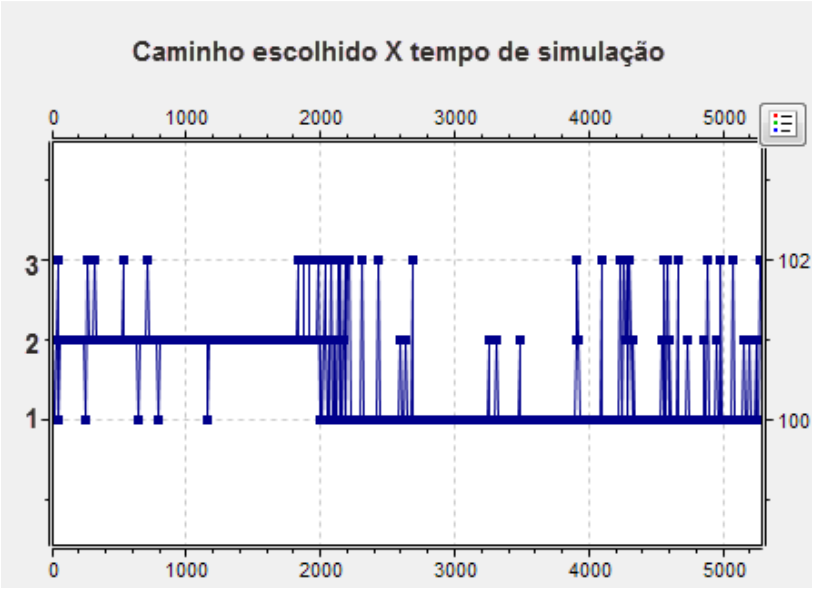

Figura 5.33 – Escolha do caminho para quantização logarítmica com $L = 10$.

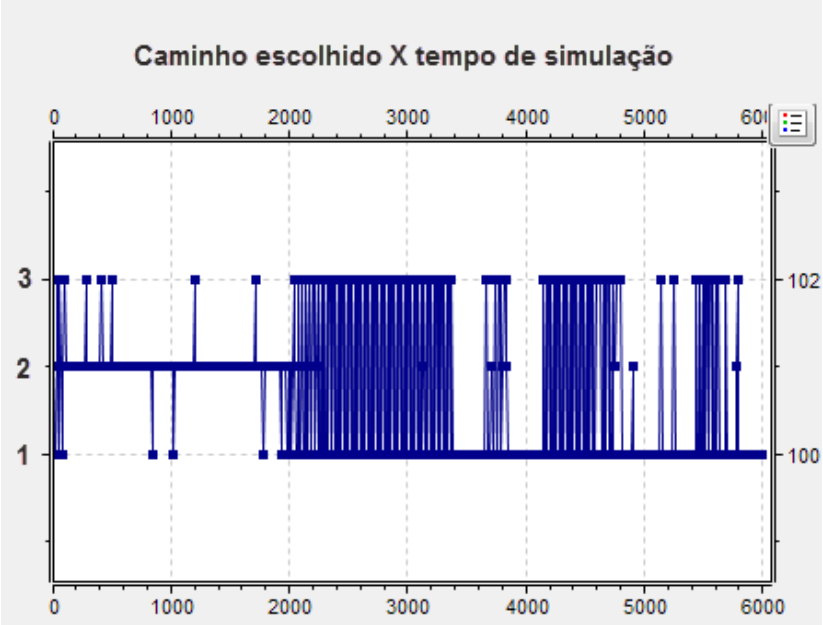

Figura 5.34 – Escolha do caminho para quantização exponencial com $L = 10$.

Tabela 5.10 – Comparação entre simulações com formas diferentes de se quantizar o atraso com $\boldsymbol{L = 10}$

Quantização do atraso	**Porcentagem de escolha da ação ótima após entrada do tráfego**	**Tempo de estabilização inicial**	**Tempo de estabilização após entrada do tráfego**
Uniforme	88,6 %	80 *s*	1030 *s*
Logarítmica	89,4 %	60 *s*	220 *s*
Exponencial	66 %	110 *s*	1380 *s*

No cenário testado com as três maneiras de se dividir o atraso, os níveis mais baixos têm importância fundamental tanto no início da simulação quanto no momento em que o tráfego injetado é ativado. No início, porque essa fase da simulação é marcada por atrasos menores. Na entrada do tráfego adicional, porque os caminhos 1 e 3 passam a ser as alternativas de menor atraso e possuem níveis próximos de atraso efetivo. Como a quantização logarítmica possui mais divisões para um nível relativamente menor de atraso (Figura 5.11), para esse cenário ela se mostra mais adequada para descrever o estado em que se encontra o ambiente, fazendo que a estabilização se dê mais rapidamente nos dois momentos citados, como mostra a Tabela 5.10. Já com a quantização exponencial, as divisões do atraso em menor nível são mais espaçadas, o que acarreta numa representação menos fiel do estado do ambiente quando há caminhos com níveis de atraso baixos porém próximos.

Conclui-se que a melhor forma de quantizar o atraso depende do perfil de atraso de cada caso, sendo a quantização uniforme a escolha mais segura quando este não pode ser conhecido ou previsto.

5.5.7 Teste 6: comparação com o *Q-routing*

Foi realizado um teste comparativo da solução desenvolvida neste trabalho, RL-RSVP-TE, com o célebre protocolo *Q-routing*, com seu parâmetro $\alpha_{Q-routing} = 0{,}7$. O

cenário da Figura 5.12 foi utilizado para as duas simulações, porém com o novo perfil de fluxos de tráfego descrito na Tabela 5.11. O tráfego de referência fica presente no cenário até o instante final da simulação t_{fim}. Os tráfegos injetados 1 e 2 são ativados a partir do instante t_{in} e permanecem até o instante $t_{out} < t_{fim}$. O propósito é analisar a capacidade de readaptação das duas estratégias. Os parâmetros utilizados para a solução RL-RSVP-TE são os mesmos da Tabela 5.2

Tabela 5.11 – Fluxos de pacotes configurados para o teste comparativo

Fluxo	**Origem**	**Destino**	**Descrição**	**Distribuição do tempo entre pacotes**	**Tempo de transmissão**	
					Início	**Fim**
Tráfego de referência	*Hosts* 1 e 2	*Host* 3	Dois fluxos de ~200 kbps	Normal (0.01 *s*, 0.000001 *s*)	1,5 *s*	t_{fim} *s*
Tráfego injetado 1	Gerador 1	*Host* 4	Fluxo de ~400 kbps	Normal (0.0049 *s*, 0.000001 *s*)	t_{in} *s*	t_{out} *s*
Tráfego injetado 2	Gerador 2	*Host* 4	Fluxo de ~460 kbps	Normal (0.00429 *s*, 0.000001 *s*)	t_{in} *s*	t_{out} *s*

Inicialmente, devem-se ressaltar algumas particularidades de cada solução. O *Q-routing* não se baseia em nenhuma tecnologia de redes utilizada amplamente. Seu funcionamento depende do envio de um pacote de confirmação para cada transferência de um pacote de dados por um único enlace. Assim, para cada pacote que um roteador enviar ocorrerá uma iteração do algoritmo neste equipamento. Por outro lado, com a ferramenta RL-RSVP-TE, realizam-se iterações de seu algoritmo numa taxa mais lenta para que o estado do ambiente seja verificado. Além disso, as iterações com política gulosa ocorrem numa taxa ainda menor que as iterações engessadas.

Na simulação com o *Q-routing*, a rede se adequou rapidamente ao melhor caminho inicial, como pode ser observado na Figura 5.35. Nela, mostra-se a interface escolhida pelo roteador 1 (no caso do RL-RSVP-TE, o LER *sender*) para o envio do tráfego de referência. O tempo necessário para que a escolha do caminho utilizado fosse estabilizada foi de 0,12 segundo, sendo o roteador 1 o responsável pela última mudança. Como o tempo médio entre pacotes é de 0,01 segundo, tem-se um número aproximado de 12 iterações do *Q-routing* para convergência de escolhas no início. A interface escolhida após a estabilização foi a de número 4, ligada ao caminho 2, que de fato oferece o melhor atraso por ter um salto a menos que os demais.

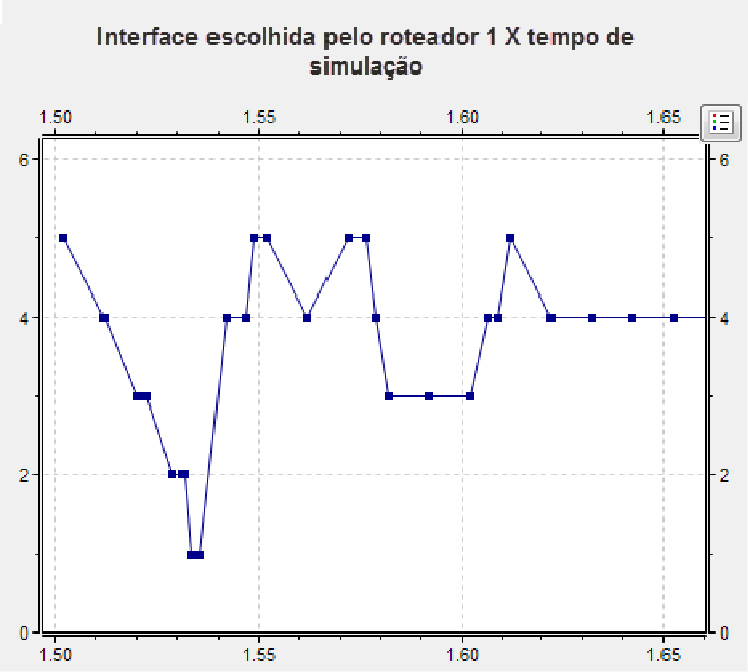

Figura 5.35 – A escolha inicial de interfaces do roteador 1, com o *Q-routing*.

A partir do instante $t_{in} = 1000{,}5$ segundos, os demais fluxos foram ativados e o *Q-routing* teve de adaptar em cada roteador a melhor interface para transmissão. A evolução da interface escolhida pelo roteador 1 pode ser vista na Figura 5.36. Observa-se que a interface que passou a ser a selecionada para encaminhar o tráfego de referência no roteador 1 foi a de número 5, que é ligada ao caminho 3. Outro resultado esperado em razão da falta de fluxos concorrentes nesse caminho. O tempo levado para

a adaptação ao novo perfil de tráfego foi de 0,64 segundo, o que corresponde aproximadamente a 64 iterações do *Q-routing*. Contudo, é possível observar na Figura 5.37 que o roteador 1 não foi o último a se adaptar na nova conjuntura. O tempo total para adaptação de todos os roteadores presentes foi de 1,52 segundo, correspondendo a aproximadamente 152 iterações do algoritmo *Q-routing* de todos os roteadores da rede.

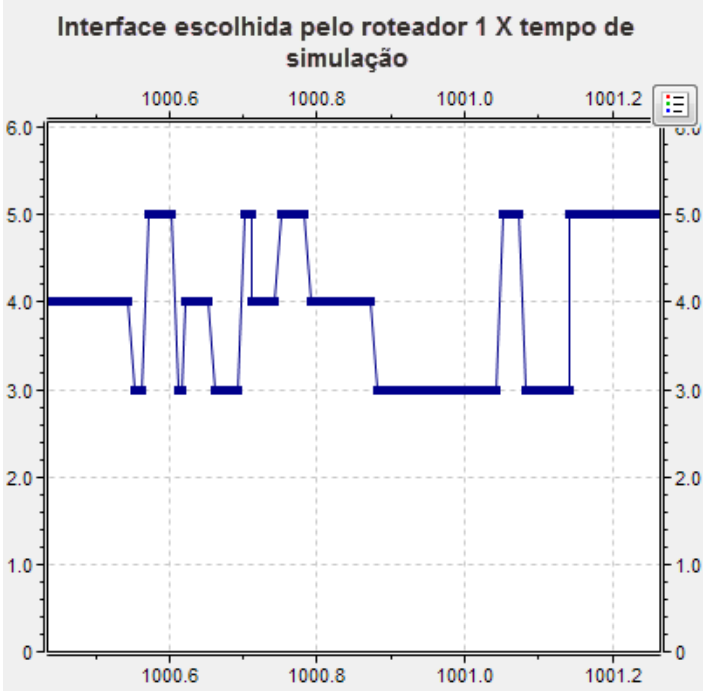

Figura 5.36 – A escolha de interfaces do roteador 1, com o *Q-routing*, após a entrada de novos fluxos.

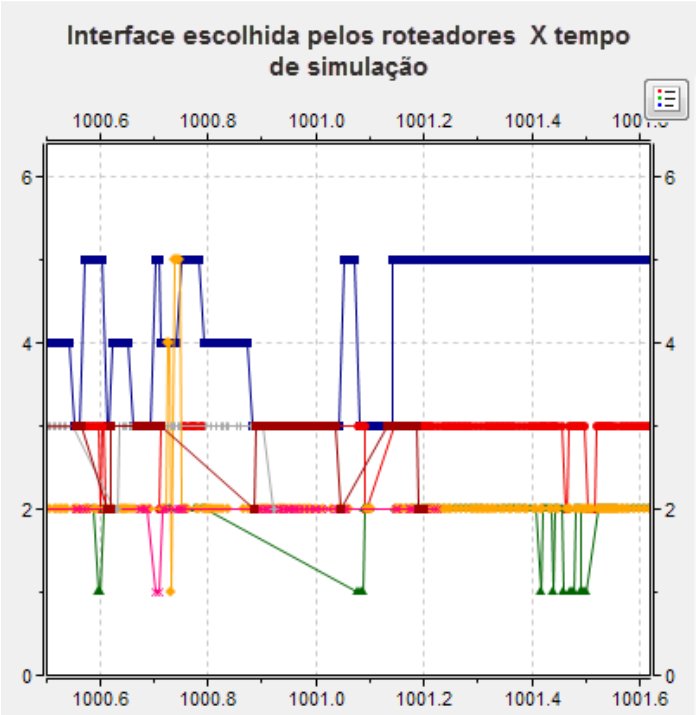

Figura 5.37 – Visão geral da escolha de interfaces de todos os roteadores, com o *Q-routing*, após a entrada de novos fluxos.

A partir de $t_{out} = 2000$ segundos, os tráfegos injetados 1 e 2 foram interrompidos para verificar a resposta do algoritmo. O perfil de interfaces escolhidas por todos os roteadores durante todo o tempo de simulação pode ser visto na Figura 5.38. Como esperado, o *Q-routing* não foi capaz de se readaptar às novas condições de menos tráfego, pois não faz distinção dos estados do ambiente e não realiza *exploration*. Os outros caminhos somente serão testados novamente se o caminho 3 vier a sofrer com alto tráfego, de modo a resultar em altas estimativas de atraso para tentar superar a dos demais caminhos.

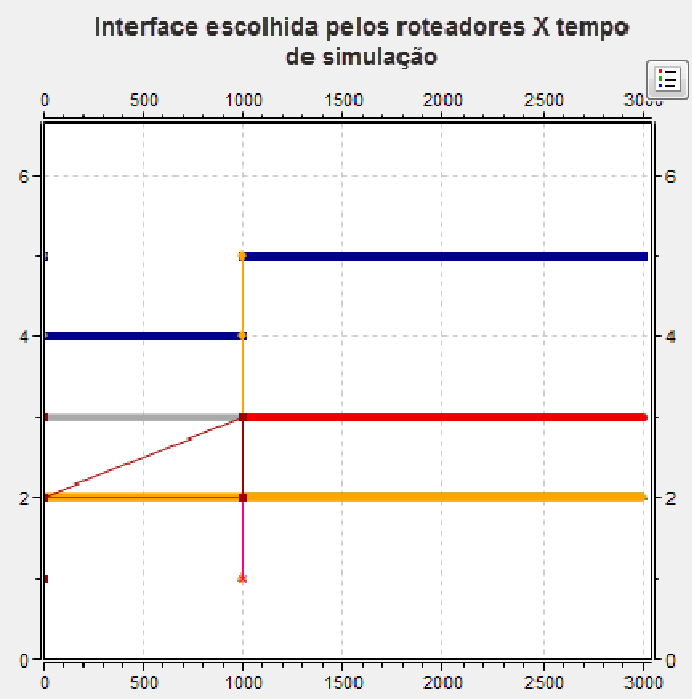

Figura 5.38 – Visão geral da escolha de interfaces de todos os roteadores, com o *Q-routing*, durante toda a simulação.

Para a comparação com o *Q-routing*, foi feita uma simulação utilizando a ferramenta RL-RSVP-TE no mesmo cenário, mesmos fluxos definidos na Tabela 5.10 e com os parâmetros definidos na Tabela 5.2. Diferentemente do clássico algoritmo adaptativo, o RL-RSVP-TE é bem sucedido ao se readaptar com a retirada do tráfego adicional, como pode ser visto na Figura 5.39. Para tal são gastos 450 segundos, que representam 450 iterações realizadas pela ferramenta – já que T2 = 1 segundo. Destas, 45 iterações são *ε-greedy* do *Q-learning* e 405 são iterações engessadas do SARSA. A Tabela 5.12 expõe um quadro comparativo entre a quantidade de iterações necessárias para as adaptações de cada protocolo.

Portanto, apesar de levar mais tempo para convergência por propositadamente tentar evitar instabilidades na rede, o RL-RSVP-TE pode ser aplicado em redes reais, com todas as vantagens e controle de falha de protocolos já utilizados e ainda oferece um contorno a uma grave falha do *Q-routing*. Além disso, à medida que permanece em funcionamento na rede, o RL-RSVP-TE refina seu funcionamento, pois ao contrário do

outro protocolo considerado, possui um descritor de estados que pode dar ao sistema um aprendizado sólido do melhor caminho a se tomar em cada situação.

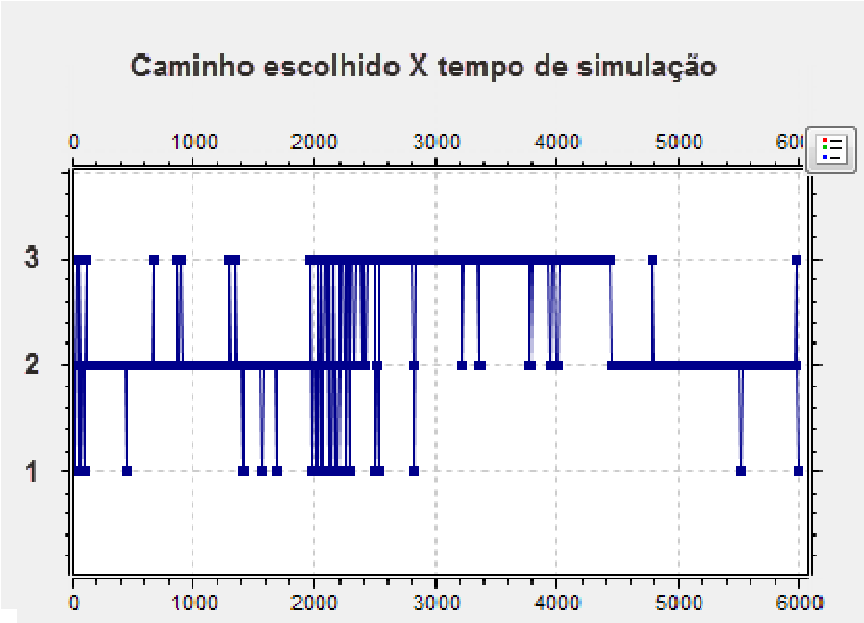

Figura 5.39 – Evolução do caminho escolhido com o uso do RL-RSVP-TE.

Tabela 5.12 – Comparação entre *Q-routing* e RL-RSVP-TE

Protocolo		**Iterações para estabilização inicial**	**Iterações para estabilização após entrada do tráfego**	**Iterações para estabilização após saída do tráfego**
Q-routing		12	152	∞
RL-RSVP-TE	Atualizações em T3	12	36	45
	Atualizações em T2	108	324	405

5.6 CONCLUSÃO DO CAPÍTULO

Neste capítulo, foram apresentados o *software* OMNeT++ e a ferramenta desenvolvida neste trabalho: sua estratégia, seus parâmetros e alguns testes. Foi mostrada a aplicabilidade da solução, a influência de parâmetros importantes e como utilizá-los para melhorar o desempenho.

Foram testados os algoritmos *Q-learning* e SARSA para as atualizações dos valores $Q(s,a)$ nos dois instantes importantes T2 e T3. Como em T2 a política utilizada não é a gulosa por várias iterações, verificou-se que o SARSA – que depende de política específica – é mais adequado para esse instante. Já em T3, o *Q-learning* ofereceu o melhor desempenho, pois se aplica política *ε-greedy*. Foi também mostrada por meio de uma simulação a importância de realizarem as atualização com ação fixa em T2, para mesmo assim atualizar valores $Q(s,a)$ e ajudar na convergência.

Foram realizadas comparações de algumas formas de se definir a recompensa. Quantizar a recompensa de acordo com os níveis de atraso pode trazer ganhos, e a melhor maneira de se iniciarem os valores $Q(s,a)$ é com valores maiores do que os valores ótimos, para se estimular *exploration* naturalmente. Foi testado também um caso em que os níveis de dois caminhos estivessem mais próximos, o que necessitou de uma divisão com maior número de níveis. Já para os três tipos diferentes de se quantizar o atraso – uniforme, logarítmico e exponencial – os testes indicaram que o melhor tipo a se usar deve ser escolhido de acordo com as características de atraso da rede, ou seja, se a maior e mais significativa variação do atraso se dará em valores próximos de Δt_{min} ou de Δt_{max}.

Também foi feita uma comparação entre a ferramenta desenvolvida e o protocolo *Q-routing*, no qual foi constatada a capacidade do RL-RSVP-TE de contornar uma grave falha de readaptabilidade que aquele apresenta.

6 CONCLUSÃO

Este trabalho foi motivado pela ideia de se acrescentar inteligência a protocolos existentes, de modo que a rede possa resolver problemas de engenharia de tráfego de forma autônoma e dinâmica. Ocupou-se, então, de um problema específico de engenharia de tráfego: a escolha do melhor caminho, entre um grupo de *L* disponíveis, para entrega de um fluxo de pacotes a um grande cliente de um provedor de serviços de dados. O parâmetro de desempenho escolhido para a avaliação de um caminho foi o atraso fim a fim dos pacotes que o percorrem.

Como resultado do trabalho, foi apresentada a ferramenta RL-RSVP-TE, uma adaptação feita ao protocolo RSVP-TE para que o roteador LER *sender* da sessão escolha o melhor de seus caminhos para a transmissão utilizando a técnica do aprendizado por reforço. A ferramenta foi implementada no ambiente OMNeT++ de simulação computacional e foram realizados testes para verificar a influência dos parâmetros do sistema em seu desempenho, além de comparação com o clássico algoritmo *Q-routing*. Foram definidos três períodos de referência, em ordem crescente:

- T1 para o envio de pacotes de teste para levantar o nível de atraso em cada caminho;

- T2 para a atualização engessada do algoritmo de aprendizado por reforço. Foi mostrado que este é um aspecto importante por melhorar o desempenho, facilitando a convergência da ferramenta. Mostrou-se também que a utilização do algoritmo SARSA nesses instantes é fundamental para o bom funcionamento da solução proposta;

- T3 para a atualização gulosa do aprendizado, com a utilização do *Q-learning* com política gulosa para melhor desempenho.

Foi possível concluir, adicionalmente, que a melhor maneira de se definir recompensas é por meio de valores adequadamente quantizados e negativos, com valores Q iniciais nulos. Isso significa que existe um compromisso entre a divisão do atraso em níveis e a resposta adequada do sistema às mudanças do perfil de atrasos da rede. Quanto mais níveis, maior o número de estados e consequentemente uma convergência mais lenta.

A ferramenta desenvolvida apresentou boa readaptabilidade aos níveis de atraso, ao contrário do *Q-routing*, que possui um ponto fraco nesse aspecto. Além disso, o RL-RSVP-TE pode ser aplicado diretamente em redes reais para funcionamento *online*, com poucas alterações em protocolos comerciais.

Pode-se utilizar uma estratégia para diminuir o tempo de convergência da solução em alguns casos: uma rede real pode ser reproduzida no simulador OMNeT++ e uma simulação é feita para submetê-la aos níveis de tráfego usuais. Após certo período de aprendizado, uma parte da tabela $Q(s, a)$ estará preenchida. Essa tabela obtida por meio de simulação pode ser, então, carregada no roteador LER de entrada da rede real para que o agente de aprendizado saiba como se comportar em situações parecidas às já vivenciadas durante a simulação. Essa estratégia visa a embutir o aprendizado adquirido via simulação na aplicação *real time* da solução. Essa rede será submetida a outras situações e, com o tempo e uma devida divisão dos níveis de atraso, ter-se-á avançado no objetivo de dar alguma inteligência à rede.

Uma das dificuldades no desenvolvimento do trabalho foi a procura de uma ferramenta computacional para auxiliar na realização dos objetivos. O OMNeT++ possibilitou várias tarefas, porém com um empecilho significativo de falta de documentação, principalmente de seu pacote INET. Para as simulações e testes realizados – tanto com o *Q-routing* quanto com a solução RL-RSVP-TE – foi necessário modificar 14 arquivos de implementação de protocolos e modelos (alguns com milhares de linhas de código), bem como criar outros 5 do mesmo tipo. Adicionalmente, foram escritos 18 arquivos de topologia e configuração de cenários de simulação.

Como trabalhos futuros, podem-se considerar outros parâmetros de desempenho para ser utilizadas como recompensa ou definição de estados do ambiente. Inclusive, pode ser investigada uma adaptação da ferramenta ao aprendizado por reforço multi-objetivo. Além disso, podem ser alguns mais passos em direção a uma solução ainda mais autônoma e com inteligência em outros aspectos. Por exemplo, pode ser feita uma estratégia para a descoberta do conjunto de melhores caminhos a serem considerados pelo RL-RSVP-TE. Outra área que pode ser investigada para possíveis avanços é a de representação adaptativa [42]: técnicas para que o agente de aprendizado por reforço também adapte automaticamente sua representação do ambiente com o objetivo de melhorar o desempenho.

REFERÊNCIAS

[1] Barakovic, S., Barakovic, J., "Traffic performances improvement using DiffServ and MPLS networks." In: XXII International Symposium on Information, Communication and Automation Technologies, 2009. ICAT 2009. IEEE, 2009.

[2] Alpaydin, E., “Introduction to Machine Learning”. London, Cambridge: The MIT Press, 2004

[3] Osborne, E. D., e Simha, A. “Traffic engineering with MPLS”. 2ª ed. Indianapolis: Cisco Press, 2002.

[4] Varga, A., "OMNeT++". Capítulo do livro "Modeling and Tools for Network Simulation", Wehrle, K.; Günes, M.; Gross, J. (editores). Springer Verlag, 2010.

[5] Tanenbaum, A. S. “Redes de Computadores”. 4ª ed. Rio de Janeiro: Campus, 2005.

[6] Kurose, J. F., Ross, Keith W. “Redes de Computadores e a Internet – uma abordagem top-down”. 3ª ed. São Paulo: Pearson, 2006.

[7] Comer, D. E. “Interligação em Redes com TCP/IP Volume 1: Princípios, protocolos e arquitetura”. 3ª ed. Rio de Janeiro: Campus, 1998.

[8] Rudin, H. “On Routing and ‘Delta Routing’: A Taxonomy and Performance Comparison of Techniques for Packet-Switched Networks”. In: IEEE TRANSACTIONS ON COMMUNICATIONS, VOL. COM-24, NO. 1, 1976.

[9] Hawkinson, J., Bates, T. “Guidelines for creation, selection, and registration of an Autonomous System (AS)”. IETF, RFC 1930, 1996. Disponível *online* em: http://www.ietf.org/rfc/rfc1930.txt

[10] Moy, J. (editor) “OSPF Version 2”. IETF, RFC 2328, 1998. Disponível *online* em: http://www.ietf.org/rfc/rfc2328.txt

[11] Cisco Document ID: 7039. “OSPF Design Guide”. Disponível *online* em: http://www.cisco.com/image/gif/paws/7039/1.pdf

[12] Bertsekas, D., Gallager, R. “Data Networks”. 2ª ed. New Jersey: Prentice Hall, 1992.

[13] Yiltas, D., Perros, H. “Quality of service-based multi-domain routing under multiple quality of service metrics”. In: IEEE COMMUNICATIONS MAGAZINE, VOL. 5, NO. 3, 2011.

[14] Oliveira, A. H. C. “Gerenciamento de Túneis em Ambiente Integrado Mobile IP a MPLS”. Dissertação de mestrado, UnB, 2006.

[15] Rosen, E., Viswanathan, A., e Callon, R. “Multiprotocol label switching architecture”. IETF, RFC 3031, 2001. Disponível *online* em: http://www.ietf.org/rfc/rfc3031.txt

[16] Nardelli, A. P. S., Nóbrega, L. B. “Tecnologia IP MPLS/VPN: Estudo de um caso real”. Monografia de projeto final de graduação, UnB, 2010.

[17] Deus, M. A. D. “Estratégias de gerenciamento de banda IP/MPLS para o transporte eficiente de serviços integrados”. Dissertação de mestrado, UnB, 2009.

[18] Zhang, L., Berson, S., Herzog, S., e Jamin, S. “Resource reservation protocol (RSVP) - Version 1 functional specification”. IETF, RFC 2205, 1997. Disponível *online* em: http://www.ietf.org/rfc/rfc2205.txt

[19] Awduche, D., et al. “RSVP-TE: extensions to RSVP for LSP tunnels”. IETF, RFC 3209, 2001. Disponível *online* em: http://www.ietf.org/rfc/rfc3209.txt

[20] Sutton, R. S., e Barto, A. G. “Reinforcement learning: An introduction.” Vol. 1. No. 1. Cambridge: MIT Press, 1998.

[21] Kaelbling, L. P., Littman, M. L., e Moore, A. W. “Reinforcement learning: A survey.” In: Journal of Artificial Intelligence Research 4, 1996.

[22] Peshkin, L. “Reinforcement Learning by Policy Search”. PhD thesis, MIT, 2002.

[23] Buşoniu, L. et al. “A Comprehensive Survey of Multiagent Reinforcement Learning”. In: IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS, 2008.

[24] Watkins, C. J., e Dayan, P. “Technical note: Q-learning”. In: Machine learning, 8(3-4), 279-292, 1992.

[25] Wang Qiang, Zhan Zhongli. “Reinforcement learning model, algorithms and its application”. In: International Conference on Mechatronic Science, Electric Engineering and Computer (MEC), 2011.

[26] Leite, J. P., Carvalho, P. H. P. de, e Vieira, R. D. “Modulação e Codificação Adaptativa em Sistemas OFDM Utilizando Aprendizado por Reforço”. In: XXX SIMPÓSIO BRASILEIRO DE TELECOMUNICAÇÕES. Brasília, 2012.

[27] Littman, M., Boyan, J. “A Distributed Reinforcement Learning Scheme for Network Routing”. In: Proceedings of the 1993 International Workshop on Applications of Neural Networks to Telecommunications.

[28] Littman, M., e Boyan, J. “Packet Routing in Dynamically Changing Networks: A Reinforcement Learning Approach”. In: Advances in Neural Information Processing Systems 6, 1994.

[29] Ouzecki, D., e Jevtic, D. “Reinforcement learning as adaptive network routing of mobile agents”. In: MIPRO, 2010 Proceedings of the 33rd International Convention, 2010.

[30] Choi, S., Yeung, D. “Predictive Q-Routing: A Memory-based Reinforcement Learning Approach to Adaptive Traffic Control”. In: Advances in Neural Information Processing Systems 8, 1996.

[31] Kumar, S., Miikkulainen, R. “Dual Reinforcement Q-Routing: An On-Line Adaptive Routing Algorithm”. In: Proceedings of the Artificial Neural Networks in Engineering Conference, 1997.

[32] Kumar, S., Miikkulainen, R. “Confidence-Based Q-Routing: An On-Line Adaptive Network Routing Algorithm”. In: Proceedings of Artificial Neural Networks in Engineering, 1998.

[33] Peshkin, L. e Savova, V. “Reinforcement Learning for Adaptive Routing”. In: Proceedings of the 2002 International Joint Conference on Neural Networks, 2002.

[34] Fortz, B., Rexford, J., Thorup, M., “Traffic engineering with traditional IP routing protocols”. In: IEEE Communications Magazine, 40(10):118-124, 2002.

[35] Zhou H., Pan J., Shen P. “Cost Adaptive OSPF”. In: Proceedings of the Fifth International Conference on Computational Intelligence and Multimedia Applications, 2003 (ICCIMA’03).

[36] Carvalho, P. H. P. de, Barreto, P. S., Ceppo, M. P., e Silva, R. D. de O. C. “A Methodology for Network Planning and Adaptative Routing in Multimedia Networks”. In: IEEE LATIN AMERICA TRANSACTIONS, VOL. 7, NO. 6, 2009.

[37] Peng Liu, Rentao Sun, Zhidu Yan. “Dynamic OSPF Protocol”. In: 4th IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT), 2011.

[38] De Neve, H., Van Mieghem, P., “A multiple quality of service routing algorithm for PNNI”. In: Proc. IEEE ATM Workshop, Fairfax, VA, USA, 1998, pp. 324–328.

[39] Korkmaz, T., Krunz, M.: “Multi-constraint optimal path selection”. In: Proc. 20th Annual Joint Conf. on IEEE Computer and Communications Societies (INFOCOM), Anchorage, AK, USA, 2001, pp. 834–843.

[40] Diniz, P. S., “Adaptive filtering: algorithms and practical implementation”. New York: Springer, 2009.

[41] Rummery, G. A., Niranjan, M. “Online Q-learning using connectionist systems”. Cambridge: University of Cambridge, 1994.

[42] Whiteson, S. A., “Adaptive Representation for Reinforcement Learning”. PhD thesis. University of Texas at Austin, 2010.

APÊNDICES

APÊNDICE A – PRINCIPAIS ARQUIVOS DO OMNET++ MODIFICADOS

Lista dos principais arquivos modificados no pacote INET 2.0 do *software* OMNeT++, em sua versão 4.2.2, para implementação da solução RL-RSVP-TE.

Arquivos modificados	Objetivo
RSVP.cc	Implementação do processo RL propriamente dito.
RSVP.h	
IPv4.cc	Tratamento dos pacotes RLTimer para envio dos pacotes de teste.
IPv4.h	
MPLS.cc	
MPLS.h	
IPv4Datagram.msg	Acrescentado *flag* para indicar se o datagrama se trata de pacote de teste.
LDP.cc	Viabilização da mudança de caminho LSP escolhido pela solução na tabela LIB.
LDP.h	
IClassifier.h	
SimpleClassifier.h	
IntServ.msg	Implementação da mensagem de controle RSVP especial para levar as informações de atraso ao LER *sender*.
SignallingMsg.msg	

APÊNDICE B – ARQUIVOS CRIADOS NO OMNET++

Lista dos arquivos que precisaram ser criados utilizando o pacote INET 2.0 do *software* OMNeT++, para implementação da solução RL-RSVP-TE.

Arquivos criados	Objetivo
RLTimer.msg	No LER *sender*, implementação do pacote RLTimer para que o módulo “ip” seja informado ao final dos períodos T1, para preparação dos datagramas dos pacotes de teste.
RLCapsule.msg	No LER *sender*, envio dos datagramas de teste do módulo “ip” para o módulo “mpls”, para envio.
RLInfo.msg	No LER *receiver*, envio da medição de atraso de cada pacote de teste do módulo “mpls” para o módulo “rsvp”.