Maria João Mesquita Rodrigues da Cunha Nicolau Pinto

Encaminhamento Diferenciado para Comunicações em Grupo com Requisitos de Qualidade de Serviço

Tese
submetida à Universidade do Minho
para a obtenção do grau de Doutor em Informática

Escola de Engenharia
Departamento de Informática
UNIVERSIDADE DO MINHO

Braga
2005

Aos meus pais

Agradecimentos

Ao Professor Alexandre Santos pela atenção que dedicou à orientação deste trabalho, pelo apoio dado aos vários níveis, pelas valiosas sugestões, por ter sido desde o inicio do trabalho muito mais do que um orientador.

Ao meu colega António Costa, meu companheiro de "batalha", com quem partilhei ao longo de todo este tempo os momentos de alegria e principalmente os de dúvida e desânimo.

Ao Professor Joaquim Macedo, por todo o interesse demonstrado e pelas palavras de incentivo com que sempre animou as nossas conversas.

Aos restantes colegas do Grupo de Comunicações por Computador, nomeadamente ao Paulo, à Solange e ao Pedro Nuno, por me fazerem sentir que não estava sozinha "neste barco".

Ao Departamento de Informática pelas condições criadas e pelo bom ambiente de trabalho proporcionado.

Aos colegas do grupo de Engenharia de Telecomputação (GET), Helena, Ricardo, Henrique, Adriano e Rui João, e à Maribel, pela solidariedade demonstrada em tantas alturas.

Ao Departamento de Sistemas de Informação por me ter dado oportunidade de fazer a "tese" que eu mais queria.

Ao Professor João Álvaro, pela sua constante preocupação e interesse.

Ao Professor Altamiro Machado, que apesar de já não estar entre nós, está sempre presente.

Ao Fernando, à Catarina e ao Zé por nunca me terem deixado esquecer que havia outras coisas mais importantes, para além do "doutramento".

Resumo

A maioria das propostas para encaminhamento *multicast* com QoS propõem que as rotas sejam calculadas a pedido, sendo necessário que os pedidos explicitem os recursos de que necessitam. Como resultado, neste contexto, o objectivo do encaminhamento com QoS é satisfazer pedidos individuais de requisitos, deixando para os mecanismos de reserva de recursos a responsabilidade de manter os recursos necessários à satisfação desses requisitos depois de ter sido descoberto, pelo processo de encaminhamento com QoS, um caminho exequível. Este tipo de estratégia adequa-se ao modelo IntServ, mas não parece adequado na presença de redes DiffServ. De acordo com o modelo DiffServ, os diferentes fluxos de tráfego são agregados em classes de serviço e cada fluxo recebe um tratamento específico de acordo com a classe de serviço a que pertence. Deixam de ser dadas garantias por fluxo passando existir apenas diferenciação por classe de serviço. Nestas redes DiffServ devem então ser calculadas rotas por classe de serviço e não por fluxo como até aqui.

Esta tese propõe um protocolo de encaminhamento *multicast*, o MCMRP (Multi Class Multicast Routing Protocol), um novo protocolo que permite a construção de árvores de distribuição de tráfego por classes de serviço. As heurísticas propostas permitem o estabelecimento de árvores directas, em vez das árvores invertidas usadas habitualmente pelos protocolos de encaminhamento *multicast* tradicionais, relevando a importância das assimetrias das ligações num ambiente que é essencialmente unidireccional. Um dos principais elementos deste protocolo de encaminhamento *multicast* é o protocolo de encaminhamento *unicast* subjacente, que terá obrigatoriamente que ser capaz de encontrar uma rota por cada classe de serviço considerada, uma vez que para construir um novo ramo por cada classe de serviço, o MCMRP consulta a tabela de encaminhamento *unicast*. Para satisfazer esta nova necessidade, foi também concebido e implementado um novo protocolo de encaminhamento *unicast* por classes de serviço, o CoSLSP (Class of Service Link State Protocol). Ambos os protocolos foram implementados usando o Network Simulator. Os resultados obtidos através das simulações foram analisados e comparados com os de outros protocolos.

A principal hipótese subjacente a este trabalho é a de que, nas redes DiffServ, uma estratégia de encaminhamento por classes de serviço pode complementar eficientemente, ao nível do encaminhamento, a diferenciação de tráfego efectuada ao nível dos nós.

Palavras Chave: Encaminhamento com Qualidade de Serviço, Multicast, Serviços Diferenciados, Encaminhamento por Classes de Serviço.

Abstract

Most of current multicast QoS routing proposals are based on the principle that QoS routes must be computed for each request, where requests explicitly express their resource requirements. As a result, with this environment, the goal of QoS routing is to satisfy individual request requirements, resorting to resource reservation mechanisms to maintain those requirements after a feasible path has been found. This type of strategy is suited within the IntServ model but does not seem adequate in presence of DiffServ networks. According to DiffServ model, traffic flows are aggregated into specific classes-of-service and each flow receives a specific treatment accordingly to its class-of-service. There are no per flow guarantees, only per class differentiation. In this environment instead of per flow path computation, per class path calculation should be made, and so, within multicast scenarios, multiple multicast trees must be computed in order to satisfy different QoS requirements of different traffic classes.

This thesis proposes MCMRP (Multi Class Multicast Routing Protocol), a new multicast routing protocol enabling per class multicast tree computation. The proposed heuristics enable directed distribution trees establishment, instead of reverse path ones, due to the importance of link asymmetry within an environment which is, essentially, unidirectional. One of the major elements of this multicast routing protocol is the unicast routing protocol in use. Although MCMRP is independent of the underlying unicast routing protocol, it must be a multi-class enabled unicast routing protocol. In other words, the unicast routing protocol must be able to find the unicast routes that can meet the QoS requirements of each Class of Service. In order to build a new tree branch for each Class of Service the multicast routing protocol will search the unicast routing table for the unicast path that is more adequate to satisfy the QoS requirements of each class. To accomplish this new feature, a new unicast routing protocol called CoSLSP (Class of Service Link State Protocol) is proposed and implemented. Both protocols are implemented and simulated using Network Simulator. A set of simulation results are presented, analyzed and compared.

The main assumption behind this work is that a per class path computation can complement, at routing level, the node level differentiation techniques, in the provision of per class differentiated handling.

Keywords: Quality of Service Routing, Multicast, Differentiated Services, Class-of-Service Routing

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Este capítulo tem como objectivo enquadrar o trabalho desenvolvido na área das comunicações por computador e mais especificamente nas sub-áreas do Encaminhamento com Qualidade de Serviço e do Encaminhamento Multicast. Na primeira secção deste capítulo é definido e caracterizado o problema a tratar. Na segunda secção são enumerados os principais objectivos do trabalho e apresentadas as diferentes contribuições. Por fim é feita uma descrição da estrutura e organização deste documento.

1.1 Enquadramento

As redes de computadores permitem-nos transportar e partilhar informação e recursos. O seu alcance pode ser uma divisão ou um edifício, mas também podem cobrir um *campus*, uma cidade e até o mundo inteiro. A Internet é hoje em dia a maior rede de computadores e permite a partilha e troca de informação entre utilizadores de todo o mundo.

Apesar de ser habitualmente caracterizada através de diferentes expressões com uma semântica mais ou menos complicada (de que são exemplo as expressões: aldeia global, ciberespaço, etc), a Internet é basicamente uma rede de computadores à dimensão mundial. Como tal pode ser encarada como uma colecção de sistemas terminais que comunicam entre si utilizando uma infra-estrutura de comunicações comum. Esta infra-estrutura é constituída por inúmeras ligações (*links*) que usam diferentes tecnologias de transmissão e que estão por sua vez ligadas entre si através de dispositivos capazes de encaminharem o tráfego através delas, e que por isso se designam por encaminhadores (*routers*).

O desenvolvimento da tecnologia tanto ao nível dos sistemas terminais como ao nível da própria infra-estrutura de comunicações permitiu um enorme crescimento desta rede de computadores. Em particular, o aumento da capacidade de processamento nos sistemas terminais fez com que aparecessem aplicações mais poderosas capazes, por exemplo, de manipular diferentes tipos de informação: texto, áudio, imagens e vídeos. São as chamadas aplicações multimédia. As tecnologias de acesso também evoluíram permitindo que cada

vez mais utilizadores se ligassem à Internet. Ao nível da infra-estrutura de comunicações o aumento da capacidade de processamento dos encaminhadores possibilitou o aparecimento de tecnologias de transmissão mais rápidas.

Tudo isto fez com que fossem introduzidos novos requisitos. Tradicionalmente o único que existia era o da conectividade, ou seja, só era necessário garantir que as entidades conseguiam comunicar entre si. Hoje em dia em muitos casos (dependendo da aplicação) é necessário garantir que os dados são enviados a tempo, a uma determinada taxa e com um número limitado de perdas. Estes requisitos designam-se por requisitos de Qualidade de Serviço.

No entanto, os protocolos da Internet disponibilizam apenas um serviço designado por serviço de "melhor esforço" (*best effort*), a todos os tipos de tráfego, independentemente dos requisitos da aplicação que o gera. Como o próprio nome sugere este modelo de serviço não oferece qualquer tipo de garantia limitando-se a fazer o melhor possível dependendo do estado da rede. Os recursos são partilhados pelos vários fluxos de tráfego activos de forma mais ou menos equitativa, não havendo à partida mecanismos que permitam tratar melhor algum tipo de tráfego em detrimento de outro.

A transformação da Internet numa infra-estrutura comercial, aliada ao aparecimento e crescente divulgação das aplicações multimédia, fez com que o serviço do tipo "melhor esforço" fornecido pela Internet se tornasse claramente insuficiente. Tornou-se notória a necessidade de criação de mecanismos que garantissem que aplicações/utilizadores mais exigentes dispusessem dos recursos necessários, ou pelo menos de mais recursos que aplicações/utilizadores menos exigentes. A implementação desses mecanismos designa-se por Qualidade de Serviço (QoS).

A Qualidade de Serviço pode ser implementada em diferentes níveis da pilha OSI[1]. Alguns investigadores defendem que a complexidade associada à implementação da Qualidade de Serviço na Internet deverá ser empurrada para as próprias aplicações ou pelo menos para os níveis superiores da pilha OSI, mantendo a rede tal qual como está, o mais simples possível. Outros há que acreditam que melhores resultados se poderão obter se forem acrescentadas funcionalidades ao nível da rede (nível IP) tornando-a capaz de oferecer outros modelos de serviço além do do "melhor esforço". Segundo esta abordagem a Qualidade de Serviço é implementada através de mecanismos que estão integrados nos próprios encaminhadores, mecanismos esses responsáveis pela gestão de recursos de rede, ou seja, por controlar a forma como os dados dos vários fluxos de informação interagem e partilham recursos como o tempo de transmissão, o espaço de armazenamento, etc.

Existem diferentes tipos de mecanismos que implementam a Qualidade de Serviço ao nível da rede. Alguns dos mais importantes são:

- O escalonamento de pacotes (*schedulling*). Este mecanismo determina como é que os diferentes fluxos são re-transmitidos nas ligações de saída de um encaminhador. A ordem pela qual os pacotes são transmitidos determina de certa forma o atraso e a taxa de transmissão experimentada por cada fluxo de tráfego (um fluxo neste con-

texto significa um grupo lógico de pacotes de dados que pertence à mesma associação estabelecida entre uma determinada fonte e um determinado destino). Geralmente, uma política de escalonamento deve cumprir um conjunto de objectivos: isolamento de fluxos, ou seja deverá proteger determinado fluxo do escalonamento de outro fluxo, justiça, ou seja, deverá garantir o acesso aos recursos em proporção com a importância do tráfego, e adicionalmente deverá proporcionar uma utilização eficiente dos recursos e ser fácil de implementar.

- A reserva de recursos. É um mecanismo de controle que estabelece informação de estado ao longo do caminho utilizado por um fluxo. Essa informação de estado tem como objectivo informar os vários encaminhadores da quantidade de recursos necessária e reservada para determinado fluxo. Essa informação é utilizada na altura em que os encaminhadores aceitam ou recusam novos fluxos (reservas). A informação de estado deve assim manter-se durante todo o tempo de vida de um fluxo, para ser possível garantir que os requisitos de Qualidade de Serviço do fluxo são satisfeitos.

- O controle de admissão. Este mecanismo garante que os recursos não vão ser sobre-alocados. Para que o escalonamento de pacotes consiga oferecer garantias, e para que a reserva de recursos possa ser implementada, é necessário que os recursos não estejam sobre-alocados. O controle de admissão mediante o estado da rede, a quantidade de reservas, e as características de tráfego dos novos fluxos, deverá ser capaz de decidir quais os novos fluxos que é possível admitir de forma a que os requisitos dos novos fluxos possam ser satisfeitos e as garantias dadas aos fluxos anteriormente admitidos, mantidas.

No seio do IETF-Internet Engineering Task Force foram propostos alguns modelos, que se servem destes e de outros mecanismos com o objectivo de implementar o conceito de Qualidade de Serviço na Internet. São eles:

- O modelo de serviços integrados (IntServ[2]). Este modelo pressupõe uma reserva de recursos, ou seja, de acordo com este modelo as aplicações, por exemplo aplicações de tempo real, estabelecem as rotas e reservam os recursos ao longo das rotas estabelecidas, antes de começarem a transmitir pacotes.

- O modelo de serviços diferenciados (DiffServ[3]). Neste modelo os pacotes são marcados distintamente de forma a obter diferentes classes de tráfego. Os pacotes recebem um tipo de tratamento consoante a classe de tráfego a que pertencem.

- O Encaminhamento com Qualidade de Serviço (*QoS Routing*). No Encaminhamento com QoS o processo de descoberta dos caminhos é alterado de forma a considerar os requisitos de Qualidade de Serviço do tráfego.

Este último ponto é neste âmbito o mais importante já que neste trabalho são estudados e propostos mecanismos para implementar Qualidade de Serviço na Internet ao nível do

Encaminhamento. O encaminhamento com Qualidade de serviço pode ser implementado usando diferentes estratégias, entre as quais se salientam as seguintes:

- Cálculo dos caminhos a pedido, por cada fluxo (*per-flow routing*); Ao contrário do que é utilizado nos protocolos de encaminhamento tradicionais, este tipo de algoritmos pressupõe que as rotas sejam calculadas a pedido. No contexto do encaminhamento com QoS este modo de funcionamento é aceitável e até mais simples. Funciona tipicamente de acordo com um modelo de encaminhamento centralizado na fonte, ou seja, o encaminhador mais próximo da fonte, conhecedor dos requisitos de Qualidade de Serviço do fluxo, consulta a base de dados topológica que contém também informação referente ao estado da rede e calcula o melhor caminho que satisfaz os requisitos de QoS do fluxo. Os encaminhadores ao longo desse caminho terão depois que ser avisados e adicionalmente poderão efectuar uma reserva de recursos para garantir que os requisitos de Qualidade de Serviço são mantidos. Esta estratégia adequa-se muito bem ao modelo IntServ, onde os requisitos de Qualidade de Serviço são garantidos por fluxo. No entanto, à semelhança do que acontece com o modelo IntServ esta estratégia de encaminhamento apresenta alguns problemas, entre os quais o mais grave é o facto de não constituir uma solução escalável. Como vantagem pode considerar-se o facto de garantir de forma mais ou menos absoluta que os requisitos de Qualidade de Serviço são satisfeitos e mantidos ao longo de todo o tempo de vida do fluxo.

- Pré-cálculo de rotas alternativas. Para resolver o problema da escalabilidade do cálculo de rotas por fluxo, a pedido, existe a possibilidade de calcular e instalar rotas alternativas nas tabelas de encaminhamento. Este cálculo é normalmente feito independentemente de qualquer fluxo, e antes da chegada de qualquer pacote. Tem sido muito usado com o objectivo de balancear o tráfego na rede, enquadrando-se mais na área da Engenharia de Tráfego.

Dos vários modelos de Qualidade de Serviço propostos no seio do IETF, o modelo de serviços diferenciados tem sido apontado como um dos modelos mais promissores, principalmente pela sua escalabilidade. A diferenciação no tratamento dado ao tráfego de cada classe faz-se nó a nó (*PHB - Per Hop Behavior*[4]), através da aplicação de mecanismos de escalonamento adequados que utilizam filas de espera diferentes conforme a classe de tráfego a que pertence determinado pacote.

Este tratamento diferenciado não afecta o encaminhamento que continua a estabelecer o caminho mais curto para cada destino sem ter em conta os requisitos de qualidade de serviço das diferentes classes. No entanto, esse caminho mais curto não tem que obrigatoriamente ser o que melhor se adequa a transportar o tráfego de uma determinada classe.

Uma solução possível é, neste ambiente, procurar as rotas mais adequadas a satisfazer os requisitos de Qualidade de Serviço das diferentes classes de tráfego presentes numa arquitectura género DiffServ. Nesse caso, a estratégia poderá passar por calcular e instalar

diferentes rotas para as diferentes classes de tráfego (*per-class routing*). Ao contrário do que se passa no cálculo de rotas por fluxo, no cálculo de rotas por classe de serviço as rotas poderão pré-calculadas, exactamente como se verifica no encaminhamento tradicional.

1.1.1 Encaminhamento com QoS

A Internet é uma rede de comutação de pacotes. Como tal, a informação é codificada em sequências de bits, organizadas em pacotes, e transportada através da Internet desde a fonte até ao destino. Os vários pacotes, que podem constituir um fluxo, não têm obrigatoriamente que seguir o mesmo percurso nem que ser entregues na mesma ordem pela qual são enviados.

Ao longo do percurso é necessário tomar decisões de encaminhamento, nomeadamente escolher qual o caminho que deve seguir determinado pacote para atingir o seu destino. Os dispositivos que tomam essas decisões são, como já foi referido, os encaminhadores. Os encaminhadores, necessitam de trocar informação entre si que lhes permite tomar uma decisão (acertada) relativamente ao encaminhamento dos pacotes. Para isso usam protocolos que se designam por protocolos de encaminhamento (*routing protocols*). Em muitos casos existem diversos caminhos alternativos para atingir determinado destino. Nesses casos os protocolos de encaminhamento deverão escolher o caminho mais adequado ou partilhar caminhos alternativos de acordo com determinado critério. Tipicamente cada ligação entre dois encaminhadores tem um custo atribuído e o objectivo dos protocolos de encaminhamento consiste em minimizar o custo dos caminhos escolhidos. Este custo pode ser atribuído administrativamente ou mais frequentemente pode ser função de métricas que mudam dinamicamente em função do estado da rede.

Quando estamos perante tráfego com requisitos de Qualidade de Serviço, idealmente os protocolos de encaminhamento deverão procurar caminhos tomando em consideração esses requisitos. O problema a resolver pode ser descrito da seguinte forma: encontrar um caminho entre uma fonte e um destino que satisfaça um conjunto de requisitos de Qualidade de Serviço. Adicionalmente, os algoritmos de encaminhamento com QoS podem também considerar o problema de optimização da utilização de recursos.

O encaminhamento com QoS é complexo devido a um conjunto de razões. Antes de mais múltiplos requisitos podem tornar o problema do encaminhamento um problema intratável, por exemplo encontrar a melhor rota que satisfaz dois requisitos independentes entre si é um problema NP-Completo. Em segundo lugar, as métricas usadas para implementar o encaminhamento com QoS, tais como a largura de banda disponível ou o atraso fim-a-fim, estão sempre a mudar graças à flutuação do tráfego e é difícil, sem introduzir demasiada perturbação, em redes de maior dimensão manter esses valores actualizados. O desempenho dos algoritmos de encaminhamento com QoS pode ser grandemente penalizado se a informação de estado da rede estiver desactualizada. Por outro lado não se pode estar estar constantemente a divulgar os valores actuais sob pena de congestionar a rede com mensagens de controle. Há por isso aqui um compromisso difícil de obter que é necessário ter em conta.

Não existe uma solução genérica para o problema do encaminhamento com QoS. Pelo contrário, as propostas são variadas e incluem estratégias diversificadas que vão desde instalar diferentes rotas para diferentes classes de serviço, instalar informação de estado por fluxo nos encaminhadores a soluções de engenharia de tráfego que procuram balancear a distribuição de tráfego na rede. Estas soluções têm diferentes graus de complexidade e todas elas apresentam vantagens e desvantagens umas em relação às outras.

1.1.2 Encaminhamento Multicast

As formas mais comuns de comunicação têm sido o *unicast* (onde a comunicação é feita de um emissor para um receptor) e o *broadcast* (onde a comunicação é feita de um emissor para todos os receptores). Entre estes dois extremos situa-se o *multicast* onde a comunicação é feita de um emissor para um conjunto seleccionado de receptores que pode ou não incluir o emissor.

O *multicast* suporta aplicações que envolvem simultaneamente vários intervenientes, onde se encaixam as aplicações que implementam a comunicação entre alguns ou todos os elementos de um grupo. Este tipo de aplicações está a tornar-se cada vez mais popular, nomeadamente as aplicações onde a comunicação é feita usando um conjunto diversificado de tipos de informação desde textos e imagens, até áudio e vídeo. Dentro deste tipo de aplicações, chamadas aplicações multimédia, enquadram-se a vídeo-conferência, as aplicações que suportam o trabalho cooperativo, o ensino à distância, etc. As aplicações desta natureza são normalmente aplicações muito exigentes em relação aos recursos de comunicação de que necessitam.

As comunicações em grupo podem ser suportadas pelos modelos *unicast* e *broadcast* mas qualquer um deles implica um desperdício significativo de recursos, nomeadamente da largura de banda. O *broadcast* implica o estabelecimento de demasiados canais de comunicação entre o emissor e um número limitado de receptores. Com o *unicast* o emissor tem que enviar uma cópia do pacote a transmitir para cada um dos receptores, mesmo que muitos deles possuam as mesmas rotas.

O modelo *multicast* consegue poupar recursos partilhando-os. Em vez de transmitir os pacotes do emissor para cada receptor separadamente, estabelece rotas que partilham as mesmas ligações de forma a transmitir os pacotes apenas uma vez enquanto isso for possível duplicando-os sempre que, e apenas, quando necessário. Mais concretamente, é o encaminhamento *multicast* que torna possível um melhor aproveitamento da largura de banda disponível ajustando dinamicamente os melhores caminhos entre os emissores e receptores.

Determinar caminhos com QoS adequados para todos os receptores de um grupo complica ainda mais o problema do encaminhamento com QoS. Também neste caso o cálculo das árvores pode ter como base os requisitos de qualidade de serviço de um determinado fluxo, ou os requisitos subjacentes a uma classe de serviço.

1.2 Objectivos e contribuições da tese

O principal objectivo deste trabalho é estudar, propor, realizar e testar soluções que permitam implementar o encaminhamento por classes de serviço ao nível do *multicast*.

Os protocolos de encaminhamento *multicast* constroem as árvores de distribuição de tráfego com base nos melhores caminhos descobertos pelos protocolos de encaminhamento *unicast*. Sendo assim, no contexto do trabalho desta tese foi necessário conceber também uma estratégia de encaminhamento *unicast* por classes de serviço.

Dadas as características das questões a tratar foi decidido que o recurso à simulação seria inevitável. O simulador escolhido foi o Network Simulator (NS)[5], por ser um ambiente de simulação largamente utilizado pela comunidade científica internacional no estudo de protocolos de comunicações.

As diferentes contribuições deste trabalho estão sintetizadas nos seguintes pontos:

- Implementação do PIM-SM no Network Simulator[6]. De entre os vários protocolos encaminhamento *multicast* existentes, o PIM-SM[7] é umas das propostas mais promissoras concebida para redes alargada e já largamente desenvolvido por diversos fabricantes. Ao longo deste trabalho o protocolo PIM-SM foi usado para comparar e estabelecer o valor acrescentado das diferentes propostas efectuadas ao nível do encaminhamento *multicast*. O Network Simulator (NS) não inclui (nem é conhecida qualquer contribuição que inclua) implementações do protocolo PIM-SM. Assim sendo, no âmbito desta tese, iniciou-se o trabalho experimental com a implementação do PIM-SM, com o duplo objectivo de familiarização com o simulador e a obtenção de uma implementação do PIM-SM para depois comparar o seu comportamento e desempenho com os diferentes protocolos propostos.

- Concepção, implementação e validação do protocolo DTMP (Directed Trees Multicast Protocol)[8]. Nos protocolos de encaminhamento *multicast* tradicionais é usado o conceito de *reverse path routing* na construção das árvores de distribuição de tráfego *multicast*. Este conceito não se adequa a ambientes em que é necessário considerar requisitos de Qualidade de Serviço, porque tipicamente, nesses ambientes as condições das ligações entre dois nós não são simétricas. O DTMP é um protocolo de encaminhamento *multicast* baseado no PIM-SM, em que as árvores *multicast* são construídas a partir do *Rendez-Vous Point* (RP) (ou fonte) em direcção ao novo membro. Este protocolo foi concebido para servir de base ao protocolo que implementa o encaminhamento *multicast* por classes de serviço.

- Concepção, implementação e validação de um protocolo de encaminhamento *unicast* (CosLSP - Class of Service Link State Protocol) capaz de calcular rotas dependendo dos requisitos de qualidade de serviço de diferentes classes de tráfego[9] Esta implementação é baseada no protocolo OSPF[10] com as seguintes modificações principais:

  - além de se divulgar o custo da ligação divulga-se também o estado de cada uma

das diferentes filas de espera em relação aos requisitos de qualidade de serviço considerados;

- o algoritmo de Dijkstra usado no cálculo da rota mais curta foi alterado no sentido de calcular a rota mais curta que respeita os requisitos de qualidade de serviço de cada uma das classes

Este protocolo de encaminhamento *unicast* é uma das peças fundamentais no modelo proposto, uma vez que o protocolo de encaminhamento *multicast* vai precisar de diferentes rotas alternativas (de preferência as que melhor se adequam aos requisitos de cada uma das classes de serviço) para a construção das diferentes árvores de distribuição *multicast*.

- Concepção, implementação e validação de um protocolo de encaminhamento *multicast* capaz de estabelecer árvores *multicast* dependendo dos requisitos de qualidade de serviço de diferentes classes de tráfego[11] [12]. A implementação deste protocolo utiliza as implementações dos dois protocolos referidos nos pontos anteriores (o DTMP e o CosLSP) para implementar o encaminhamento *multicast* por classes.

1.3 Organização do documento

Este documento foi estruturado em oito capítulos.

Este primeiro capítulo enquadra o trabalho desenvolvido, apresentando a motivação e os principais objectivos. É feito um resumo das contribuições e por último é apresentada a estrutura deste documento.

O segundo capítulo apresenta os conceitos e fundamentos. O objectivo é introduzir alguns conceitos fundamentais subjacentes ao encaminhamento e à qualidade de serviço em redes IP.

No terceiro capítulo apresenta-se o trabalho relacionado na área do encaminhamento *unicast* e *multicast* com Qualidade de Serviço.

No quarto capítulo é feita uma descrição do Network Simulator, nomeadamente da forma como o NS suporta os protocolos de encaminhamento *unicast* e *multicast*. Para fazer esta descrição ir-se-á usar como base uma implementação do protocolo de encaminhamento *unicast* LS, integrado na versão 2 do Network Simulator e uma implementação do protocolo de encaminhamento *multicast* PIM-SM, desenvolvida no contexto deste trabalho. Estes dois protocolos, o LS e o PIM-SM, são utilizados como referencial para comparação com os diferentes protocolos de encaminhamento *unicast* e *multicast* propostos no âmbito deste trabalho.

O quinto capítulo descreve um dos protocolos de encaminhamento *multicast* concebidos, implementados e avaliados no âmbito deste trabalho: o protocolo DTMP. Este protocolo foi implementado usando o Network Simulator. São apresentados e discutidos alguns resultados resultantes da simulação do DTMP. Este resultados são comparados

com resultados obtidos usando os mesmos cenários de teste utilizando a implementação do PIM-SM.

O sexto capítulo apresenta o protocolo CoSLSP, um protocolo de encaminhamento *unicast* por classes de serviço. O CoSLSP foi implementado e avaliado com o Network Simulator. Os resultados obtidos com as simulações efectuadas com o CoSLSP foram comparados com resultados obtidos com simulações efectuadas nas mesmas circunstâncias usando a implementação do protocolo LS.

No sétimo capítulo é apresentada uma proposta que tem como objectivo implementar o encaminhamento *multicast* por classes de serviço. Basicamente é esta proposta, já instanciada no protocolo MCMRP, que vem de encontro ao objectivo principal deste trabalho, o de estudar, conceber e testar estratégias de encaminhamento *multicast* por Classes de Serviço.

O oitavo capítulo apresenta as últimas conclusões do trabalho efectuado. É feita um síntese de todo o trabalho realizado, seguida de alguns comentários críticos e de uma previsão relativa ao trabalho futuro.

Capítulo 2

Conceitos e Fundamentos

Este capítulo introduz conceitos fundamentais subjacentes ao encaminhamento, nomeadamente ao encaminhamento em redes IP, e tem como objectivo familiarizar o leitor com a terminologia e conceitos usados nos capítulos seguintes.

Como já foi referido no capítulo anterior, este trabalho tem como objectivo estudar soluções que permitam implementar o encaminhamento por classes de serviço ao nível do encaminhamento *multicast*. Para concretizar esse objectivo foi necessário conceber também uma estratégia de encaminhamento *unicast* por classes de serviço.

Por esse motivo este capítulo foi divido em três secções principais: na primeira secção é abordado o encaminhamento *unicast* e os principais protocolos de encaminhamento *unicast* hoje em uso na Internet. A segunda secção é dedicada ao encaminhamento *multicast*. Por fim, na terceira secção são descritas sucintamente as principais estratégias utilizadas para implementar Qualidade de Serviço na Internet.

2.1 Encaminhamento Unicast

O encaminhamento *unicast* é a capacidade de transportar informação através de uma rede de uma fonte para um destino. Pelo caminho, tipicamente a informação passa por um ou mais nós intermédios. Este nós intermédios são designados por encaminhadores porque são eles os responsáveis pelo encaminhamento do tráfego.

O encaminhamento compreende duas actividades básicas: descobrir o melhor (ou melhores) caminho (*path*) entre vários possíveis e transportar grupos de informação (pacotes) através desse caminho. Naturalmente que a caracterização "melhor caminho" depende do conjunto de parâmetros específicos utilizados para o qualificar. A actividade que consiste em descobrir o melhor caminho é uma actividade complexa e existem múltiplas abordagens para a concretizar. Pelo contrário a actividade que consiste em transportar os pacotes de dados através de um caminho já estabelecido é bastante simples e apesar de existirem

algumas variantes, é implementada pelos diferentes protocolos de encaminhamento basicamente da mesma forma.

Para conseguirem determinar o melhor caminho, os algoritmos de encaminhamento inicializam e mantêm tabelas que são designadas por tabelas de encaminhamento. Os algoritmos de encaminhamento preenchem estas tabelas com informação variada. Tipicamente cada linha da tabela representa uma associação **destino/próximo salto** (*nexthop*). Esta associação significa que o encaminhador em causa deve enviar o pacote para o encaminhador representado pelo **próximo salto** para assim conseguir alcançar o **destino** através do melhor caminho possível. Quando um encaminhador recebe um pacote, verifica o endereço **destino** e tenta associar este endereço a um **próximo salto**, consultando para isso a tabela de encaminhamento.

Os encaminhadores comunicam uns com os outros para manter as tabelas de encaminhamento actualizadas, através da transmissão de mensagens. As mensagens de actualização variam consoante o algoritmo de encaminhamento utilizado: poderão incluir toda a tabela de encaminhamento (por exemplo, no caso dos protocolos RIP[13] e IGRP[14]), ou apenas um subconjunto dela (por exemplo, no caso do protocolo BGP[15]), ou então incluírem apenas informação relativa ao estado das ligações de um encaminhador (como acontece no protocolo OSPF[16]). Analisando estas mensagens vindas de todos os encaminhadores, um encaminhador procura determinar a topologia da rede.

Adicionalmente as tabelas de encaminhamento podem conter outro tipo de informação, por exemplo, métricas que caracterizam os caminhos. Nesse caso os encaminhadores comparam essas métricas com o objectivo de escolher os melhores caminhos. Estas métricas também variam de acordo com o algoritmo de encaminhamento usado.

Uma vez conhecida, total ou parcialmente a topologia de rede, os protocolos de encaminhamento utilizam um algoritmo de encaminhamento para calcular os melhores caminhos para todos os destinos possíveis preenchendo de seguida e em conformidade a tabela de encaminhamento.

Como já foi referido, ao contrário da actividade da descoberta do melhor caminho, os algoritmos que implementam o reenvio em direcção ao destino dos pacotes de informação, assim que estes chegam a um encaminhador, são relativamente simples e não variam substancialmente com os protocolos de encaminhamento. Na maior parte dos casos, ao receber um pacote, um nó (que não o destino final), deve reencaminhá-lo para outro nó. A fonte, envia o pacote endereçado especificamente para o endereço lógico de um encaminhador local, que representa o **próximo salto**, usando como endereço de rede o endereço do **destino** final. O encaminhador, ao receber o pacote, tem de averiguar se sabe ou não como atingir o destino deste, consultando a tabela de encaminhamento. Se não souber descarta o pacote e envia uma mensagem ICMP[17] de volta para a fonte. Se souber mais não faz do que reencaminhá-lo para a interface correspondente, substituindo o endereço lógico pelo endereço lógico do próximo encaminhador, e mantendo o endereço de rede do destino final. Assim, o endereço de rede **destino** de um pacote mantém-se durante todo o caminho, só mudando o endereço lógico, que é sempre o endereço lógico

do próximo encaminhador no caminho até ao destino.

Por vezes os protocolos de encaminhamento implementam algumas variantes destes algoritmos. Como exemplo, pode referir-se o caso do encaminhamento com caminhos múltiplos (*multipath routing*) (usado por exemplo pelo protocolo EIGRP[14]), e o caso do encaminhamento centralizado efectuado na fonte. No caso do encaminhamento com múltiplos caminhos, são descobertos e inseridos na tabela de encaminhamento caminhos alternativos para o mesmo destino. Este tipo de encaminhamento tem normalmente como objectivo proceder ao balanceamento de tráfego. Neste caso o algoritmo que implementa o reenvio dos pacotes nos encaminhadores deverá escolher alternativamente um dos caminhos que consta na tabela de encaminhamento para determinado destino. No caso do encaminhamento centralizado, o melhor caminho é especificado na fonte, e todos os encaminhadores deverão implementar o reenvio dos pacotes até ao destino de acordo com o que for estabelecido pela fonte.

2.1.1 Definições Matemáticas

O cálculo do melhor caminho é normalmente concretizado recorrendo a algoritmos baseados na teoria dos grafos[18]. A titulo de exemplo, nesta secção descreve-se matematicamente o algoritmo de Dijkstra[19]. Este algoritmo é um dos algoritmos mais utilizados pelos protocolos de encaminhamento *unicast* e foi o algoritmo que serviu de base à estratégia concebida para implementar o encaminhamento *unicast* no âmbito deste trabalho.

Da teoria dos grafos vamos usar as seguintes definições:

- Um grafo $G = (V, A)$ consiste num conjunto finito não vazio V e numa colecção A de pares não ordenados de V. Cada elemento do conjunto V é designado por vértice do grafo G, e cada par não ordenado pertencente a A é designado por aresta do grafo G. Uma aresta $a = x, y$ é uma aresta entre os dois vértices x e y.
- Um grafo G é um grafo dirigido se consiste num conjunto finito V de vértices e numa colecção A de pares **ordenados** de V que nesse caso se designam por arcos.
- Dois vértices de V dizem-se adjacentes quando há um arco ou aresta a uni-los e não adjacentes em caso contrario.
- Um grafo G é um grafo completo se todos os vértices que o constituem são adjacentes, ou seja, se existe sempre uma aresta a unir qualquer par de vértices de V.
- Um grafo G é um grafo pesado se existir uma função w que associe a cada aresta pertencente a A um número real. Nesse caso, o número $w(a)$ é designado por peso da aresta a.
- Um caminho entre o vértice $x_1 \in V$ e o vértice $x_r \in V$ de um grafo é uma sequência $x_1,\ e_1,\ x_2,\ e_2,\ x_3,\ e_3, ... e_{r-1}, x_r$, onde $x_1, x_2, x_3, ..., x_r$ são vértices do grafo e e_k

é uma aresta entre x_k e x_{k+1}. Um caminho é habitualmente representado na forma $< x_1, x_2, x_3, ..., x_k >$, isto é sem mencionar explicitamente as arestas.

- Num caminho não existem vértices repetidos e consequentemente não existem arestas repetidas, ou arcos repetidos se consideramos os grafos dirigidos.
- O comprimento de um caminho de um grafo G é o número de arestas que o constituem. Se estivermos perante um grafo pesado o comprimento de um caminho é a soma dos pesos associados às várias arestas ou arcos que o constituem.
- Se x e y forem dois vértices distintos de um grafo pesado G, o caminho mais curto ente x e y designa-se por caminho de comprimento mínimo. Ao comprimento do caminho mais curto entre x e y chama-se distância entre x e y. O "Problema dos Caminhos Mais Curtos" (*Shortest Path Problem*, na terminologia Inglesa) consiste em encontrar este tipo de caminhos (os mais curtos) entre os diferentes vértices que constituem um grafo.

Uma rede de comunicações pode ser modelada por um grafo pesado $G = (V, A)$. Os vértices V do grafo representam os encaminhadores e os sistemas terminais da rede (ou seja os nós da rede), e as arestas A representam as ligações entre eles. Os pesos associados a cada aresta são os custos das diferentes ligações. Se as ligações forem simétricas, ou seja se as ligações tiverem o mesmo custo nos dois sentidos o grafo G é um grafo não dirigido, caso contrário, ou seja se as ligações forem assimétricas, será um grafo dirigido. O problema do encaminhamento em redes pode desta forma ser resolvido recorrendo ao mesmo tipo de soluções encontradas para o "Problema dos Caminhos Mais Curtos" da Teoria dos Grafos.

Como já foi referido, um dos algoritmos mais usados pelos protocolos de Encaminhamento para determinarem os caminhos mais curtos é o algoritmo de Dijkstra. Este algoritmo é um algoritmo muito simples capaz de calcular de uma só vez os caminhos mais curtos para todos os destinos possíveis. O seu modo de funcionamento consiste em calcular de forma incremental uma árvore com os caminhos mais curtos para todos os destinos. Apresenta-se a seguir uma descrição Matemática deste algoritmo.

Seja $G = (N, L)$ um grafo que representa uma rede com N nós e L ligações. Seja c_{ij} o custo da ligação entre os nós i e j (se não existir nenhuma ligação entre dois nós i e j então $c_{ij} = \infty$). Seja $Adj(i)$ o conjunto de todos os nós adjacentes de i ($Adj(i) = \{j : l = (i, j) \in L\}$). Seja s o nó origem. Sejam D_k a distância entre o nó origem e o nó k e $Pred_k$ o nó que antecede o nó k no melhor caminho para chegar do nó origem ao nó k.

Começar
$P = \{s\}$; $T = N \setminus \{s\}$;
$D_s = 0$;
$\forall\, k \in Adj(s) : D_k = c_{sk}$; $Pred_k = s$;
$D_k = \infty\, for\, other\, nodes$;

Enquanto $P \neq N$ **fazer**
Encontrar $k \in T : D_k = \min_{i \in T} D_i$
$P = P \cup \{k\}$
$T = T \setminus \{k\}$
$\forall\, j\, in\, Adj(k)$, **Se** $D_j > D_k + c_{kj}$ **Então**
$D_j = D_k + c_{kj}$;
$Pred_j = k$;
Fim EnquantoFazer
Fim

Começa-se por inserir o nó origem num conjunto P, inicialmente vazio, excluindo-o do grafo N. Depois consideram-se as distâncias iguais aos custos das ligações apenas para os nós adjacentes ao nó origem, considerando a distância igual a ∞ para todos os outros nós. De entre os diferentes nós que ainda não pertencem a P escolhe-se o que tem o menor custo (que na primeira iteração há-de ser de certeza um nó adjacente ao nó origem), incluindo-o no conjunto P e excluindo-o do grafo N. Depois é necessário encontrar as distâncias mínimas do nó origem aos nós adjacentes do nó escolhido. Uma vez essas distâncias encontradas, volta-se a escolher entre todos os nós que ainda não pertencem ao conjunto P, o que tem a menor distância. Este processo repete-se até que façam parte do conjunto P todos os nós do grafo.

2.1.1.1 Exemplo

A título de exemplo vamos considerar uma topologia muito simples, constituída por seis nós, que está representada na figura 2.1. Se executarmos o algoritmo de Dijkstra assumindo que a fonte é o nó 1 da topologia, os resultados de cada uma das iterações (5 no total) podem ser visualizados na tabela 2.1. A figura 2.2 representa o grafo dos caminhos mais curtos a partir do nó 1 para todos os outros nós que constituem a topologia.

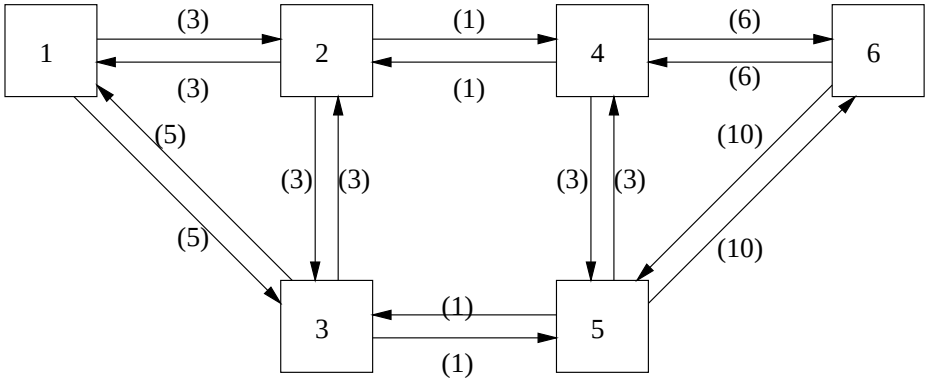


Figura 2.1: Exemplo de Topologia de Rede com 6 nós

Iteração	P	T	Distâncias	Predecessores
1	1	2, 3, 4, 5, 6	$D_2 = 3$	$Pred_2 = 1$
			$D_3 = 5$	$Pred_3 = 1$
2	1, 2	3, 4, 5, 6	$D_2 = 3$	$Pred_2 = 1$
			$D_3 = min\{5, 6\} = 5$	$Pred_3 = 1$
			$D_4 = 4$	$Pred_4 = 2$
3	1, 2, 4	3, 5, 6	$D_2 = 3$	$Pred_2 = 1$
			$D_3 = 5$	$Pred_3 = 1$
			$D_4 = 4$	$Pred_4 = 2$
			$D_5 = 7$	$Pred_5 = 4$
			$D_6 = 10$	$Pred_6 = 4$
4	1, 2, 4, 3	5, 6	$D_2 = 3$	$Pred_2 = 1$
			$D_3 = 5$	$Pred_3 = 1$
			$D_4 = 4$	$Pred_4 = 2$
			$D_5 = min\{7, 6\} = 6$	$Pred_5 = 3$
			$D_6 = 10$	$Pred_6 = 4$
4	1, 2, 4, 3, 5	6	$D_2 = 3$	$Pred_2 = 1$
			$D_3 = 5$	$Pred_3 = 1$
			$D_4 = 4$	$Pred_4 = 2$
			$D_5 = 6$	$Pred_5 = 3$
			$D_6 = min\{10, 16\} = 10$	$Pred_6 = 4$
5	1, 2, 4, 3, 5, 6		$D_2 = 3$	$Pred_2 = 1$
			$D_3 = 5$	$Pred_3 = 1$
			$D_4 = 4$	$Pred_4 = 2$
			$D_5 = 6$	$Pred_5 = 3$
			$D_6 = 10$	$Pred_6 = 4$

Tabela 2.1: Algoritmo de Dijkstra

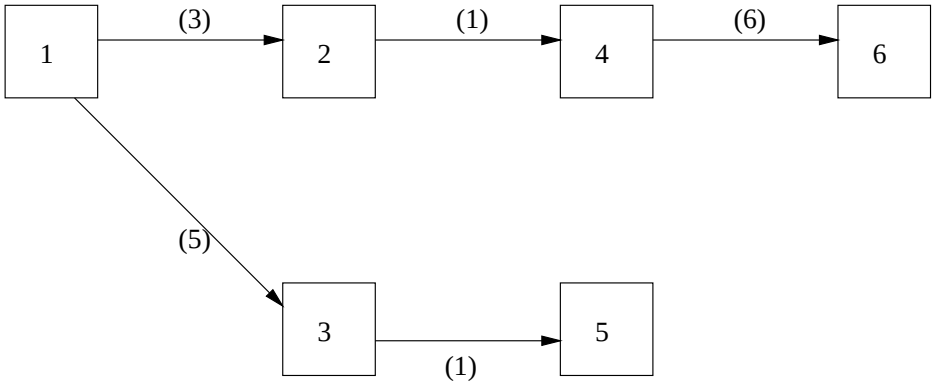


Figura 2.2: Resultado da execução do Algoritmo de Dijkstra

2.1.2 Métricas

As tabelas de encaminhamento podem conter outro tipo de informação além das associações destino/próximo salto, como por exemplo, métricas que caracterizam os caminhos.

Nesse caso os encaminhadores comparam essas métricas para escolher os melhores caminhos. As métricas utilizadas dependem do algoritmo de encaminhamento usado. Algoritmos mais sofisticados podem basear as suas escolhas em mais do que uma métrica combinando-as ou não num único valor.

O custo total do caminho contabilizado pelo número de ligações utilizadas é a métrica mais comum. Esta métrica designa-se por número de saltos. Um salto corresponde a uma ligação entre dois encaminhadores vizinhos. O número de saltos de um caminho corresponde ao número total de ligações que o constituem. Alguns protocolos de encaminhamento permitem que seja atribuído por configuração um custo a cada ligação. Neste caso, o custo total do caminho, também designado por tamanho do caminho, corresponde ao somatório dos custos de todas as ligações que constituem o caminho.

Outras métricas possíveis são aquelas que reflectem o estado da rede, como por exemplo a largura de banda disponível, o atraso fim-a-fim, a taxa de perdas, etc. A utilização deste tipo de métricas está directamente relacionada com a incorporação de mecanismos de Qualidade de Serviço no processo de encaminhamento. Como tal, remete-se este assunto para o capítulo 3.

2.1.3 Algoritmos de Encaminhamento Unicast

Os protocolos de encaminhamento *unicast* apresentam diferentes características que nos permitem distingui-los uns dos outros. Os encaminhadores podem ser pré-programados com rotas estáticas ou podem calcular rotas dinamicamente usando um de vários algoritmos de encaminhamento dinâmicos. Os algoritmos de encaminhamento dinâmicos são usados pelos encaminhadores para implementar a descoberta de caminhos, caminhos esses que são usadas pelos encaminhadores para expedir pacotes ou datagramas.

Existem várias maneiras de classificar os protocolos de encaminhamento. Uma delas é classificá-los de acordo com a forma como os vários encaminhadores descobrem e calculam novas rotas. Se usarmos esse critério podemos dividir os protocolos de encaminhamento em duas categorias[20][21]: vector de distância e estado de ligação.

Outro critério comum usado para distinguir os vários protocolos de encaminhamento é o seu domínio de utilização. Neste caso existem também duas categorias[21][22]: os protocolos que são usados para implementar o encaminhamento dentro de um sistema autónomo (*intra-domain*), e os protocolos que implementam o encaminhamento entre sistemas autónomos (*inter-domain*).

Existem outras possibilidades, tais como: algoritmos centralizados versus distribuídos, algoritmos planos versus hierárquicos, algoritmos de um único caminho versus algoritmos de múltiplos caminhos, etc.

2.1.3.1 Encaminhamento Estático ou Dinâmico

A maior parte do protocolos de encaminhamento utilizados hoje em dia são protocolos dinâmicos que se adaptam às circunstancias analisando mensagens de actualização provenientes de outros encaminhadores. Se uma determinada mensagem sugere que houve alguma alteração, um encaminhador recalcula as suas rotas e envia mensagens de actualização aos encaminhadores vizinhos. Os protocolos de encaminhamento dinâmicos podem ser complementados com rotas estáticas sempre que tal for apropriado.

No encaminhamento estático as rotas são pré-programadas manualmente pelo administrador da rede. Ou seja, a responsabilidade de descobrir e divulgar as melhores rotas é deixada a cargo de um administrador. Existem algumas vantagens, mas principalmente muitos inconvenientes na utilização de rotas estáticas. Resumidamente, pode-se afirmar que existem algumas situações em que a sua utilização se justifica.

Como principal desvantagem aponta-se o facto de, em caso de falha na rede, ou outro tipo de alteração da topologia, o administrador da rede ter que, manualmente, reagir a estas alterações. No entanto em redes muito pequenas com apenas um caminho possível para qualquer destino a utilização de encaminhamento estático pode ser satisfatória e traz como vantagem a redução do consumo de recursos de largura de banda, ocupação de memória ou tempo de processamento, usados para descobrir rotas ou para comunicar com outros encaminhadores.

Por vezes as rotas estáticas podem ser úteis por questões de segurança, quando se pretende por exemplo fazer passar todos os pacotes por um servidor que implemente um qualquer mecanismo de segurança. Neste caso a solução pode passar por encaminhar todos os pacotes para esse servidor, através de encaminhamento estático.

No encaminhamento dinâmico as rotas são calculadas automaticamente e reflectem parâmetros dinâmicos da rede, tais como a topologia, o estado operacional ou até o nível de tráfego ou a taxa de erros.

2.1.3.2 Estado da Ligação ou Vector de Distância

Os algoritmos de estado de ligação (também chamados algoritmos do caminho mais curto) divulgam a informação de encaminhamento para todos os nós da rede. No entanto, cada encaminhador divulga apenas a parte da tabela de encaminhamento que descreve o estado das suas próprias ligações.

Pelo contrário nos algoritmos de vector de distância (também chamados algoritmos de Bellman-Ford, em homenagem aos seus criadores), cada encaminhador envia parte ou toda a sua tabela de encaminhamento apenas aos encaminhadores vizinhos.

Os algoritmos de estado de ligação convergem mais rapidamente do que os de vector de distância, mas em contrapartida requerem mais CPU e memória.

Vector de Distância

Os algoritmos de vector de distância baseiam o seu funcionamento num vector, onde é armazenada a informação relativa à "melhor distância" conhecida para cada um dos destinos possíveis.

Cada encaminhador mantém um destes vectores, na forma de tabela de encaminhamento. As tabelas de encaminhamento contêm além dos endereços destino possíveis e das "melhores distâncias" para lá chegar, os "próximos saltos", ou seja, os encaminhadores vizinhos que estão no caminho até aos destinos e as interfaces de rede através das quais se atingem esses encaminhadores vizinhos.

A distância é um conceito generalizado que não é mais do que uma métrica ou um conjunto de métricas que permite medir coisas como o tempo que se demora a chegar a um determinado destino, o número de saltos do percurso, a largura de banda disponível nas várias ligações que constituem o caminho, etc. É esta distância que se pretende optimizar.

Cada encaminhador dá a conhecer aos seus vizinhos a sua tabela de encaminhamento através de mensagens. Quando um determinado encaminhador recebe uma destas mensagens procura actualizar a sua tabela de encaminhamento em conformidade com a mensagem recebida. Como é óbvio antes de efectuar a comparação entre a tabela de encaminhamento do encaminhador vizinho e a sua, o encaminhador necessita de somar a todas as distâncias contidas na tabela de encaminhamento do vizinho, a distância entre ele e o encaminhador vizinho.

A titulo de exemplo, suponhamos que o encaminhador R1 contém na sua tabela de encaminhamento uma rota para um determinado destino D1 através do encaminhador R2 com a distância de x. Convém distinguir duas situações possíveis:

- Se R1 recebe uma mensagem proveniente de R2 com uma rota para D1, a distância da rota para D1 na tabela de encaminhamento de R1 tem que ser sempre actualizada em conformidade. Neste caso soma-se à distância da rota para D1 contida na tabela de encaminhamento de R2 a distância entre R1 e R2 e é essa distância que se insere na entrada relativa a D1 da tabela de encaminhamento de R1, mantendo-se R2 como o "próximo salto".
- Se R1 recebe uma mensagem proveniente de outro vizinho diferente de R2, a rota para D1 só será actualizada se a soma da distância para o mesmo destino contida na tabela de encaminhamento recebida com a distância entre R1 e o encaminhador que enviou a mensagem de actualização for menor do que x.

Um protocolo de encaminhamento baseado num algoritmo de vector de distância tem assim que implementar as seguintes funcionalidades:

- Cada encaminhador mantém uma tabela de encaminhamento que terá de conter uma

entrada para cada um dos destinos possíveis. A entrada contém a distância para o respectivo destino bem como o próximo encaminhador no percurso até lá chegar.

- Periodicamente os encaminhadores enviam as suas tabelas de encaminhamento para os seus vizinhos em forma de mensagem de actualização. Quando um encaminhador R1 recebe uma mensagem de actualização de um encaminhador vizinho R2, a primeira coisa que faz é somar todas as distâncias contidas na tabela de encaminhamento de R2, com a distância entre R1 e R2.

- Depois compara as distâncias obtidas com as da sua própria tabela de encaminhamento, uma a uma. Se encontra uma mais pequena, é necessário actualizar a rota respectiva. Neste caso a rota recebe uma actualização não só no campo da distância mas também no próximo salto que passa a ser o encaminhador que enviou a mensagem de actualização. Se a distância contida na tabela de encaminhamento de R1 for menor que a soma da distância recebida na mensagem de actualização de R2 e a distância entre R1 e R2, a rota só é actualizada se o próximo salto corresponder ao encaminhador que enviou a mensagem de actualização, neste caso se o próximo salto for R2.

Estado de Ligação

Em vez do cálculo distribuído e incremental efectuado pelos algoritmos de Vector de Distância, os algoritmos de Estado de Ligação baseiam o seu funcionamento numa base de dados onde é mantida a topologia completa da rede, e o estado de todas as ligações. Esta base de dados designa-se por Base de Dados Topológica.

Cada encaminhador mantém uma réplica da Base de Dados Topológica e contribui para a sua actualização divulgando o estado interno de todas as suas ligações. Estes anúncios são designados por LSAs (Link-Sate Advertisements) e são difundidos por todos os encaminhadores da rede que desta forma mantêm actualizada a sua réplica da Base de Dados Topológica. Por outras palavras, a base de dados topológica de um encaminhador é o resultado da concatenação dos LSAs enviados por todos os encaminhadores que constituem a topologia.

Depois, em cada nó, é apenas necessário usar um algoritmo de caminho mais curto, tipicamente o algoritmo de Dijkstra [1] ou uma variante, que utiliza a Base de Dados Topológica para descobrir os caminhos mais curtos para todos os nós presentes na topologia. Esta informação é usada para construir e actualizar as tabelas de encaminhamento.

2.1.3.3 Centralizados ou Distribuídos

Nos protocolos de encaminhamento centralizados, é normalmente, a fonte que determina todo o caminho até ao nó destino. Nestes sistemas de encaminhamento, os encaminhadores

[1] O algoritmo de Dijkstra foi descrito na secção 2.1.1 deste capítulo.

actuam apenas como dispositivos do tipo *store and forward* limitando-se a reenviar o pacote para o próximo encaminhador num caminho pré-estabelecido. Para isso, todos os nós (pelos menos todos os que podem assumir o papel de fontes) conseguem determinar caminhos porque possuem um conhecimento completo da topologia da rede. Os caminhos são por isso calculados localmente, e antes de ter inicio a transmissão de dados, é enviada uma mensagem de controle ao longo do caminho determinado, mensagem essa que avisa todos os nós intermédios do caminho que fica assim estabelecido. Pode ser usado um protocolo de estado de ligação para actualizar o estado global da rede em todos os nós.

Embora tenha grandes problemas, a maior parte deles relacionados com a escala, o encaminhamento centralizado apresenta algumas vantagens que podem justificar a sua utilização em determinadas circunstâncias. A maioria destes algoritmos são conceptualmente simples e fáceis de implementar e avaliar além de garantirem a ausência total de ciclos. Como irá ser referido mais adiante neste capítulo e no próximo, alguns problemas de encaminhamento são NP-completos[2]. Nessas situações é bem mais fácil conceber heurísticas centralizadas do que distribuídas.

No encaminhamento distribuído, o cálculo dos caminhos é feito de forma distribuída pelos encaminhadores que se encontram entre a fonte e o destino. São trocadas mensagens de controle entre os vários encaminhadores e a informação de estado mantida em cada nó é usada de forma colectiva no cálculo dos caminhos. Alguns desses algoritmos continuam a necessitar que cada nó mantenha o estado global da rede, o que pode ser conseguido através da utilização de um protocolo de estado de ligação. Outros optam pela utilização de um protocolo de vector de distância, onde a quantidade de informação de estado é menor. No encaminhamento distribuído, o encaminhamento de pacotes é implementado em cada nó com base na informação de estado. Não apresenta problemas de escala tão graves como o encaminhamento centralizado embora não os resolva totalmente, pois continua a exigir que em cada nó exista um conhecimento mais ou menos completo da topologia de rede e do estado de todas as ligações. Como alternativa, existe outro tipo de algoritmos também distribuídos, que baseiam o seu funcionamento numa técnica designada por *flooding* na terminologia inglesa. Esta técnica não exige a manutenção de informação de estado global, apenas estado local. Em contrapartida implica um número adicional de mensagens de controle. Quando uma fonte pretende começar a transmitir, envia um conjunto de mensagens de controle que são, digamos assim, espalhadas pela rede, mensagens essas que irão seguir diferentes caminhos até ao destinatário coleccionando pelo caminho o estado das diferentes ligações que atravessam. Mediante esta informação é escolhido e estabelecido o melhor caminho. Este tipo de abordagem apesar de reduzir drasticamente a informação de estado que é necessário manter em cada nó introduz uma sobrecarga adicional provocada pela grande quantidade de mensagens de controle introduzidas na rede.

[2]Nondeterministic in Polynomial time complete, uma medida de complexidade computacional

2.1.3.4 Um só caminho ou múltiplos caminhos

A maior parte dos protocolos de encaminhamento determina um só caminho (o melhor entre vários possíveis) para atingir um determinado destino. No entanto existem alguns protocolos que prevêem a existência de vários caminhos simultaneamente utilizáveis para um mesmo destino, dividindo o tráfego por múltiplas ligações. As vantagens são óbvias: estes protocolos são mais fiáveis e podem conseguir um melhor desempenho. Podem, além disso, ser usados para implementar o balanceamento de tráfego na rede, sendo por isso uma ferramenta valiosa na área da Engenharia de Tráfego.

2.1.3.5 Planos ou hierárquicos

Num sistema de encaminhamento plano, qualquer encaminhador actua como par de todos os outros. Num sistema de encaminhamento hierárquico, alguns encaminhadores formam um *backbone* de encaminhamento, designado por domínio, sistema autónomo ou área. Dentro de um domínio existem encaminhadores que comunicam com encaminhadores de outros domínios enquanto que outros comunicam apenas com encaminhadores dentro do mesmo domínio. Forma-se assim uma hierarquia com diferentes níveis.

O encaminhamento hierárquico é normalmente utilizado para resolver os problemas de escalabilidade que outros tipos de encaminhamento colocam. Neste tipo de encaminhamento cada nó mantém apenas um estado (global) parcial, relativo ao domínio em que se encontra. Esta informação de estado é de alguma forma agregada antes de ser exportada para nós de outros domínios. No entanto, não devemos esquecer que quando a informação de estado da rede é agregada é introduzida alguma imprecisão que pode não ser menosprezável.

2.1.3.6 Interno ou Externos

É possível estruturar a Internet em diferentes níveis de uma hierarquia. No nível mais baixo encontram-se os sistemas terminais onde se ligam os diferentes utilizadores. Estes sistemas podem estar ligados em redes locais. Por sua vez diferentes redes locais podem estar ligadas entre si constituindo diferentes áreas de encaminhamento, constituindo por sua vez um sistemas autónomo. Um sistema autónomo é um conjunto de redes com uma estratégia de encaminhamento coerente sob um mesmo domínio administrativo. Para encaminhar o tráfego dentro de um sistema autónomo utiliza-se um protocolo de encaminhamento interno. Os diferentes sistemas autónomos estão por sua vez ligados entre si e constituem o nível mais alto da hierarquia. O encaminhamento de tráfego entre sistemas autónomos faz-se através de um protocolo de encaminhamento externo.

Os protocolos de encaminhamento externos diferem dos internos principalmente pelas métricas utilizadas. Apesar de serem também designadas por distâncias, não estão directamente relacionadas com a distância física ou com o número de saltos. O encaminhamento entre sistemas autónomos envolve principalmente questões económicas e políticas. A

atribuição de distâncias aos caminhos é baseada em decisões estratégicas locais, e consequentemente as decisões de encaminhamento tomadas por diferentes sistemas autónomos não têm que ser coerentes.

2.1.4 Protocolos de Encaminhamento Unicast usados na Internet

A Internet está dividida em sistemas autónomos, que não são mais do que domínios independentes de encaminhamento. Correspondem a maior parte da vezes a um fornecedor de serviço (ISP). Dentro do sistema autónomo são utilizados protocolos de encaminhamento internos (*intra-domain routing protocols*). Existem vários, e cada domínio pode optar por usar o protocolo interno que bem entender. Os protocolos de encaminhamento interno mais usados na Internet são o RIP[13], e o OSPF[23]. Por outro lado, para ligar os vários sistemas autónomos uns aos outros e disponibilizar conectividade global é necessário usar um protocolo de encaminhamento externo (*inter-domain routing protocol*). Actualmente o mais usado a este nível (senão mesmo o único), é o protocolo BGP (Border Gateway Protocol)[24].

2.1.4.1 RIP

O RIP (Routing Information Protocol)[13][25] [26] é baseado num algoritmo de vector de distância. A versão 1 deste protocolo está especificada na RFC 1058[13] e foi especialmente concebida para ser utilizada como um protocolo de encaminhamento interno, em redes de pequena dimensão e de topologia simples.

As mensagens de actualização contendo os "vectores de distância" são trocadas periodicamente pelos encaminhadores. Estas mensagens contêm pares que associam os endereço dos destinos à "distância" associada ao caminho. O próximo salto é o encaminhador que originou a mensagem.

Um encaminhador ao receber uma mensagem RIP de um vizinho compara esta informação com a informação que contém na sua tabela de encaminhamento para determinar futuros caminhos para os diferentes nós da rede. Relativamente aos endereços destino que já constam na tabela de encaminhamento, é necessário comparar as distâncias. Se a distância que consta na mensagem RIP for inferior, quer dizer que o nó que originou a mensagem RIP é o melhor próximo salto para atingir aquele destino. Nesse caso, a tabela de encaminhamento deve ser actualizada em conformidade. As entradas na tabela de encaminhamento são constituídas por tuplos que contêm o endereço do destino, a métrica associada ao caminho (distância) e o próximo salto. Depois de ter actualizado a tabela de encaminhamento em função de uma mensagem RIP, um encaminhador está então em condições de construir uma mensagem RIP com base na sua tabela de encaminhamento actualizada e enviá-la para todos os seus vizinhos. Os vizinhos repetirão este processo e assim sucessivamente, até que todos os nós do domínio de encaminhamento tenham actualizado as suas tabelas de encaminhamento.

Uma mensagem RIP tem um tamanho máximo de 512 *bytes* e pode conter no máximo 25 pares, o que permite que a mesma mensagem seja usada para actualizar mais do que uma entrada da tabela de encaminhamento. Por outro lado, em redes maiores, o pedido de actualização de uma tabela de encaminhamento completa pode implicar a transmissão de várias mensagens RIP. Por exemplo, se um encaminhador contiver na sua tabela de encaminhamento 100 entradas, a partilha desta informação com outros encaminhadores vai implicar a transmissão de, pelo menos, quatro mensagens RIP.

No protocolo RIP não existem preocupações com a sequenciação dos vários pacotes, ou seja, se um determinado nó destino receber o quarto pacote RIP (contendo as entradas numeradas da 76 até à 100), ele actualiza apenas a parte da tabela correspondente sem esperar pelos outros pacotes RIP. Isto faz com que os pacotes RIP possam ser transportados sem a sobrecarga de um protocolo de transporte como o TCP.

Como já foi referido, os protocolos de encaminhamento baseados em vectores de distância utilizam métricas para medir as distâncias que separam um encaminhador de todos os destinos conhecidos. É esta informação que permite que um encaminhador identifique qual o melhor próximo salto para chegar a um destino não adjacente. Na primeira versão do RIP (RFC 1058) só existe uma métrica: o número de saltos. Sempre que um encaminhador recebe e reenvia um pacote RIP a métrica é incrementada um valor. A tabela de encaminhamento RIP identifica o próximo salto para um pacote chegar ao seu destino com o menor número de saltos.

A primeira versão do RIP utiliza uma estratégia de actualizações tão simples que causa alguns problemas. O mais problemático é sem dúvida o tempo de convergência. Além disso quando há alterações na topologia de rede (provocadas, por exemplo por uma falha de conectividade numa ligação), pode ocorrer uma problema que é normalmente designado por problema da contagem até ao infinito (*counting to infinity problem*). Suponhamos que numa determinada topologia de rede, o sistema terminal H1 estava directamente ligado ao encaminhador R1 que por sua vez é vizinho do encaminhador R2. Numa primeira fase o encaminhador R1 anuncia ao encaminhador R2 uma rota para H1 com o custo 2. Supondo que R1 é a única forma de chegar a H1, o R2 não recebe mais nenhum anúncio para H1 e é mesmo essa rota (via R1) que insere na sua tabela de encaminhamento e que periodicamente passa a anunciar aos seus vizinhos. Se H1 falha, o encaminhador R1 quando se apercebe disso insere na sua tabela de encaminhamento uma rota para H1 (via H1) com o custo 16, uma vez que o valor 16 no RIP representa uma rota inatingível. Se R2 enviar para R1 um anuncio da rota para H1 (com custo 3) R1 vai actualizar a sua tabela de encaminhamento em conformidade, ou seja substituir a entrada para H1 via H1 com custo 16 por uma entrada para H1 via R2 com custo 3. E temos um ciclo no caminho para H1 (para atingir H1, R1 aponta para R2 e R2 para R1).

Esta situação acaba por se desfazer ao fim de algum tempo através dos anúncios periódicos. Ou seja, R1 depois de inserir na tabela de encaminhamento o caminho para H1 via R2 com custo 3, divulga-o para os vizinhos (inclusive para R2), aumentando um salto ao custo que passa a ser de 4. O R2 ao receber esta mensagem e apercebendo-se que ela foi originada por R1 o próximo salto que consta na sua tabela de encaminhamento para

atingir H1, actualiza a tabela de encaminhamento em conformidade, ou seja incrementa o custo do caminho. E passado algum tempo envia outra vez o anuncio da rota para R2, que faz o mesmo. Esta situação repete-se sucessivamente até as rotas atingirem o custo de 16, que como já foi referido tem o significado de rota inatingível. Só nessa altura é que se resolve o problema!

Esta e outras limitações da primeira versão do RIP (RIPv1[13]), foram corrigidas em versões mais recentes deste protocolo[25] [26].

2.1.4.2 OSPF

O OSPF (Open Shortest Path First)[23][10] é um protocolo de encaminhamento interno (ou seja, foi concebido para implementar o encaminhamento dentro de um sistema autónomo) e está especificado na RFC 2328[10]. É baseado num algoritmo de "estado de ligação", ou seja, cada nó divulga a informação de encaminhamento para todos os nós do seu domínio. No entanto, cada encaminhador divulga apenas a informação que descreve o estado das suas próprias ligações. Essa informação começa por ser divulgada de um nó para todos os seus vizinhos em forma de LSA (Link State Advertisement). Por sua vez os vizinhos reenviam o LSA recebido para todos os seus vizinhos e assim sucessivamente até o LSA ter sido recebido por todos os nós do domínio de encaminhamento.

Cada nó mantém uma base de dados com a topologia completa da rede bem como com o estado de todas as ligações. Esta base de dados é construída por cada um dos nós concatenando os vários LSAs recebidos.

Para calcular as melhores rotas para todos os destinos, cada nó executa o algoritmo Dijkstra sobre a base de dados topológica. Este algoritmo é um algoritmo muito simples que calcula os caminhos mais curtos para todos os destinos possíveis, de uma só vez. O algoritmo Dijkstra está descrito matematicamente na secção 2.1.1, onde também se apresenta um exemplo.

O OSPF foi desenvolvido no seio do IETF (Internet Engeneering Task Force) e é actualmente um standard de facto largamente usado ao nível do encaminhamento interno. Apresenta algumas vantagens em relação aos protocolos baseados em algoritmos de "vector de distância". Como cada nó possui um conhecimento completo da topologia da rede, a convergência do protocolo é muito mais rápida e não há o perigo de ciclos. O problema da contagem até ao infinito nunca se verifica, nem o problema dos ciclos temporários. Adicionalmente, torna-se fácil de utilizar outras métricas que não o custo ou o número de saltos, podendo mesmo combinar-se múltiplas métricas. Suporta também o encaminhamento com caminhos múltiplos de igual custo.

Como não poderia deixar de ser também apresenta alguns problemas. Guardar a informação sobre toda a topologia da rede e o estado de todas as ligações em cada um dos encaminhadores, assim como difundir para toda a rede os LSAs relativos a cada nó, limita

a dimensão do domínio de encaminhamento e traz alguns problemas de escala[3]. Este facto faz com que seja complicado usar um protocolo de "estado da ligação" entre domínios.

2.1.4.3 BGP

O protocolo BGP (Border Gateway Protocol)[24][27] [28] está especificado na RFC 1771[28] e é o protocolo de encaminhamento externo mais usado na Internet, senão mesmo o único.

Em cada sistema autónomo que faz parte da Internet existe um ou mais encaminhadores, designados por encaminhadores de fronteira (*border routers* na terminologia inglesa). Os encaminhadores de fronteira dos vários sistemas autónomos são os encaminhadores que executam o protocolo BGP. Um determinado encaminhador de fronteira tem pelo menos um par, que é configurado de forma administrativa pelo operador da rede. Um par BGP de um encaminhador de fronteira de um sistema autónomo é um encaminhador de fronteira de outro sistema autónomo com quem o encaminhador troca informação de encaminhamento.

Para iniciar a troca de informação de encaminhamento com um determinado par, um encaminhador de fronteira envia uma mensagem do tipo **BGP Open**, à qual o par responde com outra mensagem **BGP Open** se estiver interessado em trocar informação de encaminhamento com esse encaminhador de fronteira. Depois disso, os dois encaminhadores de fronteira trocam uma série de mensagem do tipo **BGP Update** que contém as tabelas de encaminhamento de um e outro. A partir daí a ligação entre os dois fica estabelecida, e é mantida através de mensagens do tipo **BGP KeepAlive**. As mensagens do tipo **BGP Update** só voltam a ser trocadas no caso de se verificar alguma alteração numa ou noutra tabela de encaminhamento.

As mensagens do tipo **BGP Update** são constituídas por uma lista de prefixos de endereços de rede e respectivos atributos. Os atributos correspondem ao que nos outros protocolos de encaminhamento se designa por métricas. Basicamente é o que permite ao protocolo seleccionar o melhor caminho de entre todos os possíveis.

O BGP difere dos protocolos de encaminhamento baseados em algoritmos de "vectores de distância" em dois aspectos fundamentais:

- Em primeiro lugar, e como já foi referido, em vez de enviar actualizações periódicas, um encaminhador BGP, depois de enviar pela primeira vez a informação constante da sua tabela de encaminhamento aos seus pares, só torna a enviar informação de encaminhamento no caso de se ter verificado alguma alteração. Até lá, a ligação com os diferentes pares é mantida através de mensagens de *keepalive*. A transmissão das actualizações, caso ocorram, é feita através do protocolo TCP para garantir a fiabilidade da transmissão[4]. Por outro lado como um encaminhador BGP só retrans-

[3] Estes problemas de escala são minimizados através da divisão em áreas

[4] No caso do RIP por exemplo, se alguma mensagem de actualização se perder não há problema de maior, porque ela será retransmitida dentro dos próximos 30 segundos

mite informação de encaminhamento ao seus pares no caso de se verificar alguma alteração, é necessário que os encaminhadores BGP armazenem todas as rotas alternativas, além da que é seleccionada como melhor rota, e que é colocada na tabela de encaminhamento. Assim, quando o caminho que consta na tabela de encaminhamento falha, o encaminhador é capaz de escolher uma rota alternativa para a substituir.

- Em segundo lugar, coloca-se a questão dos ciclos. Como o protocolo BGP é um protocolo de encaminhamento externo, as métricas que usa para seleccionar os melhores caminhos não dependem a maior parte das vezes do tamanho do caminho, mas sim de políticas locais. Por esse motivo tem que ser tomado um cuidado especial com o aparecimento de caminhos com ciclos, uma vez que as políticas adoptadas pelos diferentes sistemas autónomos não têm que ser coerentes. O BGP lida com esta questão de forma bastante simples. Quando anuncia um prefixo, o caminho completo para atingir esse prefixo também é divulgado. É um dos atributos do caminho divulgados. O caminho completo é representado pela sequência de sistemas autónomos que terá que ser percorrida para atingir o destino. Cada sistema autónomo é representado através do seu identificador universal, o AS# obrigatoriamente registado[29]. Um encaminhador evita os ciclos verificando se um determinado prefixo tem no caminho que lhe está associado o mesmo sistema autónomo a que ele pertence. Se tiver, esse caminho não é considerado para atingir o prefixo. Por este motivo o protocolo BGP é muitas vezes classificado como um protocolo de vectores de caminhos (*path vectors*) em vez de vectores de distancia (*distance vectors*).

2.2 Encaminhamento Multicast

A comunicação *multicast* é a capacidade de transportar o mesmo pacote de informação para mais do que um destinatário poupando ao máximo os recursos de rede. Para isso, os pacotes *multicast* são enviados uma única vez através da mesma ligação independentemente do número de destinatários que são alcançáveis através dessa ligação. Os mecanismos de encaminhamento *multicast* promoverão a duplicação de pacotes sempre que (e apenas quando) a localização topológica dos participantes a isso obrigue.

As soluções propostas nesta área baseiam-se na construção de árvores a partir de grafos. Existem duas razões para este facto:

- numa topologia tipo árvore os dados podem ser transmitidos simultaneamente ao longo dos seus ramos; e
- pode ser transmitido apenas um número mínimo de cópias, sendo a duplicação feita apenas quando for necessário, ou seja, quando um ramo se subdivide.

À semelhança do modelo *unicast* também aqui cada ligação tem um custo associado e

o custo da árvore é a soma dos custos de todas as ligações incluídas na árvore. Minimizar o custo total da árvore reflecte-se na melhoria da utilização dos recursos de rede disponíveis.

O problema de encontrar a árvore de custo mínimo é um problema NP-completo, muitas vezes referido na literatura como o problema da árvore de Steiner (*Steiner tree problem*, na terminologia inglesa).

2.2.1 Definições Matemáticas

Da teoria dos grafos vamos usar as seguintes definições:

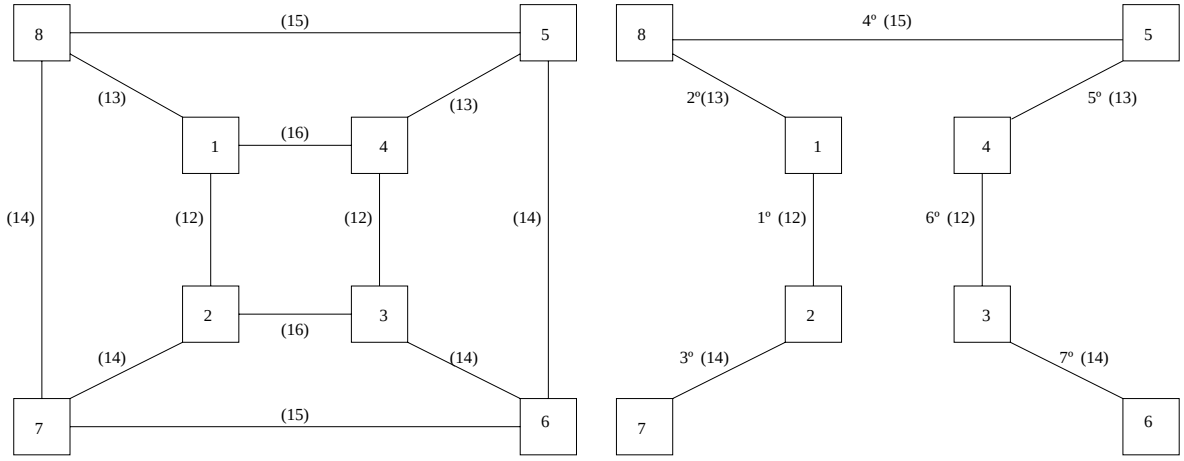
- Um grafo G é uma árvore se, e só se, existe um único caminho possível entre dois quaisquer dos seus vértices.
- Se $G = (V, A)$ é um grafo e $T = (V, B)$ é um subgrafo de G que também é uma árvore, então T é uma *spanning tree* no grafo G. O conjunto B é neste caso designado pelo conjunto de ramos que constitui a árvore T.
- Se w for a função que associa a cada aresta a do grafo G um peso $w(a)$, então se H for um subgrafo de G, $w(H)$ é a soma dos pesos de todas as arestas que constituem o subgrafo H, e designa-se por custo total de H.
- Uma *spanning tree* T de um grafo G é uma *spanning tree* de custo mínimo (*Minimal Spanning Tree*-MST) se não existir outra *spanning tree* de G com um custo total menor do que $w(T)$.

Existem vários algoritmos que conseguem facilmente determinar a árvore de custo mínimo de um grafo. A título de exemplo pode referir-se o algoritmo proposto por Prim[30]. Segundo este algoritmo, a árvore de custo mínimo de um grafo pode ser construída da seguinte forma: começa-se por escolher arbitrariamente um vértice qualquer de G, vértice esse que é inserido na *spanning tree* T, ainda vazia. Depois disso e em cada iteração do algoritmo é escolhida a aresta de custo mínimo que une um dos vértices da árvore já construída a um qualquer vértice de G que ainda não faz parte da árvore, aresta essa que é adicionada à *spanning tree*. O algoritmo termina quando todos os vértices de G já constam da *spanning tree* T, que é então uma *spanning tree* de custo mínimo. É apresentado um exemplo da aplicação deste algoritmo na figura 2.3.

Além do algoritmo de Prim existem outros que conseguem encontrar a *spanning tree* de custo mínimo de um grafo G. O mesmo já não se verifica se se pretender resolver o problema da árvore de Steiner. Este problema é uma variante do problema da *spanning tree* de custo mínimo.

Dado um subconjunto S de V (sendo V o conjunto dos vértices de um grafo $G = (V, A)$, o que se pretende determinar é um subgrafo $T = (S, B)$ de G tal que

- T é uma árvore;



(a) *Topologia Inicial* (b) *Minimal Spanning Tree Obtida*

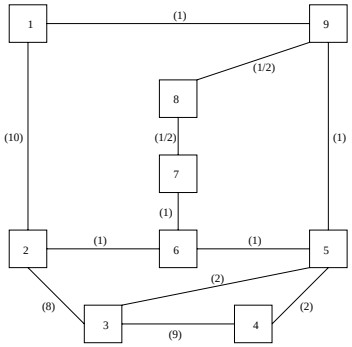
Figura 2.3: Exemplo da aplicação do Algoritmo de Prim

- $w(T)$ é mínimo;
- B contem todos os vértices de S e é um subconjunto de A.

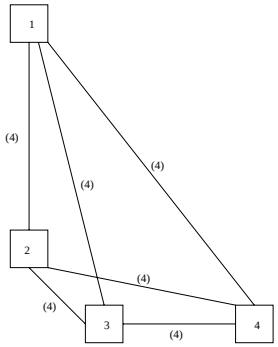
Ao contrário do problema da *spanning tree* de custo mínimo, o problema da árvore de Steiner, é um problema NP-Completo.

Existem algumas heurísticas que propõem soluções aproximadas para o problema da árvore de Steiner, entre as quais referimos a título de exemplo a heurística KMB[31] (sigla baseada no nome dos seus autores: Kou, Markowsky e Berman).

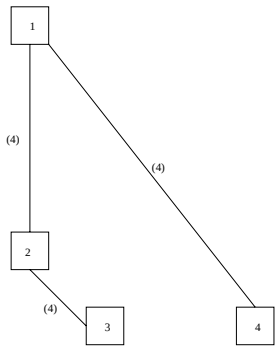
A heurística KMB começa por construir um grafo completo de distâncias a partir do grafo original. Este grafo contem todos os vértices que correspondem ao grupo *multicast* e que é um subconjunto do conjunto de vértices do grafo que representa a topologia de rede completa. Cada par de nós deste subgrafo está ligado por uma aresta que corresponde ao caminho mais curto que os liga no grafo original e tem o custo desse caminho (que corresponde à distância entre esses dois vértices). Depois é utilizado o algoritmo de Prim para encontrar uma *spanning tree* de custo mínimo a partir do grafo completo de distâncias. Uma vez construída a MST, todos os ramos são substituídos pelos caminhos a que correspondem. O grafo obtido pode conter ciclos. Para os retirar é de novo calculada usando o mesmo algoritmo de Prim uma *spanning tree* de custo mínimo, agora a partir do último grafo obtido. Finalmente são removidos todos os nós folha que não correspondem a membros do grupo *multicast*. Na figura 2.4 é apresentado um exemplo da aplicação desta heurística a uma topologia com nove nós, em que apenas quatro (os nós 1, 2, 3 e 4) são membros do grupo *multicast*.



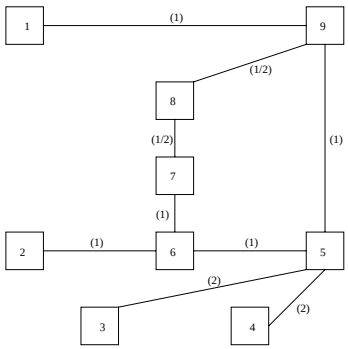
(a) *Topologia Inicial*



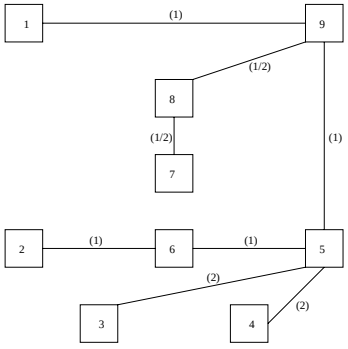
(b) *Grafo Completo de Distâncias*



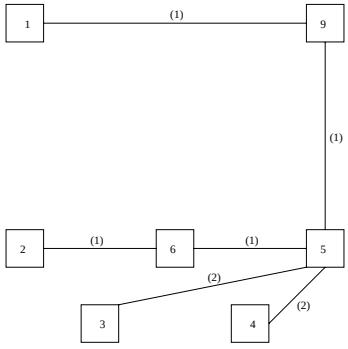
(c) *MST obtida a partir do grafo completo de distâncias*



(d) *Grafo obtido depois da reposição dos ramos originais*



(e) *MST obtida a partir do grafo anterior*



(f) *Árvore Multicast obtida*

Figura 2.4: Exemplo da Aplicação da Heurística KMB à construção da árvore multicast

Esta heurística não conduz à solução óptima, no entanto está provado que o custo da árvore obtida usando a heurística KMB é sempre menor do que o dobro do custo da árvore de Steiner.

Na comunicação *multicast*, problema da árvore de Steiner é agravado pelo facto de muitas vezes estarmos perante grupos *multicast* dinâmicos. Como se sabe, na maior parte das aplicações *multicast* a constituição de um grupo é alterada frequentemente, ou seja, tornam-se necessários algoritmos de encaminhamento *multicast* dinâmicos que consigam suportar convenientemente essas alterações.

O problema do encaminhamento *multicast* pode definir-se da seguinte forma:

- Seja $G = (N, L)$ um grafo que representa uma rede com N nós e L ligações.
- Seja c_{ij} o custo da ligação entre os nós i e j.
- Seja M um grupo *multicast* (incluindo fontes) que está contido no conjunto de nós N.
- Encontrar uma ou mais árvores $T = (N', R)$, subgrafos de G, onde N' é um subconjunto de N e contém todos os nós de M.

2.2.2 Algoritmos de Encaminhamento Multicast

Os algoritmos de encaminhamento *multicast* usam diferentes estratégias para resolver o problema do encaminhamento *multicast*. Enquanto alguns procuram encontrar uma única árvore que liga todas as fontes aos destinatários, outros procuram estabelecer uma árvore por cada fonte activa.

As diferentes estratégias usadas apresentam vantagens e desvantagens umas em relação às outras. Nesta secção descrevem-se as principais estratégias usadas pelos algoritmos de encaminhamento na construção de árvores de distribuição de tráfego *multicast*, realçando as suas principais diferenças.

2.2.2.1 Flooding

De acordo com esta estratégia um encaminhador ao receber um pacote *multicast*, tem apenas que verificar se é a primeira vez que recebe o pacote. Se for esse o caso, o encaminhador limita-se a re-enviar o pacote para todas as interfaces de saída com uma única excepção a interface por onde recebeu o pacote. No caso de já ter recebido esse pacote, o pacote é simplesmente descartado.

Esta estratégia é bastante simples. A única dificuldade consiste em determinar se é a primeira vez que se recebe determinado pacote. Existem diferentes soluções para este problema: uma hipótese seria armazenar todos os pacotes *multicast* que chegam a

determinado encaminhador; outra solução consiste em transportar no pacote a lista de encaminhadores pelos quais o pacote já passou.

Apesar de serem muito simples e robustos, os algoritmos baseados neste tipo de estratégia consomem muitos recursos de rede e memória, e apresentam por isso sérios problemas de escala.

Existem algumas variantes desta estratégia que procuram melhorá-la e minimizar as suas desvantagens.

O algoritmo RPF (Reverse Path Forwarding)[32], por exemplo, propõe que um encaminhador, ao receber um pacote *multicast* e antes de o enviar para todos as interfaces de saída verifique se a interface por onde chegou o pacote é a interface usada pelo encaminhador para atingir o nó que originou o pacote (fonte) através do melhor caminho. Se for, o pacote é então re-enviado para todas as interfaces de saída do encaminhador com excepção da interface por onde chegou o pacote. Se não, o pacote é descartado. Claro que para fazer este tipo de verificação o algoritmo tem de consultar as tabelas de encaminhamento *unicast*.

Apesar de melhorar significativamente a estratégia de *flooding* indiscriminado, o algoritmo RPF continua a fazer chegar o tráfego *multicast* a todos o nós da rede, independentemente de estes terem potenciais destinatários ligados ou não, Para resolver este problema existe um outro algoritmo que é designado por algoritmo de *flood and prune*, e que permite que os ramos da árvore que não conduzam a nenhum destinatário activo sejam eliminados. Este algoritmo desenrola-se em duas fases. Numa primeira fase todos os nós recebem os pacotes *multicast* e a única verificação que é feita, é implementada pelo algoritmo RPF. Quando um encaminhador "folha" recebe um pacote *multicast* e sabe que não tem destinatários para esse grupo, envia para cima uma mensagem de *prune*. Esta é a segunda fase do algoritmo. Gradualmente, os encaminhadores são informados que existem algumas sub-árvores que não conduzem a qualquer destinatário, e como tal deixam de re-enviar pacotes *multicast* através das interfaces que conduzem a essa sub-árvores.

Esta variante do algoritmo RPF limita a distribuição de tráfego *multicast* aos ramos efectivamente úteis. Por outro lado, a primeira fase (designada por *flood*) tem que ser periódica uma vez que é necessário descobrir novos destinatários. Este algoritmo também obriga a que os encaminhadores guardem informação de estado por cada grupo *multicast*.

Qualquer umas das estratégias descritas nesta secção constrói árvores diferentes por fonte activa, o que traz algumas vantagens, nomeadamente quando à utilização dos recursos de rede, que fica assim mais distribuída. Apesar de construir uma árvore por cada fonte activa, como a estratégia usada é baseada no algoritmo RPF, as árvores não são construídas com base nos caminhos mais curtos deste a fonte até cada um dos destinatários, mas sim no sentido inverso. Ou seja, é uma estratégia adequada apenas à utilização em redes simétricas, ou seja, em redes onde o custo das ligações é igual em ambos os sentidos.

2.2.2.2 Árvores Partilhadas

Uma alternativa aos algoritmos que estabelecem árvores centradas nas fontes são aqueles que estabelecem uma única árvore partilhada por todas as fontes e todos os destinatários de um grupo.

Esta estratégia pressupõe a existência de um nó pré-definido, muitas vezes designado por *core router* ou *rendez-vous point* (RP), a partir do qual se estabelece a árvore partilhada. Quando um novo membro decide juntar-se, envia um pedido explícito dirigido a esse nó pré-definido, e todos os encaminhadores no caminho, ao receberem esse pedido, actualizam as suas tabelas de encaminhamento *multicast* no sentido de construírem um novo ramo que liga o novo membro à árvore de distribuição *multicast*. O pedido de *join* tipicamente é re-enviado até encontrar o primeiro nó que já pertence à árvore, pois é a partir deste nó que é estabelecido o novo ramo da árvore.

Esta estratégia apresenta inúmeras vantagens em relação às estratégias baseadas no *flooding* sendo a mais importante o facto de poupar memória e recursos de rede. Por outro lado o tráfego é re-enviado sempre através das mesmas ligações, independentemente da fonte que o gera, o que pode conduzir a uma excessiva sobrecarga e até situações de congestão nestas ligações.

Também no caso destes algoritmos, as árvores são construídas com base no pressuposto de que as redes são simétricas, ou seja que as ligações tem o mesmo custo em ambos os sentidos. Como foi referido os novos caminhos são construídos com base nos melhores caminhos desde os novos membros até à raíz da árvore, ou seja, precisamente no sentido oposto ao sentido utilizado pelo tráfego.

2.2.2.3 Árvores de Caminhos Mais Curtos

De acordo com esta estratégia as árvores têm obrigatoriamente que ser construídas no mesmo sentido que é usado pelo tráfego. Para isso, os encaminhadores têm que possuir um conhecimento completo da topologia da rede e têm que saber qual a constituição dos grupos, para assim poderem estabelecer os melhores caminhos desde cada uma das fontes até aos diferentes destinatários. Estes caminhos são estabelecidos com base na tabelas de encaminhamento *unicast*.

Convém aqui referir, que as árvores de Caminhos Mais Curtos minimizam o custo dos caminhos entre a raíz da árvore (fonte), e cada uma das folhas (destinatários), não querendo no entanto isso dizer que resolve o problema da árvore de Steiner. De acordo com o problema da árvore de Steiner o que se pretende minimizar é o custo total da árvore, e não o custo individual de cada um dos seus ramos.

2.2.3 Modelo de Serviço do Multicast IP

De acordo com o modelo de serviço usado no Multicast IP um grupo *multicast* é um grupo totalmente aberto. Qualquer sistema terminal pode enviar ou receber tráfego *multicast* do grupo, sem qualquer pedido de autorização prévia. Ou seja, os grupos *multicast* são muito dinâmicos: ao longo do seu tempo de vida a sua constituição é alterada diversas vezes.

Um mesmo sistema terminal pode fazer parte de diferentes grupos *multicast*, sem qualquer restrição. Uma fonte pode enviar tráfego para um grupo *multicast* pertencendo ou não ao grupo.

A constituição do grupo não tem que ser conhecida previamente, nem pelas fontes, nem pelos diferentes destinatários.

Cada grupo *multicast* é identificado através de um endereço IP, designado por endereço *multicast*. No caso do IPv4 um endereço *multicast* é um endereço da classe D que está compreendido no intervalo de 224.0.0.0 até 239.255.255.255. Ao contrário do que se passa com os endereços *unicast*, os endereços *multicast* não são geralmente atribuídos estaticamente por uma qualquer autoridade. Em vez disso são escolhidos dinamicamente pelas fontes.

2.2.4 Protocolos de Encaminhamento Multicast IP

Existem vários protocolos de encaminhamento *multicast* que podem ser usados para construir e manter árvores para grupos *multicast*. Alguns dos mais conhecidos são o PIM-DM (Protocol Independent Multicast - Dense Mode)[33], o DVMRP (Distance-Vector Multicast Routing Protocol)[34], o MOSPF(Multicast Open Shortest Path First)[35], o PIM-SM (Protocol Independent Multicast - Sparse Mode)[7] e o CBT (Core Based Trees)[36]. Tanto o PIM-DM como o DVMRP utilizam o algoritmo de *flood and prune* e constroem uma árvore diferente por cada fonte activa, ao passo que o PIM-SM e o CBT constroem uma única árvore que é partilhada por fontes e destinatários. O MOSPF também constrói uma árvore por fonte activa mas baseia o seu funcionamento no algoritmo SPT (Shortest Path Tree), ao contrário do DVRMP e do PIM-DM que utilizam o algoritmo RPF (Reverse Path Forwarding). Em termos de requisitos para o funcionamento, o MOSPF requer a utilização do protocolo *unicast* OSPF e o DVMRP um protocolo *unicast* interno tipo RIP. As duas versões do PIM (PIM-DM e PIM-SM), como indica a sua designação (*Protocol Independent*), não têm requisitos especiais em termos do protocolo de encaminhamento *unicast*.

2.2.4.1 DVRMP

O DVRMP (Distance Vector Routing Multicast Protocol) está definido na RFC 1075[34] e constituiu a base do MBONE [37] (IP Multicast Backbone on the Internet). Utiliza a técnica de *flooding* descrita na secção 2.2.2.1. Apesar de melhorado com a utilização do

algoritmo RPF apresenta ainda problemas de escala que se fazem notar principalmente em versões que não implementam o *prune* de ramos que não conduzem a membros activos.

Para implementar o algoritmo RPF, o DVRMP inclui o seu próprio protocolo de encaminhamento *unicast*. O protocolo de encaminhamento *unicast* do DVRMP é um protocolo de vector de distância muito semelhante ao RIP, já descrito na secção 2.1.4.1. Cada entrada no vector contem não a distância para um determinado destino, mas sim a distância para determinada fonte. À semelhança do que acontece no RIP estas distâncias são medidas em número de saltos. É através da consulta destes vectores em cada um do nós que o algoritmo RPF determina se deve ou não re-enviar determinado pacote *multicast*: só re-envia se o tiver recebido pela interface que está no caminho mais curto para atingir a fonte, senão o pacote é descartado.

Se o pacote é para ser re-enviado, o algoritmo RPF obriga o encaminhador a enviar uma cópia do pacote para todos os caminhos até aos diferentes destinos excepto no caminho de volta para a origem. Os encaminhadores ligados a LANs que não queiram receber tráfego de um grupo de difusão particular, devem enviar uma mensagem de *prune* de volta para a origem para impedir o fluxo desnecessário de dados naquele percurso. Só as versões mais recentes do protocolo DVRMP é que incluem esta última funcionalidade.

2.2.4.2 MOSPF

O MOSPF (Multicast Open Shortest Path First)[38] é uma extensão ao protocolo de encaminhamento *unicast* OSPF, descrito na secção 2.1.4.2 deste capítulo.

De acordo com o protocolo OSPF, cada encaminhador envia periodicamente LSAs para todos os outros encaminhadores da rede, permitindo a cada encaminhador deter um conhecimento completo da topologia e do estado das diferentes ligações. A partir desta informação qualquer um dos encaminhadores está em condições de estabelecer os caminhos mais curtos para todos os destinos possíveis, utilizando para isso, o algoritmo de Dijkstra.

O MOSPF extende os LSAs para transportarem também informação acerca da constituição dos grupos *multicast*, ou seja cada encaminhador com receptores interessados em fazer parte de determinado grupo *multicast*, divulga essa informação em forma de LSAs. Desta forma, todos os encaminhadores ficam a saber a que encaminhadores estão ligados os membros dos diferentes grupos *multicast*.

Juntando esta informação com a informação sobre a topologia de rede, cada encaminhador pode, desde que tenha uma fonte interessada, construir uma árvore *multicast* centrada nele próprio, em que os ramos coincidem precisamente com os caminhos mais curtos para cada um dos receptores do grupo.

Os problemas de escala sentidos pelos algoritmos baseados na estratégia de *flood and prune* (por exemplo o DVRMP), não são ultrapassados pelo MOSPF. Este protocolo apesar de não implicar a difusão de mensagens de controle exige, no entanto, que todos os encaminhadores possuam uma base de dados com a constituição de todos os grupo

multicast, além da base de dados topológica.

2.2.4.3 PIM

O PIM (Protocol Independent Multicast) inclui dois modos distintos de operação: o modo denso (DM - *Dense Mode*) e o modo esparso (SM - *Sparse Mode*), que se adequam melhor respectivamente a grupos de difusão densos e a grupos de difusão esparsos.

O modo-denso do PIM tem um modo de funcionamento muito semelhante ao protocolo DVMRP, já descrito na secção 2.2.4.1 deste capítulo, só se distinguindo dele pelo facto de não usar nenhum protocolo de encaminhamento específico. Ou seja, o PIM-DM usa qualquer protocolo de encaminhamento *unicast* que esteja disponível, enquanto o DVMRP utiliza o seu próprio protocolo de encaminhamento *unicast*.

O modo esparso está documentado na RFC 2362[7]. Foi concebido para ser utilizado em redes alargadas, tendo por isso como uma das suas principais preocupações, a questão da escala. O PIM-SM prevê a construção de dois tipos de árvores: uma árvores partilhada e árvores centradas nas fontes.

Começa por construir a árvore partilhada em torno de um ponto pré-definido, designado por ponto de encontro (RP - *Rendez-Vous Point* na terminologia inglesa). Quando um sistema pretende enviar dados, envia-os por *unicast* para o ponto de encontro. Por outro lado, quando um sistema pretende receber dados, o encaminhador mais próximo (designado por *designated router*) deve registar-se junto do ponto de encontro para os poder receber. O fluxo de dados flui assim da origem para o ponto de encontro e deste para os diferentes destinos. Os encaminhadores incluídos no trajecto optimizam o percurso e removem automaticamente pontos intermédios desnecessários.

Numa segunda fase, o protocolo PIM-SM, prevê que os receptores possam iniciar a construção de árvores centradas nas fontes, podendo assim usufruir de uma melhor árvore do que a árvore partilhada.

O protocolo PIM-SM está extraordinariamente divulgado e foi usado como protocolo de referência e comparação com os diferentes protocolo de encaminhamento *multicast* desenvolvidos neste trabalho. Por esse motivo, foi incluída uma descrição mais detalhada do seu funcionamento no capítulo 4.

2.2.4.4 CBT

O CBT (Core Based Trees)[36] é um protocolo de encaminhamento *multicast* que baseia o seu funcionamento apenas numa única árvore partilhada (por grupo). Fontes e receptores estão interligados através de uma árvore bidireccional centrada num ponto pré-definido que aqui se designa por *core router*. Quando um novo membro pretende juntar-se a um grupo *multicast* envia uma mensagem de *join* em direcção ao *core router*. É esta mensagem que ao longo do seu caminho é responsável pela criação de informação de estado nos

diferentes encaminhadores, informação essa que constitui o novo ramo da árvore *multicast*. A mensagem de *join* é re-enviada pelos encaminhadores até que um nó que já faz parte da árvore seja encontrado. Um encaminhador na árvore *multicast*, ao receber um pacote proveniente de uma fonte, re-envia-o para todas as interfaces que constam da respectiva entrada na tabela de encaminhamento multicast, excepto para a interface por onde recebeu o pacote.

Neste protocolo a questão da localização topológica do *core router* é fundamental para o seu desempenho. Por outro lado, existe uma grande concentração de recursos o que pode tornar-se uma desvantagem. Como principal vantagem em relação aos outros protocolos de encaminhamento *multicast* apresenta o facto de necessitar de uma quantidade bastante reduzida de informação de estado (os encaminhadores têm apenas que armazenar informação de estado por grupo e não por cada par fonte-grupo).

2.3 Qualidade de Serviço em Redes IP

O facto da Internet actual basear o seu nível de serviço no modelo do "melhor esforço" (*best-effort*) tem permitido manter uma grande simplicidade ao nível da rede. Este facto teve como consequência transformar a Internet numa infra-estrutura escalável o que facilitou o seu enorme crescimento nos últimos anos. No entanto, o facto de cada vez mais utilizadores e respectivos sistemas terminais se irem ligando à rede, faz com que por vezes o tráfego gerado pelas diferentes aplicações ultrapasse a capacidade da infra-estrutura instalada. Na Internet, tal como a conhecemos actualmente, o serviço nunca é negado, o que acontece é que a qualidade do serviço prestado vai diminuindo gradualmente. Ou seja, aumentam os atrasos, os pacotes perdidos, mas a rede continua a funcionar. Para certo tipo de aplicações, de que são exemplo o Correio Electrónico, a Transferência de Ficheiros, etc, essa diminuição do nível de serviço prestado pode passar despercebida ou ser tolerável. No entanto há outro tipo de aplicações, como as aplicações multimédia e as aplicações em tempo real para as quais essa arbitrariedade não é aceitável. Esse tipo de aplicações pressupõe que a qualidade e a quantidade de recursos disponíveis sejam de alguma forma determinísticos. Para que isso aconteça na Internet torna-se necessário dotar a Internet de mecanismos adicionais que permitam distinguir diferentes tipos de tráfego: tráfego com e sem requisitos. Para designar essa nova capacidade utiliza-se o termo Qualidade de Serviço (Quality of Service - QoS) que não pretende criar mais largura de banda nem outros recursos de rede, limitando-se a geri-los de forma eficiente para ir de encontro aos diferentes requisitos das aplicações. O objectivo da Qualidade de Serviço é disponibilizar algum grau de previsibilidade e controle para além do actual serviço *best-effort*.

No âmbito do IETF têm sido propostas diferentes soluções para implementar QoS na Internet. Nas próximas secções serão descritas as propostas que a comunidade considera mais importantes.

2.3.1 Modelo de Serviços Integrados

O modelo de serviços integrados, IntServ[39], propõe duas novas classes de serviço além da classe do serviço de "melhor esforço". São elas:

- A classe de Serviço Garantido (Garanteed Quality of Service)[40] para aplicações que necessitam de um limite fixo para o atraso.
- A classe de serviço de Carga Controlada (Controlled Load[41]), para aplicações que necessitam de um serviço do tipo *best-effort* melhorado e de confiança.

Este modelo pressupõe que os encaminhadores são capazes de efectuar uma reserva de recursos com o objectivo de fornecer qualidade de serviço a fluxos de tráfego específicos. Para isso têm que manter informação de estado por fluxo.

O RSVP (Resource Reservation Protocol)[42] é um protocolo de sinalização que normalmente possibilita que as aplicações efectuem reserva de recursos. O processo de sinalização é efectuado em duas fases:

- Na primeira fase a fonte especifica as características do tráfego que vai ser gerado. Essa caracterização é efectuada em termos dos limites superior e inferior de largura de banda, atraso e *jitter* (variações no atraso) e designa-se por especificação de tráfego (TSpec - Traffic Specification). O protocolo RSVP envia a TSpec da fonte para o destinatário numa mensagem que se designa por **mensagem PATH**. Um encaminhador ao receber uma mensagem deste tipo mais não faz do que reencaminhá-la pelo caminho "mais curto" até ao destino, mantendo numa tabela a informação de estado relativo ao caminho seguido, pela **mensagem PATH**, que terá que incluir o endereço do encaminhador de onde a mensagem veio. Esta informação é absolutamente essencial para que durante a segunda fase, a resposta à **mensagem PATH** consiga efectuar exactamente o mesmo caminho que a **mensagem PATH**, no sentido contrário.
- A segunda fase tem início quando o destinatário, ao receber a **mensagem PATH**, gera uma **mensagem RESV** que, percorrendo o caminho inverso, vai reservar os recursos necessários para satisfazer os requisitos de QoS daquele fluxo. Para isso, o destinatário acrescenta à TSpec contida na **mensagem PATH** uma especificação de requisitos (RSpec - Request Specification) que indica o tipo de classe (Serviço Garantido ou Carga Controlada) da reserva que se pretende fazer. E uma outra especificação designada por *filter spec* que caracteriza os pacotes para os quais a reserva está a ser feita (por exemplo o protocolo e o número da porta). Estas duas especificações juntas constituem o descritor de fluxo que é usado pelos encaminhadores para identificar as várias reservas.

  Essa mensagem vai percorrer exactamente o caminho percorrido pela **mensagem PATH**, no sentido contrário. Um encaminhador ao receber uma **mensagem RESV**

pode aceitá-la ou não. Se recusar é enviada uma mensagem de erro ao destinatário e o processo de reserva de recursos fica por aí. Se aceitar tem que alocar os recursos solicitados (largura de banda e espaço de armazenamento), e acrescentar no encaminhador a informação de estado respectiva.

Além do RSVP, o modelo de serviços integrados incluí ainda uma componente que faz controle de admissão, um classificador, e um escalonador de pacotes. As rotinas de controle de admissão são responsáveis por decidir quando é que um pedido de recursos pode ou não ser satisfeito. As outras duas componentes (o classificador e o escalonador de pacotes) entram em jogo já na fase de transmissão de dados. Ou seja, quando um pacote chega a um encaminhador, o classificador efectua a classificação do pacote com base nos vários campos do cabeçalho (endereço e porta de origem, endereço e porta de destino, protocolo, etc). Essa classificação serve para identificar qual a reserva a que pertence o pacote. É com base no resultado obtido, que o pacote é colocado numa determinada fila de espera. O escalonador há-de depois tratar o pacote de forma a cumprir os requisitos de qualidade de serviço requeridos pelo fluxo de tráfego respectivo.

O modelo de Serviços Integrados apresenta alguns problemas que dificultam a sua implementação e utilização em larga escala, nomeadamente:

- A informação de estado que é necessário manter em cada encaminhador, aumenta proporcionalmente com o número de fluxos, o que traz a este modelo problemas de escala.
- Os requisitos exigidos aos encaminhadores são grandes. Todos eles têm que suportar o protocolo RSVP, controle de admissão, um classificador e um escalonador de pacotes.

2.3.2 Modelo de Serviços Diferenciados

O modelo de Serviços Diferenciados, DiffServ[3], surgiu como uma alternativa ao modelo de serviços integrados devido, fundamentalmente, aos problemas de escala que este apresenta.

No modelo de serviços diferenciados, o campo TOS (Type of Service)[43][44] dos pacotes IP é aproveitado para que as aplicações possam indicar os seus requisitos de qualidade de serviço. O campo TOS passa a ser designado por Campo DS (Differenteated Services) e fornece aos encaminhadores informação caracterizadora do tráfego transportado que condicionará o tipo de tratamento que lhes deverá merecer cada pacote. Assim, além do formato do campo DS, o modelo de serviços diferenciados define também um conjunto de tratamentos a dar aos pacotes quando chegam a um encaminhador (PHB - Per Hop Behaviors[4]). É marcando os pacotes através do campo DS e dando-lhes um tratamento específico de acordo com a marcação feita, que este modelo consegue criar diferentes classes de serviço.

Um determinado utilizador para ter acesso a um tratamento diferenciado deverá estabelecer um contrato de serviço com o seu fornecedor de serviço Internet (SLA - Service Level Agreement). Um SLA especifica as classes de serviço permitidas e a quantidade de tráfego que o utilizador pode enviar em cada classe de serviço. Existem dois tipos de SLAs distintos: os SLAs estáticos e os SLAs dinâmicos. Os SLAs estáticos são negociados periodicamente (mensalmente ou anualmente, dependendo), e os dinâmicos recorrem a um protocolo de sinalização (por exemplo o RSVP), para requisitarem os recursos a pedido.

Os campos DS dos pacotes são marcados pelos próprios utilizadores ao executarem determinadas aplicações, ou então pelo encaminhador de ingresso num domínio DiffServ. Neste caso, os encaminhadores classificam o pacote à entrada com base nalguns campos do próprio pacote (endereço fonte, endereço destino, porta, etc). O campo DS do pacote é depois marcado com base no resultado obtido por esta classificação.

Além de classificar os pacotes, os encaminhadores de ingresso têm que ser capazes de os policiar e também limitar. A classificação, policiamento e limitação dos pacotes dependem dos SLAs estabelecidos. Além dos SLAs estabelecidos entre fornecedores de serviço e utilizadores também existem SLAs entre domínios DS. Quando um pacote sai de um domínio e entra noutro pode ter que ser remarcado com base no SLA estabelecido entre os dois domínios.

O modelo dos serviços diferenciados, consegue resolver grande parte das insuficiências do modelo dos serviços integrados. Por um lado, apesar de permitir um número maior de classes de serviço, como os recursos são alocados por classe, a informação de estado é proporcional ao número de classes e não ao número de fluxos. Consegue-se assim resolver os problemas de escala presentes no modelo de serviços integrados.

Por outro lado as operações mais pesadas de classificação por campos múltiplos, marcação, policiamento e limitação só têm que estar presentes nos encaminhadores de fronteira. Os encaminhadores no interior de um domínio DS só precisam de determinar qual a classe a que pertence determinado pacote. Ou seja, num encaminhador de fronteira quando o pacote chega é classificado de acordo com alguns campos do seu cabeçalho e é marcado o respectivo campo DS com um determinado comportamento agregado (BA - Behavior Aggregate). É este BA que vai indicar aos encaminhadores qual a classe de serviço a que pertence aquele pacote e consequentemente qual o tratamento que deve ter. Os encaminhadores no interior de um domínio DS necessitam apenas de fazer uma classificação do pacote muito simples que lhes vai permitir determinar apenas qual o BA do pacote para que o possam colocar na fila de espera respectiva.

2.3.3 Encaminhamento com QoS

O encaminhamento com QoS é utilizado para descobrir rotas com múltiplas restrições, ou seja, a partir dos requisitos de serviço de determinado fluxo, ou de uma agregação de fluxos, é calculada uma rota capaz de satisfazer esse requisitos. Para determinar essa rota, os mecanismos de encaminhamento têm de considerar não só a topologia da rede,

mas também os recursos disponíveis em cada uma das ligações. A rota determinada pode não ser a mais curta, mas sim um caminho que esteja mais livre e que por isso consiga satisfazer os requisitos de qualidade de serviço em questão. Para conseguirem implementar este mecanismo de encaminhamento os encaminhadores necessitam de divulgar informação do estado das ligações e determinar rotas com base nessa informação.

O capítulo 3 é dedicado ao Encaminhamento com QoS. Aí é descrito algum trabalho relacionado tanto ao nível do encaminhamento *unicast* como ao nível do encaminhamento *multicast*.

Capítulo 3

Encaminhamento com Qualidade de Serviço

Os protocolos de encaminhamento tradicionais seleccionam o caminho de custo mínimo desde uma fonte até ao destino, sendo o custo uma função de um número bastante limitado de parâmetros muitas vezes tão simples quanto o número de saltos (*number of hops* na terminologia inglesa). Este modelo adequa-se ao paradigma do "melhor esforço"(*best-effort* na terminologia inglesa), mas é desadequado quando se pretende dotar a rede da capacidade de fornecer a Qualidade de Serviço (QoS) requerida por muitas aplicações. Nesse caso pode não interessar enviar todo o tráfego destinado a determinado nó através do mesmo caminho: apesar de eventualmente ser o mais curto pode não satisfazer os requisitos de QoS, e existirem caminhos alternativos que por sua vez os satisfazem.

O encaminhamento com QoS tem como principal objectivo influenciar as decisões de encaminhamento para que sejam levados em consideração os diferentes requisitos de QoS do tráfego. Existem diferentes formas de implementar o encaminhamento com QoS, nomeadamente:

- escolhendo os caminhos por fluxo, mediante os seus requisitos de QoS e efectuando reserva de recursos;
- instalando diferentes caminhos para diferentes classes de serviço;
- procedendo ao balanceamento da distribuição de tráfego na rede

Estas diferentes alternativas têm vantagens e desvantagens umas em relação às outras e não há, até ao momento, uma solução genérica amplamente aceite para implementar o encaminhamento com QoS.

A primeira alternativa anteriormente referida, encaminhamento por fluxo, é sem dúvida a que tem sido mais explorada. Neste caso quando uma fonte pretende começar a enviar dados com determinados requisitos de QoS para um determinado receptor, efectua

um pedido onde especifica não só o destinatário da informação mas também os requisitos de QoS. O sistema começa por procurar um caminho com recursos disponíveis capazes de satisfazer os requisitos de QoS especificados e depois reserva os recursos necessários ao longo desse caminho. Depois da conexão estabelecida a fonte envia os dados através desse caminho e a qualidade de serviço é garantida através da reserva de recursos efectuada. Existem inúmeras propostas, com diferentes alternativas, para implementar o encaminhamento por fluxo.

Neste capítulo são descritas algumas dessas propostas tanto ao nível do encaminhamento *unicast*, como ao nível do encaminhamento *multicast*.

3.1 Definições

Uma rede pode ser modelada por um grafo $G = (V, A)$. Os vértices V do grafo representam os encaminhadores e os sistemas terminais, e as arestas A representam as ligações entre eles. O grafo é um grafo não dirigido se as ligações são simétricas, ou seja se as ligações forem consideradas equivalentes nos dois sentidos e é um grafo dirigido no caso contrário, ou seja se as ligações forem assimétricas.

Na maiorparte das redes de comunicações, nomeadamente nas que consideram os requisitos de QoS do tráfego, as ligações são assimétricas, uma vez que a distribuição do volume de tráfego faz com que apresentem diferentes características em cada um dos sentidos. Cada ligação tem um estado que é medido através das métricas de QoS que estão a ser consideradas. Por exemplo, é possível definir uma função de "atraso na ligação" $d : A \rightarrow R^+$, que faz corresponder um número Real positivo a cada ligação da topologia de rede. O valor representado por $d(l)$ é basicamente uma medida do atraso experimentado pelos pacotes que atravessam a ligação $l \in A$, e depende do tempo de processamento do pacote, do atraso nas filas, do tempo de transmissão e do tempo de propagação. Da mesma forma, é possível associar a cada uma das ligações um parâmetro que corresponde à largura de banda disponível numa interface de saída, ou ao número de perdas de pacotes numa fila... Este conjunto de métricas que caracterizam uma ligação designa-se habitualmente por "estado da ligação". O conjunto dos diferentes "estados de ligação" de todas as ligações que constituem uma topologia designa-se por "estado da rede".

A complexidade dos protocolos de encaminhamento com QoS depende das métricas escolhidas para caracterizar os caminhos. As mais comuns são: os custos associados às ligações de forma administrativa, o número de saltos, a largura de banda disponível, os atrasos, as perdas e o *jitter* (variação no atraso sentido pelo receptor). Estas métricas podem ser classificadas em três categorias: aditivas, multiplicativas e côncavas.

Seja $m(i, j)$ um métrica relativa à ligação (i, j), ou seja, à ligação que liga o nó i ao nó j. Para o caminho $C = s \rightarrow i \rightarrow j \rightarrow ...l \rightarrow t$:

- a métrica m é côncava se $m(C) = min[m(s, i), m(i, j), ...m(l, t)]$. Por exemplo, a largura de banda disponível é uma métrica côncava, ou seja, a largura de banda

disponível num caminho é igual ao valor mínimo das larguras de banda disponíveis em cada uma das ligações que constituem esse caminho.

- A métrica m é aditiva se $m(C) = m(s,i) + m(i,j) + ... + m(l,t)$. Como exemplo pode referir-se o atraso fim a fim que é uma métrica aditiva, à semelhança do *jitter* e do custo. O atraso fim a fim é o somatório dos atrasos verificados em cada ligação que constitui o caminho.

- A métrica m é multiplicativa se $m(C) = m(s,i) * m(i,j) * ... * m(l,t)$. As perdas são um exemplo de métrica multiplicativa.

Os requisitos de QoS de uma conexão são dados em termos de um conjunto de restrições em relação às ligações que constituem os caminhos ou ao próprio caminho fim a fim. Uma restrição em relação às ligações é tipicamente uma métrica côncava como por exemplo a largura de banda disponível, que basicamente restringe as ligações que constituem um caminho possível a ligações com pelo menos uma determinada quantidade de largura de banda disponível. Uma restrição em relação ao caminho diz respeito a um requisito de QoS fim a fim, por exemplo, o atraso fim a fim ou o número máximo de perdas suportadas são restrições que se aplicam ao caminho.

Um caminho exequível é um caminho que tem disponíveis recursos suficientes para satisfazer os requisitos de QoS de uma conexão. O objectivo do encaminhamento com QoS é precisamente descobrir caminhos exequíveis que vão de encontro aos requisitos de QoS do tráfego. Adicionalmente pode também considerar a optimização de utilização dos recurso da rede. Essa utilização é habitualmente medida em temos de uma métrica abstracta definida como "custo".

3.2 Encaminhamento unicast com QoS

O problema do encaminhamento *unicast* com QoS pode ser definido da seguinte forma: dado um nó fonte f, um nó destino d e um conjunto de requisitos de QoS Q, o problema do encaminhamento *unicast* com QoS é encontrar o melhor caminho de f para d com recursos suficientes para satisfazer Q.

Dependendo das métricas utilizadas este problema pode dividir-se em problemas de diferentes tipos:

- Com uma única métrica podemos ter quatro tipos de problema, dependendo da natureza da métrica e do objectivo a alcançar:

  - Optimização das ligações, por exemplo da largura de banda disponível. Neste caso o que interessa é encontrar o caminho com maior largura de banda disponível, ou seja, o caminho cujas ligações têm maior largura de banda disponível.

  - Restrição das ligações, por exemplo da largura de banda disponível. Neste caso o que interessa é encontrar um caminho cuja largura de banda disponível seja superior a determinada quantidade
  - Optimização do caminho, por exemplo do atraso fim a fim. Neste caso o que interessa é encontrar o caminho com menor atraso fim a fim.
  - Restrição do caminho, por exemplo do atraso fim a fim. Neste caso o que interessa é encontrar um caminho cujo atraso fim a fim seja inferior a determinado valor.

  Qualquer destes problemas pode ser facilmente resolvido directamente usando o algoritmo de Dijkstra (ou o de Bellman-Ford)[19] com ligeiras variantes.

- Se se pretender usar mais do que uma métrica, dependendo dos tipos de métricas e do tipo de problema (optimização ou restrição) podemos ter problemas NP-Completos.
  - Duas ou mais restrições ao nível do caminho em que as métricas são independentes, por exemplo limitar o atraso fim a fim e o *jitter*.
  - Uma ou mais restrições e uma ou mais optimizações, ou seja existe pelo menos uma métrica que se quer restringir e pelo menos uma outra que se pretende optimizar. Por exemplo, o caminho de menor custo cujo atraso fim a fim é inferior a determinado valor.

- Também é possível combinar métricas e chegar a problemas que são facilmente resolvidos:
  - Optimização do caminho em relação a uma métrica e uma restrição nas ligações, por exemplo o problema de encontrar o caminho com menor atraso que tem pelo menos x unidades de largura de banda disponível em cada uma das ligações que o constitui.

3.2.1 Algoritmos de encaminhamento unicast com QoS

Ao nível do encaminhamento *unicast* com QoS, têm sido propostos muitos algoritmos diferentes nos últimos anos. Além de procurarem resolver problemas diferentes tendo em conta as diferentes métricas (ou requisitos de QoS), podem também distinguir-se uns dos outros por se basearem em heurísticas centralizadas ou em heurísticas distribuídas. Outra característica que nos permite diferenciá-los é o facto de permitirem o pré-cálculo de caminhos, ou apenas o cálculo dos caminhos a pedido.

3.2.1.1 Algoritmo proposto por Wang e Crowcroft

O algoritmo de Wang e Crowcroft[45] pretende fornecer uma solução para o problema de encontrar um caminho sujeito a restrições em termos do atraso fim a fim e da largura de banda disponível.

O algoritmo apresenta duas versões para a solução do problema: uma versão centralizada e uma distribuída. Na versão centralizada o caminho é estabelecido de forma centralizada pelo nó fonte e respeitado por todos os nós entre o nó fonte e o nó destino. Na versão distribuída as decisões de encaminhamento são tomadas nó a nó, por todos os nós no caminho entre a fonte e o destino.

Na versão centralizada do algoritmo a solução é encontrada em duas fases, primeiro começa-se por eliminar todas as ligações que não satisfaçam os requisitos da largura de banda e depois é calculado o caminho mais curto em termos de atraso fim a fim, usando para isso o algoritmo de Dijkstra. Se esse caminho obedecer à restrição imposta no atraso fim a fim, então encontrou-se um caminho exequível.

Na versão distribuída, Wang e Crowcroft, propõem que se encontre o caminho com maior largura de banda disponível. Se houver vários caminhos possíveis, escolhe-se entre esses o que tem menor atraso fim a fim.

Qualquer uma das versões deste algoritmo pressupõe que o cálculo dos caminhos exequíveis é feito a pedido e por fluxo. A informação do estado da rede é mantida em todos os nós por um protocolo de estado de ligação.

3.2.1.2 Algoritmo proposto por Guerin e Orda

O algoritmo proposto por Guerin e Orda[46] lida com a questão da imprecisão da informação de estado disponível nos encaminhadores. Esta questão é bastante importante quando estamos perante protocolos de encaminhamento com QoS. Como já foi referido os protocolos de encaminhamento trocam entre si informação relativa ao estado das suas ligações para poderem decidir qual o melhor caminho para encaminhar determinado pacote. Quando estamos perante protocolos de encaminhamento com QoS essa informação tem obrigatoriamente a ver com os recursos disponíveis na rede tanto nos encaminhadores como nas ligações. É essa informação que é usada para escolher um caminho que tenha os recursos necessários para satisfazer determinados requisitos. Ño entanto, o aparecimento constante de novos fluxos, assim como o fim de outros, faz com que a informação relativa à disponibilidade de recursos na rede esteja em constante mutação.

Por outro lado, os protocolos de encaminhamento não podem estar constantemente a comunicar essas alterações sob pena de sobrecarregarem demasiado, eles próprios com as suas mensagens de controle, a rede. O resultado desta limitação é muitas vezes a imprecisão da informação de estado.

Para lidar com este problema, o algoritmo de encaminhamento proposto por Guerin e Orda, considera um modelo baseado em funções de distribuição de probabilidades. O objectivo do algoritmo é encontrar os caminhos que tenham a maior probabilidade de acomodar um fluxo com determinados requisitos de QoS (largura de banda ou atraso fim a fim).

Para o caso da largura de banda, cada encaminhador mantém para cada ligação (l)

uma função de probabilidades ($pl(w)$) da ligação (l) ter pelo menos w largura de banda disponível. Esta função é estabelecida usando o último valor anunciado da largura de banda disponível juntamente com a informação relativa à variação máxima que faz com que seja despoletada uma nova actualização. Nos protocolos de estado de ligação, o mecanismo de actualização das métricas é quase sempre baseado em *thresholds*. Ou seja, um encaminhador só desencadeia uma actualização do seu estado, quando ele é significativamente diferente do último anunciado, por exemplo, uma actualização é despoletada se a diferença entre o valor actual e o último anunciado for superior a um determinado valor.

Uma vez estabelecida a função de distribuição de probabilidades, para todas as ligações presentes na topologia, o algoritmo de Guerin e Orda procura encontrar o caminho que maior probabilidade tem de satisfazer os requisitos de largura de banda expressos por cada nova conexão. Assim, se uma determinada conexão estabelecer como requisito x unidades de largura de banda, o problema da descoberta de um caminho exequível pode ser resolvido pelo algoritmo de descoberta do caminho mais curto em que o custo de cada ligação l, é dado por $(-log\ pl(x))$.

Este algoritmo assume um ambiente em que o nó fonte é confrontado com o pedido de estabelecimento de um novo fluxo com determinados requisitos de QoS, expressos em termos da largura de banda disponível e é responsável por encontrar o caminho mais adequado. Por outras palavras, encaixa no modelo de encaminhamento centralizado na fonte. Por outro lado, como pressupõe que o encaminhador fonte (o que calcula o melhor caminho) tem um conhecimento completo da topologia de rede e do estado de todas as ligações e nós, pode classificar-se como sendo um algoritmo de estado de ligação.

Por último, em relação à forma como lida com os requisitos de QoS, é um protocolo que faz o encaminhamento por fluxo e a pedido, necessitando de um protocolo de reserva de recursos (por exemplo, o RSVP) para manter a qualidade dos caminhos estabelecidos.

Guerin e Orda procuraram também resolver este mesmo problema no caso em que os requisitos de QoS são expressos em função do atraso fim a fim. No entanto neste caso, a imprecisão da informação de estado tem um impacto muito grande na complexidade que introduz no processo de selecção do melhor caminho, impacto esse que no entender dos autores torna o problema intratável. Apesar disso, através de algumas heurísticas que basicamente procuram mapear a restrição global (em termos de atraso fim a fim) em restrições locais por cada par nó/ligação, propõem alternativas que podem em algumas situações fornecer soluções aceitáveis para o problema.

3.2.1.3 Algoritmo proposto por Apostolopoulos e Guerin

Apostolopoulos e Guerin[47] propõem uma variante do algoritmo de Bellman-Ford para resolverem o problema de encontrar o menor caminho (com o menor número de saltos) com a maior largura de banda disponível.

A ideia básica do algoritmo consiste em construir uma matriz $[mxn]$ em que m é o número de destinos possíveis e n o número máximo de saltos que pode ter um caminho.

Cada elemento desta matriz contém informação acerca do melhor caminho a seguir para determinado destino i (em que i é uma das linhas da matriz), caminho esse que terá no máximo j saltos, (sendo j uma coluna da matriz). O melhor caminho é escolhido em função de uma única métrica, a largura de banda disponível.

Para construir esta matriz um determinado encaminhador, $n0$, começa por preencher a primeira coluna da matriz com a informação acerca dos caminhos para os seus vizinhos. Depois disto, como estamos perante uma variante dos algoritmo de Bellman-Ford o encaminhador $n0$ começa a receber anúncios dos vizinhos. Considera-se que a segunda iteração do algoritmo acontece quando o $n0$ recebe o primeiro anuncio dos seus vizinhos, a terceira quando recebe o segundo anuncio e assim sucessivamente. Ou seja, na segunda iteração o encaminhador $n0$ preenche a segunda coluna da matriz (caminhos com 2 saltos no máximo), na terceira preenche a terceira coluna (caminhos com 3 saltos no máximo) e assim sucessivamente. Em cada uma destas iterações, começa-se por copiar o conteúdo da coluna anterior para a coluna actual, e depois compara-se cada um dos elementos copiados com o conteúdo do anuncio recebido. Se a informação contida no anuncio recebido mostrar um caminho possível com maior largura de banda disponível o elemento da matriz é substituído pelo do anuncio. Senão mantém-se igual.

Depois da matriz construída é muito fácil seleccionar o caminho mais curto com determinada largura de banda disponível. Ao longo de cada linha o valor da largura de banda disponível aumenta, assim como o número de saltos. Assim sendo, basta seguir uma linha desde o inicio: o primeiro valor encontrado com uma largura de banda disponível que satisfaça é de certeza o caminho mais curto que respeita os requisitos de largura de banda.

Este algoritmo é dos poucos algoritmos que suportam a utilização de pré-calculo de caminhos e não o cálculo dos caminhos a pedido. Foi concebido para o modelo de encaminhamento distribuído, ou seja, as decisões de encaminhamento são tomadas nó a nó.

3.2.1.4 Algoritmo proposto por Salama *et al*

Salama *et al*[48] propuseram um algoritmo de encaminhamento distribuído para resolver o problema de encontrar o caminho mais curto (de custo mínimo) com restrições no atraso fim a fim.

Segundo o algoritmo proposto são mantidos dois vectores em cada nó que constitui a topologia por um protocolo de vector de distância: um vector *cost* que contém para cada destino possível o próximo nó no caminho de custo mínimo, e um vector *delay* que contem para cada destino possível o próximo nó no caminho de atraso mínimo. Quando uma fonte pretende começar a emitir é enviada uma mensagem de controle da fonte até ao destino no sentido de construir o caminho de custo mínimo com restrições no atraso fim a fim. Todos os nós ao receber essa mensagem de controle terão de escolher uma das duas alternativas representadas nos vectores *cost* e *delay*. A ligação representada no vector *cost*

tem prioridade desde que se conclua que, no nó seguinte, a ligação no caminho de atraso mínimo não viola a restrição imposta ao nível do atraso fim-a-fim. Se for esse o caso é a ligação do vector *delay* que é escolhida.

O facto da mensagem de controle ora seguir o caminho de custo mínimo, ora o caminho de atraso mínimo pode conduzir ao aparecimento de ciclos, que basicamente têm como consequência o facto da mensagem de controle não conseguir atingir o destino. Para resolver esse problema, Salama *et al* propõem um mecanismo que consiste basicamente no seguinte: quando é detectado um ciclo, ou seja, quando um nó detecta que está a receber a mensagem de controle pela segunda vez, o processo de encaminhamento tem de recuar até ao último nó que escolheu seguir pelo caminho de custo mínimo. O processo de encaminhamento recomeça a partir desse nó escolhendo o caminho de atraso mínimo em vez do caminho de custo mínimo. Os autores conseguiram provar que este mecanismo funciona e consegue evitar os ciclos, muito embora seja necessário garantir que a informação contida nos vectores *cost* e *delay* está actualizada ou no mínimo consistente, o que nem sempre é fácil num rede dinâmica.

Sun e Langendorfer[49] propuseram algumas alterações ao algoritmo de Salama *et al* no sentido de colmatar algumas das suas limitações, nomeadamente a questão da detecção e remoção de ciclos que representa um sério revés neste algoritmo.

A ideia subjacente ao algoritmo proposto por Sun e Langendorfer é evitar os ciclos em vez de os detectar e remover. Para isso a mensagem de controle viaja sempre através do caminho de atraso mínimo até ao nó em que o caminho de custo mínimo satisfaz a restrição imposta ao atraso fim-a-fim. A partir daí a mensagem de controle passa a viajar pelo caminho de custo mínimo até atingir o destino. Para concretizar esta alteração, é necessário que os vectores *cost* e *delay* contenham ambos as duas métricas: custo e atraso.

3.2.1.5 Algoritmo proposto por Costa e Fdida

Costa e Fdida propuseram um algoritmo que tenta verificar os requisitos de QoS expressos em função de três métricas: largura de banda disponível, atrasos e perdas[50].

A largura de banda é tratada usando a técnica proposta por Wang e Crowcroft [1], ou seja, eliminando todas as ligações que não têm largura de banda disponível em quantidade suficiente para satisfazer os requisitos de QoS.

As outras duas métricas, atraso e perdas, são combinadas originando uma única métrica com uma componente inteira e uma componente decimal. A componente inteira corresponde aos atrasos, partindo do principio, os autores, que os atrasos são sempre expressos na unidade de milissegundos, sendo valores inteiros. A componente decimal da métrica é o valor absoluto do logaritmo da probabilidade de sucesso de transmissão. Ou seja, a probabilidade de um pacote ser descartado é mapeada na componente decimal da métrica, transformando-se além disso uma métrica multiplicativa numa métrica aditiva

[1] já detalhada na secção 3.2.1.1 deste capítulo

com a aplicação da função logaritmo.

O algoritmo consiste em, depois de eliminadas as ligações que não satisfazem os requisitos impostos em termos de largura de banda disponível, aplicar uma variante do algoritmo de Dijkstra usando a métrica combinada. Claro que não chega encontrar o caminho que minimiza a métrica. Se o fizéssemos estar-se-ia simplesmente a minimizar o atraso, que é a componente inteira da métrica, a menos que houvessem dois caminhos com o mesmo atraso. Só nesse caso é que seria possível minimizar as duas componentes. No entanto, se P for o limite imposto ao logaritmo da probabilidade de sucesso de transmissão e $Slog(i,j)$ for o logaritmo da probabilidade de sucesso de transmissão na ligação (i,j) é possível concluir que num dado caminho C com x saltos, se $Slog(i,j)$ for menor do que P/x em todas as ligações que constituem o caminho C, então $Slog(C)$ é menor do que P. O único problema deste raciocínio é o facto de só se saber o número total de saltos x quando o algoritmo termina a sua execução. Para resolver esta questão, Costa e Fdida, propõem a utilização de duas variáveis que são actualizadas iteração a iteração. A primeira (SS), é a diferença entre o tamanho máximo que um caminho pode ter e o tamanho do caminho actual. A segunda (SP), é a diferença entre o limite P e o valor actual do logaritmo da probabilidade de sucesso da transmissão. Os autores propõem assim uma variante do algoritmo de Dijkstra em que em cada iteração é escolhido o caminho mais curto nos termos da métrica única desde que a parte decimal da métrica seja menor do que SP/SS.

3.2.1.6 Algoritmo proposto por Shin-Chou

Shin-Chou propuseram um algoritmo distribuído baseado na técnica de *path probing*[51]. Ao receber um pedido para estabelecer uma conexão, um encaminhador envia para todos os seus vizinhos, mensagens (*probes*) destinadas ao destinatário da conexão. Estas mensagens transportam não só os requisitos de QoS da nova conexão em termos de atraso fim a fim, mas também o atraso acumulado pela própria mensagem no caminho já percorrido. Os encaminhadores no caminho só re-enviam estas mensagens através das ligações cuja a soma do atraso com o atraso acumulado especificado nas mensagens não exceda o requisito de QoS da conexão.

Um determinado *probe* que chegue ao destino, atravessou um caminho exequível capaz de satisfazer os requisitos da nova conexão. O encaminhador destino selecciona então de entre todos os *probes*, o melhor, estabelece a ligação e procede à reserva de recursos pelo caminho por ele percorrido.

Este algoritmo tem a grande vantagem de não exigir que todos os encaminhadores possuam um conhecimento global da topologia e do estado das ligações. Por outro lado os *probes* introduzem alguma sobrecarga na rede, o que pode ser critico se a rede estiver congestionada. O tempo que se demora a estabelecer uma conexão é também por vezes demasiado longo.

Chen e Nahrsted propuseram algumas melhorias e alterações ao algoritmo proposto

por Shin e Chou no sentido de colmatar algumas das suas limitações[52]. Segundo estes autores o caminho mais curto deve ser sempre o escolhido caso seja um caminho exequível, só havendo necessidade de procurar caminhos alternativos se o caminho mais curto não tiver os recursos suficientes para satisfazer os requisitos de QoS.

3.2.2 Discussão

Como já foi referido no capítulo 2, as heurísticas centralizadas apresentam um conjunto de desvantagens que no caso do encaminhamento com QoS podem ainda tornar-se mais problemáticas.

O modelo subjacente ao encaminhamento centralizado (adoptado pelas propostas referidas nas secções 3.2.1.1 e 3.2.1.2) pressupõe que o estado global da rede seja mantido em cada um dos nós. Este facto, traz ao encaminhamento *unicast* com QoS as seguintes dificuldades:

- Em primeiro lugar as actualizações desta informação têm que ser suficientemente frequentes para lidar correctamente com o dinamismo das métricas de QoS consideradas, de que são exemplo a largura de banda disponível em cada uma das ligações, os atrasos, etc. A actualização destes parâmetros, que apresentam habitualmente um grande dinamismo pode sobrecarregar excessivamente as redes, principalmente se se tratarem de redes de grandes dimensões.

- Em segundo lugar, estas mensagens de actualização sofrem, elas próprias, atrasos, nem que seja apenas devido ao tempo de propagação, partindo do principio que a rede não está congestionada e não há filas. Este atraso é responsável por um certo grau de imprecisão na informação de estado. Quando as mensagens de actualização chegam aos diferentes nós da topologia, a informação de estado que contêm, pode já estar desactualizada.

- Por último, o próprio protocolo de estado de ligação em que se baseiam estas heurísticas é pouco escalável, pela quantidade de informação que obriga a manter. Não esquecer que cada nó individualmente tem que manter informação de estado detalhada, não só em relação ao nós mas também em relação a todas as ligações que constituem a rede.

Estes problemas mantêm-se, apesar de minorados, nas heurísticas distribuídas que requerem a manutenção do estado global da rede em cada um dos nós (estratégia adoptada pelas propostas referidas nas secções 3.2.1.3, 3.2.1.4 e 3.2.1.5). No entanto, o facto do cálculo dos caminhos ser repartido pelos vários nós diminui o tempo de resposta, o que torna o algoritmo mais escalável.

Uma outra abordagem é a estratégia de *Path Probing* seguida pelos algoritmos descritos na secção 3.2.1.6. Segundo esta estratégia, os diferentes nós não necessitam de manter

o estado global da rede, apenas informação de estado local. Neste caso, quando uma fonte pretende iniciar uma conexão, "espalha" pela rede um conjunto de mensagens de controle destinadas ao receptor, mensagens essas que procuram caminhos exequíveis, ou seja caminhos que possam satisfazer os requisitos especificados pela fonte. São as próprias mensagens de controle que ao longo dos diferentes percursos tornam possível que cada um do nós intermédios averigue se existem ou não recursos suficientes para satisfazer os requisitos. Este tipo de algoritmos pressupõe que os caminhos são sempre calculados a pedido, ou seja não é possível implementar o pré-cálculo de caminhos.

3.3 Encaminhamento Multicast com QoS

O *multicast* é particularmente útil para as aplicações multimédia que envolvem vários intervenientes, por exemplo, aplicações que implementam vídeo-conferência, ensino-à-distância, trabalho cooperativo, etc. Muitas destas aplicações têm requisitos de Qualidade de Serviço. As árvores de custo mínimo são adequadas para transmitir dados que não são sensíveis à qualidade de serviço, mas quando as aplicações têm requisitos de QoS há uma grande probabilidade das árvores de custo mínimo não serem as mais adequadas.

Para suportar esse tipo de aplicações é necessário encontrar uma árvore "exequível" (*feasible tree* na terminologia inglesa) que não é mais do que uma árvore *multicast* (se possível a árvore de custo mínimo), que obedece a uma determinada restrição ou então a várias. Estas restrições são tipicamente requisitos de qualidade de serviço exigidos por uma determinada aplicação (disponibilidade de largura de banda, limite no atraso fim-a-fim, limite de perdas, etc).

Até aqui o problema do encaminhamento *multicast* consistia basicamente em encontrar uma árvore de custo mínimo que interligasse todos os membros de um grupo *multicast*. Como foi referido no capítulo 2 esse problema é designado por problema da árvore de *Steiner* e é um problema NP_Completo. Actualmente, tendo como objectivo dotar a infraestrutura de rede da capacidade de implementar encaminhamento com QoS, o problema do encaminhamento *multicast* transformou-se num problema ainda mais complexo. Além de se pretender encontrar a árvore de custo mínimo, árvore essa que optimiza a utilização dos recursos de rede, também se pretende que essa árvore respeite determinados requisitos de QoS. Esses requisitos podem ser impostos a cada umas das ligações que constitui a árvore, individualmente (por exemplo se se tratar de um requisito expresso em termos da largura de banda disponível), mas também podem ser requisitos impostos aos caminhos que unem as fontes a cada um dos destinatários (se se tratar por exemplo dos atrasos fim a fim).

No entanto, o problema da árvore de *Steiner* pretende optimizar a métrica custo como um todo, ou seja o custo total da árvore e não o custo individual de cada um dos caminhos que unem as fontes a cada um dos destinatários. Ou seja, o problema do encaminhamento *multicast* com QoS é encontrar a melhor árvore de distribuição *multicast* que optimize determinada métrica, tipicamente o custo das ligações que a constituem, de

forma a optimizar a utilização dos recursos de rede. Adicional e simultaneamente deve ainda suportar os requisitos das aplicações, requisitos esses que poderão ter características fim a fim, e como tal têm que ser medidos ao longo dos diferentes caminhos entre as fontes e cada um dos destinatários.

O problema de encontrar a melhor árvore de distribuição *multicast* que satisfaça requisitos de Qualidade de Serviço é designado por Problema da Árvore de *Steiner* Sujeita a Restrições (*Constrained Steiner Tree Problem* na terminologia inglesa). Como já foi referido, o Problema da Árvore de *Steiner* é um problema NP_Completo. O facto de acrescentarmos restrições suplementares - os requisitos de QoS - em nada altera esse facto; pelo contrário, complica ainda mais as heurísticas que poderão contribuir com soluções aproximadas para o problema.

Dependendo das restrições impostas e dos parâmetros que interessa optimizar (ou seja, das métricas utilizadas) o problema do encaminhamento *multicast* com QoS pode classificar-se em diferentes categorias[53]:

- Problema do encaminhamento *multicast* com restrições impostas às ligações úteis, por exemplo, se se pretender construir uma árvore de distribuição de tráfego *multicast* em que existe em todos os ramos um mínimo de x unidades de largura de banda disponível. Este problema é facilmente resolvido, mesmo que se pretenda impor mais do que uma restrição às ligações. Normalmente a solução encontrada passa por eliminar da topologia de rede todas as ligações que não satisfazem as restrições;

- Problema do encaminhamento *multicast* com restrições impostas aos diferentes ramos úteis, por exemplo, se a restrição imposta for relativa ao atraso fim-a-fim. Este problema é NP_completo se se pretender usar mais do que uma restrição deste tipo, e é facilmente tratável no caso de só haver uma restrição;

- Problema do encaminhamento *multicast* com uma restrição imposta às ligações e outra imposta aos diferentes ramos fim-a-fim úteis, por exemplo determinar a árvore com restrições na largura de banda disponível e atraso fim-a-fim. Neste caso o problema pode ser resolvido de acordo com a estratégia proposta por Wang e Crowcrof[45] e já detalhada na secção 3.2.1.1 deste capitulo;

- Problema da optimização da árvore também conhecido por problema da árvore de *Steiner*. Este problema é NP_completo, e já foram apresentadas algumas heurísticas para a sua resolução no capitulo 2 deste documento;

- Problema da optimização da árvore sujeita a restrições também conhecido por problema da árvore de *Steiner* com restrições. Este problema, à semelhança do anterior é um problema NP_completo, sejam essas restrições impostas às ligações ou aos ramos que constituem a árvore.

3.3.1 Algoritmos de Encaminhamento Multicast com QoS

À semelhança do que acontece no encaminhamento *unicast*, as heurísticas existentes e que contribuem com soluções aproximadas para o problema da árvore de *Steiner* sujeita a restrições podem dividir-se em duas categorias distintas: heurísticas centralizadas e heurísticas distribuídas. As heurísticas centralizadas partem do pressuposto de que os vários nós da topologia possuem um conhecimento completo da constituição do grupo. Estas heurísticas são pouco adequadas à resolução do problema no contexto do *multicast* IP, uma vez que uma das premissas do modelo é precisamente o facto dos grupos serem dinâmicos e não ter de haver um conhecimento prévio da constituição do grupo.

3.3.1.1 Extensões aos protocolos de encaminhamento multicast existentes

Os vários protocolos de encaminhamento *multicast* já detalhados no capítulo 2 podem distinguir-se quanto ao tipo de árvores *multicast* que utilizam para distribuir os dados. Existem duas alternativas:

- Árvores centradas na fonte, também chamadas *shortest path trees*, que são árvores dos caminhos mais curtos entre fonte e cada um dos destinatários. Esta abordagem é utilizada pelo DVRMP, MOSPF e PIM-DM;
- Árvores partilhadas centradas num nó da rede (designado por *Rendez-vous Point* ou *Core Router*, na terminologia inglesa) que interligam todos os membros do grupo. Quando uma fonte quer enviar um pacote para um grupo *multicast* envia-o para a raíz da árvore e esta depois utiliza a árvore partilhada para distribuir o pacote a todos os receptores. Esta abordagem é seguida pelo PIM-SM e pelo CBT.

O CBT é o único dos cinco protocolos referidos acima que não suporta árvores centradas na fonte. Todos os pacotes de dados são transmitidos através de uma única árvore partilhada bidireccional.

Na ausência de requisitos de QoS e reservas de recursos, uma árvore deste tipo pode trazer alguns benefícios porque minimiza o número de ligações envolvidas por grupo *multicast*. Por outro lado, a partilha de ligações provoca alguma concentração de tráfego e consequentemente uma previsível degradação do desempenho fim-a-fim de uma sessão *multicast*.

Para o *multicast* com reserva de recursos esta árvore partilhada coloca ainda dois problemas adicionais:

- Se as ligações são partilhadas por várias fontes, as ligações podem ter que suportar reservas para mais do que uma delas. O número de ligações que podem potencialmente ser usadas na abordagem seguida pelos algoritmos que usam árvores centradas na fonte é potencialmente maior, ou seja, a quantidade de tráfego sujeito a restrições (e

consequentemente o número de membros do grupo) é potencialmente mais pequena no caso do CBT;

- A selecção do *core router* não é baseada nos requisitos de QoS dos vários receptores e pode vir a dificultar o acondicionamento destes requisitos por limitar de certa forma a topologia da árvore.

Ignorando de certa forma estes problemas, Hou *et al*[54] propuseram algumas extensões ao protocolo CBT de forma a que este consiga suportar o encaminhamento com QoS. As extensões propostas abrangem o processo de junção (*join*) de um novo membro ao grupo e o processo de abandono (*leave*) de um membro do grupo. Cada encaminhador mantém uma base de dados com o nível de QoS disponibilizado pela sub-árvore respectiva. Quando um encaminhador recebe um pedido de *join* verifica se o nível da QoS requerido pelo novo membro pode ser satisfeito pela sua sub-árvore, consultando para isso esta base de dados. No caso do novo membro ser uma fonte é necessário verificar também se o QoS já disponibilizado poderá ser afectado pela nova fonte. A mensagem de *join* tem que chegar à raíz da árvore uma vez que cada encaminhador mantém apenas o estado para a sua sub-árvore. Segundo este protocolo a manutenção da QoS da árvore é mantida restringindo o sucesso das mensagens de *join*.

Tanto o DVRMP como o PIM-DM usam árvores centradas na fonte, árvores essas que são construídas recorrendo ao mecanismo de *flooding*. Graças a este mecanismo, os pacotes de dados começam a chegar aos potenciais receptores ainda antes de eles se terem juntado ao grupo. Ou seja, o caminho entre uma fonte e um destino não pode ser escolhido tendo em conta os requisitos de qualidade de serviço especificados por este último.

O facto do MOSPF suportar simultaneamente árvores centradas na fonte e *joins* iniciados pelos receptores torna este protocolo um potencial candidato para suportar *multicast* com qualidade de serviço. Para construir uma árvore *multicast* um encaminhador necessita de saber quem são os elementos de um grupo. Este informação já é disponibilizada no MOSPF, através de um LSA (Group MemberShip LSA) específico que é periodicamente difundido por todos os encaminhadores de uma área. Cada um dos encaminhadores pode então, assim que chega um pacote *multicast*, calcular uma árvore dos caminhos mais curtos (*shortest path tree*, na terminologia inglesa) por cada par fonte/grupo, através da qual difunde o pacote. Adicionalmente, para suportar QoS, um encaminhador pode, usando os mecanismos de disseminação do estado das ligações existente no MOSPF, ficar a saber quais é que são os recursos já reservados nas várias ligações por um determinado grupo e desta forma quando um pacote *multicast* chegar a um encaminhador, usando esta informação, consegue calcular a árvore necessária para distribuir aquele pacote de acordo com a especificação dos requisitos de qualidade de serviço.

O PIM-SM além de suportar árvores baseadas na fonte, possui um mecanismos para construir uma árvore partilhada uni-direccional com raíz num ponto pré-definido (o *rendezvous point*). Um novo membro começa por se juntar a essa árvore partilhada, e só depois, se a qualidade de serviço não o satisfizer, é que se junta a uma árvore centrada na fonte que lhe possa garantir a qualidade de serviço pretendida. De todos é talvez, o que melhor

se adequa a incorporar mecanismos que lhe permitirão disponibilizar encaminhamento *multicast* com qualidade de serviço.

Em [55], Biswas *el al*, propõem uma estratégia de encaminhamento com QoS para ser incorporada no protocolo PIM-SM. Os pressupostos são a heterogeneidade do diferentes membros do grupo em termos de requisitos de QoS e a existência de um protocolo de encaminhamento *unicast* com QoS capaz de disponibilizar informação de estado relativa à disponibilidade de recursos nos diferentes nós e ligações da topologia. As heurísticas propostas têm como objectivo descobrir caminhos para ligarem os novos membros às diferentes fontes mediante os seus requisitos. A quantidade de informação de estado que é necessário manter em cada um dos nós é a grande desvantagem desta proposta.

3.3.1.2 Algoritmo proposto por Kompella *et al*

Kompella *et al*[56] propuseram uma heurística centralizada designada por heurística KPP (KPP são as iniciais dos autores da heurística: Kompella, Pasquale e Polyzo), que disponibiliza uma solução aproximada para o problema da árvore de Steiner sujeito a restrições de atrasos.

Os autores assumem que tanto os atrasos em cada ligação, como o limite para o atraso fim-a-fim são valores inteiros, enquanto que o custo de cada ligação é um número real positivo. A heurística é uma variante da heurística KMB[31] já descrita no capítulo 2 e usada para resolver o problema da árvore de *steiner* não sujeito a restrições. Segundo a heurística KPP o grafo construído a partir da topologia de rede não é apenas um subgrafo completo de distâncias. Em vez disso, as arestas que unem dois quaisquer vértices do subgrafo correspondem aos caminhos mais curtos que satisfazem a restrição do atraso, ou seja, cujo atraso é menor do que o limite imposto para o atraso fim-a-fim. Depois à semelhança do que acontece com a heurística KMB, é necessário encontrar uma *spanning tree*. Isso é feito usando um algoritmo semelhante ao algoritmo de Prim[30].

Kompella *et al* propõem duas heurísticas diferentes: uma delas tenta apenas minimizar o custo da árvore enquanto obedece à restrição imposta ao atraso; a outra procura combinar custos com atrasos. Ou seja, na primeira a *spanning tree* é construída procurando sempre a ligação de menor custo, exactamente à semelhança do que acontece com o algoritmo de Prim, extendendo-o para, em cada iteração, verificar se a restrição de atraso continua a ser respeitada senão escolhe-se outra ligação. Na segunda heurística, ao mesmo tempo que se minimiza o custo tenta maximizar-se o atraso na tentativa de evitar que essa ligação continue a ser escolhida para ligar outro nó. O objectivo é aumentar a utilização das ligações e como efeito lateral provocar uma diminuição nos atrasos fim-a-fim.

Finalmente as arestas da *spanning tree* obtida são substituídas pelos caminhos correspondentes, e por fim são retirados os ciclos, caso existam.

3.3.1.3 Algoritmo proposto por Van Mieghem e Kuipers

Van Mieghem e Kuipers propuseram um algoritmo de encaminhamento *multicast* designado por MAMCRA (Multicast Adaptative Multiple Contraints Routing Algorithm)[57]. Este algoritmo consegue garantir os requisitos de QoS aos membros de um grupo *multicast* em prejuízo por vezes da função de custo. O algoritmo funciona da seguinte forma:

- Começa por encontrar o conjuntos dos caminhos mais curtos, em termos das métricas de QoS a considerar, que ligam a fonte a cada um dos destinos. Para isso utilizam um protocolo de encaminhamento *unicast* com QoS, o SAMCRA[58] (desenvolvido pelos mesmos autores), que pretende encontrar caminhos que satisfaçam múltiplas restrições;
- Depois, procura substituir cada um deste ramos por caminhos de menor custo, desde que estes não violem as restrições impostas à métrica de QoS considerada.

Os caminhos mais curtos em termos das métricas de QoS, são encontrados recorrendo ao algoritmo SAMCRA. Depois são removidos os ramos duplicados de forma a que as replicações de pacotes só sejam efectuadas quando é estritamente necessário. Um conjunto de caminhos deste tipo pode conter ciclos, ou seja, é natural que dois dos caminhos se cruzem em um ou mais pontos. Portanto, numa segunda fase procura-se eliminar estes ciclos, se isso for possível, sem violar as restrições impostas.

3.3.1.4 Algoritmo proposto por Calberg e Crowcroft

Calberg e Crowcroft propuseram um protocolo, o protocolo YAM[59], que propõe algumas alterações às árvores partilhadas do PIM-SM e do CBT no sentido de fornecer a um novo membro várias alternativas para o estabelecimento de um novo ramo.

Como já foi referido aquando da descrição destes dois protocolos, as árvores partilhadas são construídas a partir do nó onde se ligou o novo membro em direcção a um nó que já esteja na árvore. O PIM e o CBT usam a informação de encaminhamento *unicast* para redireccionar o seus pedidos de *join* em direcção à raíz da árvore partilhada (designado por *Rendez-vous Point* no PIM-SM e por *Core Node* no CBT). Ao longo do seu caminho esta mensagem de *join* vai construindo o novo ramo que irá permitir a junção de um novo membro ao grupo. Quando finalmente a mensagem de *join* encontra um nó que já está na árvore (que é sempre o nó na árvore partilhada que está mais próximo do novo membro) o processo de junção do novo membro termina. Ou seja, os novos membros são ligados à árvore partilhada através do caminho mais curto desde o novo membro até à raíz da árvore.

O objectivo do YAM é conseguir encontrar não um ramo, mas sim vários ramos alternativos, para juntar um novo membro a uma árvore.

Esta estratégia, com algumas variantes, foi depois usada por vários protocolos que pretendem implementar o encaminhamento com QoS. Este foi o primeiro de muitos protocolos que basearam o seu funcionamento no *flooding* no sentido de encontrar caminhos alternativos para juntar um novo membro a uma árvore de distribuição *multicast* já parcialmente construída.

O YAM é então um algoritmo que constrói árvores *multicast* capazes de satisfazer determinados requisitos de QoS (à partida impostos pela fonte). Para cada novo membro são pesquisadas diferentes alternativas de junção à árvore, ou seja a mensagem de *join* é difundida para todos os caminhos possíveis no sentido de procurar um nó que já esteja na árvore. Quando a mensagem de *join* chega a um nó que já está na árvore a difusão pára e o nó envia por *unicast* uma mensagem para o originador da mensagem de *join*. Ou seja é de esperar que ao fim de algum tempo o nó que enviou a mensagem de *join* receba várias mensagens de resposta provenientes de diferentes nós da árvore. Entre as diferentes possibilidades o candidato a novo membro escolhe a que mais lhe convém, tendo em conta os seus requisitos de QoS. É só depois disso que é estabelecido um novo ramo da árvore através de uma mensagem que o novo membro envia exactamente pelo mesmo caminho por onde recebeu a resposta ao *join* escolhida.

3.3.1.5 Algoritmo proposto por Michalis Faloutsos *et al*

Michalis Faloutsos *et al* propuseram um protocolo, o QoSMIC [60], que procura extender o protocolo YAM de forma a contornar algumas das suas desvantagens, nomeadamente o excesso de mensagens de difusão introduzidas na rede, na altura em que um novo membro se junta à árvore.

O QoSMIC utiliza simultaneamente dois processos para construir uma árvore que satisfaça os requisitos de Qualidade de Serviço: um processo de procura local e outro de procura a partir da árvore *multicast* já construída. A procura local é iniciada pelo candidato a novo membro através da difusão de mensagens do tipo BID-REQ para os seus vizinhos. O âmbito destas mensagens é controlado através do TTL (Time To Live) das mensagens. Se algum nó da árvore *multicast* receber uma mensagem BID-REQ torna-se um nó candidato e responde com uma mensagem BID que é enviada por *unicast* para o candidato a novo membro. Ao longo do seu caminho a mensagem BID vai recolhendo informações sobre o estado das várias ligações que atravessa, informações essas que poderão ser usadas pelo candidato a novo membro para seleccionar o ramo que mais lhe convém.

O processo de procura através da árvore *multicast* já construída é iniciado por um nó especial designado por gestor. Esse nó recebe uma mensagem do tipo M-JOIN do candidato a novo membro e responde enviando mensagens do tipo BID-ORDER a um conjunto de nós da árvore, que se tornam assim candidatos a conduzir a construção de um novo ramo. Esses nós enviam mensagens do tipo BID ao candidato a novo membro exactamente como no processo de procura local.

3.3.1.6 Algoritmo proposto por Shigang Chen e Klara Nahrstedt

Shigang Chen e Klara Nahrstedt propuseram um protocolo designado por QRMP (QoS-Aware Multicast Routing Protocol)[61]. À semelhança dos protocolos YAM e QoSMIC também o QRMP é baseado na estratégia de *path probing* (difusão não selectiva, ou *flooding*, de mensagens de controle).

No protocolo QRMP existem dois modos de procura: o modo de caminho único e o modo de caminhos múltiplos. O processo de encaminhamento começa pelo modo de caminho único. Nesse modo a procura de um caminho que satisfaça os requisitos é feita através de um único caminho, o mais curto. Neste caso, é enviada uma mensagem de *join request* em direcção à árvore *multicast* através do caminho mais curto. A mensagem de *join request* contem os requisitos de QoS do candidato a novo membro e ao longo do caminho cada nó que recebe a mensagem verifica se os requisitos continuam a poder ser satisfeitos.

Se um encaminhador verifica que não têm recursos para satisfazer determinado pedido, muda para o modo de funcionamento de pesquisa por múltiplos caminhos. Esse modo de funcionamento é activado através de uma mensagem do tipo *not acknowledge* que é enviada para o encaminhador anterior, ou seja, o último encaminhador com recursos suficientes. Esse encaminhador, ao receber a mensagem de *not acknowledge* envia uma mensagem do tipo *join request* para todos os nós vizinhos, excepto para aqueles através dos quais foram recebidas as mensagens de *join request* e de *not acknowledge*.

Uma vez descoberto um caminho que satisfaça os requisitos do candidato a novo membro, é enviada uma mensagem do tipo *acknowledge* de volta até ao nó que activou o modo de funcionamento dos caminhos múltiplos. Se esse nó receber mais do que uma mensagem de *acknowledge*, terá que seleccionar entre as várias hipóteses o melhor caminho e rejeitar os outros.

3.3.1.7 Algoritmo proposto por Ion Stoica (Reunite)

Ion Stoica propôs um protocolo, o REUNITE[62], que implementa a distribuição do tráfego *multicast* com base na infra-estrutura de encaminhamento *unicast*, mais concretamente, usando árvores *unicast* recursivas. A motivação para o desenvolvimento deste protocolo assenta no pressuposto de que, numa árvore de distribuição *multicast* típica a maioria dos nós implementa simplesmente o re-envio de pacotes de uma interface de entrada para uma única interface de saída, ou seja, poucos são os nós da árvore que implementam a replicação de pacotes.

Apesar disso, todos os protocolos de encaminhamento *multicast* mantém informação por grupo em todos os encaminhadores que fazem parte da árvore de distribuição *multicast*. O protocolo REUNITE propõe a divisão dessa informação em duas tabelas distintas: a *Multicast Control Table (MCT)*, que é mantida em todos os encaminhadores que fazem parte da árvore de distribuição *multicast*, e a *Multicast Forwarding Table (MFT)* que é

mantida apenas nos encaminhadores onde é necessário replicar pacotes.

O principio básico subjacente ao funcionamento deste protocolo é a utilização de endereçamento *unicast*. Os encaminhadores responsáveis pela replicação de pacotes, ao criarem as diferentes réplicas utilizam endereços *unicast* destino de forma a que todos os membros de um grupo recebam o tráfego destinado ao grupo. Isto é concretizado a partir da tabela MFT que para cada fonte tem uma entrada com os endereços *unicast* de alguns dos membros, os suficientes para que seja enviada uma cópia para cada ramo da árvore.

Para construir as árvores de distribuição *multicast* o REUNITE utiliza dois tipos de mensagens: a mensagem de *join* que basicamente não altera a informação de estado nos encaminhadores, e é enviada pelo candidato a novo membro até ao primeiro nó da árvore e um novo tipo de mensagem, a mensagem *tree*, que é então enviado como reposta a uma mensagem de *join* pelo primeiro nó da árvore rumo ao candidato a novo membro. É esta mensagem, a mensagem *tree*, que altera as tabelas de encaminhamento (neste caso a MCT e a MFT) de forma a implementar a construção da árvore de distribuição *multicast*.

Este protocolo por si só não implementa o encaminhamento *multicast* tendo em conta os requisitos de QoS das aplicações, mas pode constituir um bom candidato por implementar a construção das árvores no sentido directo (ou seja, da fonte até aos diferentes destinos) em oposição ao sentido inverso, como a maioria dos protocolos de encaminhamento *multicast*, adequados ao paradigma do melhor esforço. Além disso, tem a vantagem de permitir o desenvolvimento progressivo da infra-estrutura *multicast* graças ao esquema de endereçamento e re-envio de tráfego utilizados. A utilização de endereçamento *unicast* faz com que o protocolo seja capaz de suportar encaminhadores exclusivamente *unicast* na árvore de distribuição. Apesar disso, o REUNITE apresenta alguns problemas quando usado em redes assimétricas. Potencialmente constrói as árvores de distribuição de forma directa, uma vez que a árvore é construída através da acção de uma mensagem (*tree* que é enviada pela fonte em direcção aos membros do grupo. A mensagem de *join* não introduz informação de estado nos encaminhadores, a menos que passe por algum que já pertença à árvore de distribuição. Nesses casos, a mensagem de *join* não continua o seu caminho e é enviada uma *tree message* a partir desse nó em direcção ao novo membro com o objectivo de criar um novo ramo. Neste caso o caminho entre a fonte e o novo membro pode não ser o menor caminho possível, logo a árvore de distribuição construída pelo REUNITE pode não ser a árvore dos caminhos mais curtos embora seja, de facto criada de forma directa e não invertida.

Luís Costa *et al* propuseram um protocolo designado por *Hop-by-Hop Routing Protocol*, HBHP[63], que é basicamente uma extensão do protocolo REUNITE proposto por Ion Stoica. São usadas na mesma as duas tabelas MCT e MFT mas no HBHP as mensagens de *join* são sempre re-enviadas até à fonte a menos que seja encontrado um nó da árvore donde já esteja a sair um ramo especificamente destinado ao receptor que fez o *join*. Desta forma o HBHP consegue construir uma árvore de caminhos mais curtos por cada fonte.

3.3.2 Discussão

Todos os protocolos que se baseiam na estratégia de *path probing* descritos nas secções 3.3.1.4, 3.3.1.5 e 3.3.1.6, têm alguns problemas em comum. O mais significativo é o facto de todos eles recorrerem *flooding* de mensagens de controle com o objectivo de procurarem um caminho que satisfaça os requisitos.

O *flooding* introduz uma grande quantidade de mensagens de controle na rede que pode prejudicar de forma significativa o seu desempenho. A ideia das diferente propostas é precisamente aliviar de alguma forma esta sobrecarga.

Outro problema que algumas destas propostas partilham é o facto da procura do melhor caminho ser feita no sentido que vai desde o candidato a novo membro até a um nó que já esteja na árvore. Esse tipo de estratégia não parece ser adequada a redes assimétricas (redes em que as ligações tem custos diferentes em ambos os sentidos).

Além disso a maioria destas propostas só procura resolver o problema de encontrar o melhor caminho que satisfaz os requisitos de QoS desde o candidato a novo membro até ao primeiro nó da árvore. Nada é feito no sentido de verificar se os requisitos disponibilizados pela árvore já existente satisfazem ou não os requisitos de QoS do novo membro. Ou seja, este tipo de soluções não suporta grupos heterógeneas, ou seja, parte do principio que todos os membros do grupo tem os mesmos requisitos de QoS.

Por outro lado, as estratégias baseadas em *Path Probing* têm a grande vantagem de não exigirem que os nós da topologia mantenham o estado global da rede. Á semelhança do que acontece no caso do encaminhamento *unicast*, esta estratégia aplicada ao encaminhamento *multicast* apenas exige que os nós da topologia mantenham informação de estado local, ou seja, cada nó mantém informação de estado relativo às suas próprias ligações.

Do lado oposto estão as estratégias que exigem que todos os nós mantenham o estado global da rede (sejam centralizadas, sejam distribuídas). Como já foi referido na discussão do encaminhamento *unicast* este pressuposto traz alguns problemas adicionais quando se trata de encaminhamento com QoS, nomeadamente, a rápida desactualização e a imprecisão desta informação de estado.

Capítulo 4

Implementação do PIM-SM no NS

Todos os protocolos de encaminhamento (*unicast* e *multicast*), desenvolvidos no âmbito deste trabalho, foram implementados e testados com recurso à simulação. O simulador escolhido foi o Network Simulator (NS-2)[5].

Cronologicamente, o primeiro protocolo implementado no contexto deste trabalho foi o protocolo PIM-SM (Protocol Independent Multicast-Sparse Mode)[7]. Esta implementação foi usada, não só como ponto de partida para a implementação de outros protocolos de encaminhamento *multicast*, mas também como referencial para comparação dos resultados obtidos.

Neste capítulo descreve-se a implementação do protocolo PIM-SM desenvolvida para o NS. Por ser o primeiro capítulo em que é descrita uma implementação de um protocolo de encaminhamento no NS, não podemos deixar de introduzir alguns aspectos relacionados com o Simulador, nomeadamente a forma como são implementados neste ambiente os protocolos encaminhamento *unicast* e *multicast*.

Assim, e depois de uma descrição genérica do protocolo PIM-SM (secção 4.1), dedica-se uma importante secção deste capitulo (secção 4.2) à forma como o NS suporta os protocolos de encaminhamento *unicast* e *multicast*. Só depois é feita a descrição da implementação do PIM-SM no NS (secção 4.3). Além da estratégia de encaminhamento, que no caso do *multicast* passa pela forma como se constroem e mantém as árvores de distribuição de tráfego *multicast*, o encaminhamento *multicast* envolve também a actividade de envio de tráfego *multicast* em cada nó. A secção 4.3.2 explica as alterações tiveram que ser introduzidas para o re-envio dos pacotes em cada nó ser efectuado de acordo com o protocolo PIM-SM. Por fim, aborda-se a questão da obtenção de resultados da simulação e a sua respectiva análise (secção 4.4).

4.1 Descrição do Funcionamento do Protocolo PIM-SM

O PIM-SM[7] é um protocolo de encaminhamento *multicast* concebido para redes alargadas. Ao contrário de outros protocolos de encaminhamento *multicast* não depende de nenhum protocolo de encaminhamento *unicast* em particular (daí a designação de independente do protocolo). Além disso suporta os dois tipos de árvores de distribuição *multicast*: árvores partilhadas e árvores centradas nas fontes, o que lhe confere uma grande flexibilidade.

O PIM-SM define um ponto de encontro (Rendez-Vous Point, adiante designado por RP) que é usado como ponto de registo para facilitar o encaminhamento de pacotes. Quando uma fonte pretende começar a enviar dados, é ao encaminhador que está mais próximo da origem que compete enviá-los para o RP. Por outro lado, quando um sistema pretende receber dados, é o encaminhador mais próximo do destino que deve registar-se junto do RP para os poder receber. O fluxo de dados flui assim da origem para o RP e deste para o destino. A árvore de distribuição do tráfego *multicast* é construída, numa primeira fase, do RP para os vários membros do grupo (destinatários) e é partilhada pelas várias fontes que mais não têm do que fazer chegar o tráfego *multicast* até ao RP. Numa segunda fase é possível a cada membro do grupo comutar para árvores centradas nas fontes, se assim o desejarem.

No PIM-SM as árvores de distribuição *multicast* são construídas e mantidas através de pedidos explícitos de junção (*join*) e abandono (*prune*) dos vários membros do grupo. Um novo receptor que deseje juntar-se a um grupo *multicast* G, informa o encaminhador mais próximo (adiante designado por encaminhador nomeado) desse facto através do protocolo IGMP (Internet Group Management Protocol)[64]. Esse encaminhador reage ao pedido de admissão de um membro criando de imediato uma entrada (*,G) na tabela de encaminhamento. Uma vez criada a entrada, o encaminhador envia uma mensagem do tipo *join* ao próximo encaminhador na direcção do RP.

Cada encaminhador no caminho até ao RP deve criar ou simplesmente actualizar uma entrada (*,G) na sua tabela de encaminhamento sempre que receber uma mensagem de *join*. Sempre que a entrada (*,G) é criada de novo, a mensagem de *join* deve ser reenviada ao próximo nó no caminho mais curto até ao RP.

Quando o tráfego de dados proveniente de uma mesma fonte ultrapassa um determinado limite, um receptor pode, se assim o desejar, enviar um pedido de junção à fonte, iniciando assim a construção de um ramo de uma árvore centrada nessa fonte. Para isso, o encaminhador nomeado (com membros directamente ligados) coloca uma entrada (S,G) na sua tabela de encaminhamento e envia uma mensagem *join* ao próximo encaminhador, na direcção da fonte. Todos os encaminhadores no trajecto até à fonte devem criar ou actualizar as entradas (S,G) em conformidade.

Um encaminhador ao receber o primeiro pacote proveniente da fonte através da árvore centrada na fonte, faz um *prune* daquela fonte na árvore partilhada, para garantir que não recebe pacotes duplicados da fonte, via árvore centrada na fonte e via árvore partilhada.

Este *prune* só deverá ser feito se o nó pertencer à árvore partilhada.

Para cada grupo *multicast*, podem coexistir árvores *multicast* centradas nas fontes, identificadas por pares (S,G), e uma árvore partilhada identificada por (*,G).

4.1.1 Construção da árvore partilhada

As árvores partilhadas são mantidas por entradas do tipo (*,G), entradas essas que são criadas quando um encaminhador recebe uma mensagem de *join* para um grupo G que ainda não consta da tabela de encaminhamento.

As entradas (*,G) são constituídas pelos seguintes dados:

- endereço do Grupo (G);
- endereço da fonte (*); Quando este campo está preenchido com um "*" significa que se trata de uma entrada (*,G).
- interface de entrada (iif = interface usada para atingir o RP pelo melhor caminho).
- lista das interfaces de saída (oiflist = interface que vai ser usada para atingir o sistema terminal que requereu a junção e que é a interface por onde chegou a mensagem de *join*).
- *flags* que caracterizam o tipo de entrada:
  - RPT-Bit (1); Quando está activa, esta *flag* indica que esta entrada corresponde a informação de estado relativa à árvore partilhada.
  - SPT-Bit (0); Numa entrada (*,G) esta *flag* não tem significado.

Uma vez criada a entrada, o encaminhador nomeado envia uma mensagem de *join* ao próximo encaminhador na direcção do RP com os seguintes dados:

- tipo de mensagem (join); Este campo especifica de que tipo de mensagem se trata: *join* ou *prune*.
- endereço do Grupo (G);
- lista de join (endereço do RP);
- lista de prune (nulo);
- WC-Bit (1); Quando está activa (ou seja, com o valor 1) esta *flag* que o encaminhador pretende receber o tráfego proveniente de todas as fontes registadas para o grupo G.
- RPT-Bit (1); Quando está activa, esta *flag* indica que se trata de um pedido de junção à árvore partilhada.

Para além destas duas acções, o encaminhador nomeado deve ainda incluir o endereço do RP na lista de notificações *join/prune* a serem efectuadas regularmente.

Quando não existirem membros directamente ligados, nem nenhum interface de saída para outras redes na lista de interfaces de saída, a entrada (*,G) deve ser apagada, bem como o endereço do RP da lista de notificações periódicas.

Cada encaminhador no caminho até ao RP deve criar ou simplesmente actualizar uma entrada (*,G) na sua tabela de encaminhamento sempre que receber uma mensagem de *join*. Sempre que a entrada (*,G) é criada de novo, a mensagem de *join* deve ser reenviada ao próximo nó no caminho mais curto até ao RP.

4.1.2 Utilização da árvore partilhada

Quando uma fonte deseja começar a enviar tráfego para um grupo, inicialmente deve fazê-lo através do RP do grupo. A fonte "encapsula" cada pacote de dados numa mensagem do tipo *register* e envia essa mensagem por *unicast* até ao RP. O RP "desencapsula" a mensagem e re-envia o pacote de dados através da árvore partilhada a todos os membros do grupo.

Se o tráfego proveniente de determinada fonte ultrapassar uma determinada taxa, o RP pode efectuar um pedido de junção à fonte. Para isso tem apenas que criar uma entrada do tipo (S,G) para a fonte em causa e enviar uma mensagem de *join* até à fonte de forma a que a informação de estado nos encaminhadores no caminho entre a fonte e o RP seja actualizada. Quando o RP começar a receber pacotes provenientes da fonte por *multicast* envia uma mensagem do tipo *register stop* dirigida à fonte. É apenas quando recebe esta mensagem que a fonte deixa de "encapsular" os pacotes e passa a enviá-los apenas por *multicast*.

4.1.3 Construção da árvore centrada na fonte

Um encaminhador, que tenha membros de um grupo directamente ligados, começa sempre por se juntar a uma árvore de distribuição partilhada e centrada num RP, podendo depois comutar para árvores centradas nas fontes quando começa a receber dados dessas fontes através da árvore partilhada.

Recomenda-se que a troca de árvore só ocorra após um período de recepção de pacotes que seja significativo, podendo haver um débito pré-definido como patamar para o início do processo. Algumas implementações - como é o caso da implementação da CISCO - colocam esse patamar a 0, significando com isto que a troca de árvore ocorre imediatamente após a recepção dos primeiros dados.

Quando decide comutar de árvore, o encaminhador nomeado (com membros directamente ligados) coloca uma entrada (S,G) na sua tabela com os seguintes dados:

- endereço do Grupo (G);

- endereço da fonte (S);
- interface de entrada (iif = interface usada para atingir a fonte S pelo melhor caminho).
- lista das interfaces de saída (oiflist = oiflist da entrada (*,G) eliminando a iif caso ela faça parte da lista)
- *flags* que caracterizam o tipo de entrada:
  - RPT-Bit (0); Significa que esta entrada não diz respeito à árvore partilhada.
  - SPT-Bit (0); Significa que a entrada ainda não está activa.

Uma vez criada a entrada, o encaminhador nomeado envia uma mensagem *join* ao próximo encaminhador na direcção à fonte com os seguintes dados:

- tipo de mensagem (join); Este campo especifica de que tipo de mensagem se trata: *join* ou *prune*.
- endereço do Grupo (G);
- lista de join (endereço da fonte S);
- lista de prune (nulo);
- WC-Bit (0);
- RPT-Bit (0);

Para além destas duas acções, o encaminhador deve ainda incluir o endereço da fonte na lista de notificações *join/prune* a serem efectuadas regularmente:

Todos os encaminhadores no trajecto até à fonte devem criar ou actualizar as entradas (S,G) em conformidade.

4.1.4 Utilização da árvore centrada na fonte

Quando um determinado encaminhador decide comutar para uma árvore centrada numa fonte são criadas entradas (S,G) com o SPT-Bit a zero em todos os encaminhadores no caminho entre o nó destinatário e a fonte. Isto significa que apesar de já existir informação de estado relativa à árvore centrada na fonte, as entradas ainda permanecem inactivas. O facto do SPT-Bit estar a zero quer dizer que o envio de pacotes deve continuar a ser feito através das entradas (*,G) se elas existirem.

Um encaminhador ao receber o primeiro pacote proveniente da fonte através da árvore centrada na fonte (comprova isso testando a interface por onde chegou o pacote, que terá que coincidir com a interface usada para chegar ao nó vizinho no caminho mais curto até

à fonte), põe o SPT-Bit a 1. Além disso faz um *prune* daquela fonte na árvore partilhada, para garantir que não recebe pacotes duplicados da fonte, via árvore centrada na fonte e via árvore partilhada. Este *prune* só deverá ser feito se o nó pertencer à árvore partilhada.

A partir do momento em que a entrada (S,G) tem o SPT-Bit a 1 deixam de ser re-enviados os pacotes provenientes da fonte S, que chegarem via árvore partilhada.

4.1.5 Destruição de árvores no PIM-SM

A destruição das árvores de distribuição *multicast* ou simplesmente de alguns dos seus ramos é o resultado de pedidos de abandono explícitos por parte dos membros do grupo.

Um pedido de abandono de um membro implica a saída respectiva da árvore partilhada, ou seja, a "poda" de todos os ramos que estão na árvore partilhada com o único propósito de fazer chegar o tráfego *multicast* a este membro. Se durante a sessão, o membro comutou para uma ou mais árvores centradas nas fontes então torna-se também necessário "podar" os ramos dessas árvores que ligam o membro do grupo às respectivas fontes.

Um pedido de abandono dá origem a um mensagem de *prune* na árvore partilhada e uma mensagem de *prune* por cada árvore centrada na fonte à qual o membro do grupo se tenha juntado.

Um encaminhador ao receber uma mensagem deste tipo começa por apagar da lista de interfaces de saída da respectiva entrada ((*,G) ou (S,G), conforme se trate de um *prune* na árvore partilhada ou de um *prune* numa árvore centrada numa fonte), a interface por onde chegou a mensagem de *prune*. Só depois disso, caso a lista de interface da saída tenha ficado vazia, é que re-envia a mensagem de *prune*. Se se tratar de um *prune* na árvore partilhada a mensagem de *prune* deverá ser re-enviada em direcção ao RP, se se tratar de um *prune* numa árvore centrada numa fonte, a mensagem deverá ser re-enviada em direcção à fonte. Além disso, as entradas deverão ser apagadas, o que alivia os encaminhadores de informação de estado desnecessária, e destrói definitivamente os ramos das árvores *multicast*.

As mensagens de *prune* podem também ser originadas na situação em que os nós comutam da árvore partilhada para árvores centradas na fontes. Como já foi referido nessas situações é necessário evitar que os pacotes que já estão a ser transmitidos por árvores centradas nas fontes continuem a ser difundidos através da árvore partilhada. Para isso utilizam-se também mensagens de *prune*, se bem que especiais, são mensagens de *prune* de fontes na árvore partilhada e distinguem-se dos outros *prunes* de fontes por terem a *flag* RPT-Bit activada. É esta *flag* que indica aos diferentes encaminhadores que se trata de um *prune* de uma fonte da árvore partilhada, e desta forma, se for necessário re-enviar a mensagem de *prune* esse re-envio é feito em direcção ao RP e não em direcção à fonte.

As mensagens de *prune* são constituídas pelos seguintes campos:

- tipo de mensagem (prune); Este campo especifica de que tipo de mensagem se trata,

neste caso deverá ser preenchido com *prune*.

- endereço do Grupo (G);
- lista de join (nulo);
- lista de prune; Se se tratar de um *prune* de uma fonte na árvore partilhada, ou de um *prune* numa árvore centrada na fonte este campo deverá ser preenchido com a fonte em causa. Senão, caso se trate de um *prune* na árvore partilhada, deverá conter o endereço do RP.
- WC-Bit e o RPT-Bit são preenchidos a 0 ou a 1 de acordo com as seguintes condições:
  - WC-Bit(1), RPT-Bit(1) caso se trate de um *prune* na árvore partilhada
  - WC-Bit(0), RPT-Bit(0) caso se trate de um *prune* numa árvore centrada na fonte
  - WC-Bit(0), RPT-Bit(0) caso se trata do *prune* de uma fonte na árvore partilhada.

4.1.6 Informação de estado mantida nos nós

Para cada grupo *multicast*, podem coexistir árvores *multicast* centradas nas fontes, identificadas por pares (S,G), e uma árvore partilhada identificada por (*,G). Para cada uma destas árvores, o estado a manter é o seguinte:

- *S* - endereço da fonte ou "*" se se trata da árvore partilhada
- *G* - endereço do grupo
- *iif* - interface de entrada por onde os pacotes de dados são esperados (é preciso verificar este campo antes de os encaminhar, para evitar ciclos!)
- *oif_list* - lista de todos os interfaces de saída por onde os pacotes de dados devem ser encaminhados
- *Flags* - *flags* de caracterização da rota, nomeadamente o SPT-Bit e o RPT-Bit.
- *Timer* - temporizador de acções de *join*

O encaminhamento *multicast* é feito com base nesta informação. A informação de estado é produzida pela acção da estratégia de encaminhamento descrita nas secções anteriores. Na secção seguinte descreve-se o uso que o PIM-SM faz dessa informação para proceder ao encaminhamento de pacotes *multicast*.

4.1.7 Re-envio dos pacotes de dados em cada nó

De acordo com o PIM-SM, os pacotes de dados são encaminhados de uma forma semelhante a outras propostas *multicast*. O encaminhador começa por extrair o endereço da fonte (origem) e o endereço do grupo (destino) do pacote de dados, e procura na sua tabela de encaminhamento *multicast* a mais longa entrada correspondente a esse par de valores.

Se existir uma entrada (S,G) na tabela, ela será portanto encontrada em primeiro lugar. Só no caso de não existir é que o encaminhador procura uma entrada (*,G). Caso não exista nenhuma das entradas (rotas *multicast*) o pacote é simplesmente descartado. Se existir, o encaminhador faz ainda uma ultima verificação sobre a interface de entrada do pacote de dados, por forma a verificar se ela corresponde à interface de entrada registada na tabela de encaminhamento (verificação RPF). Se não coincidir, o pacote é descartado. Se coincidir o pacote é encaminhado para todas as interfaces de saída enumeradas na rota.

No entanto, torna-se necessário efectuar algumas acções especiais para evitar que se percam pacotes de dados durante a comutação da árvore partilhada para uma árvore centrada na fonte. Essas acções, que se enumeram de seguida, ocorrem apenas quando é encontrada uma rota (S,G) na tabela de encaminhamento:

- Se o SPT-Bit da rota está activado (1), então:
  - Se a interface de entrada do pacote de dados corresponde à *iif* referida na rota, deve-se encaminhar o pacote para todas as interfaces de saída de (S,G)
  - Se a interface de entrada é diferente da *iif* de (S,G), a solução é descartar o pacote.
- Se o SPT-Bit da rota está desactivado (0), então:
  - se a interface de entrada do pacote de dados corresponde à *iif* referida na rota, deve-se então encaminhar o pacote para todas as interfaces de saída de (S,G). Adicionalmente, o SPT-Bit para essa entrada deverá ser activado desde que a interface de entrada seja diferente da *iif* da entrada (*,G).
  - se a interface de entrada é diferente da constante da rota (S,G), então verifica-se se a referido interface é igual à que consta na rota (*,G). Se a interface de entrada corresponder, o pacote será encaminhado para todos os interfaces de saída correspondentes.
  - se a interface de entrada não coincidir com nenhuma *iif* de nenhuma entrada para o grupo G, o pacote é descartado.

Os pacotes de dados nunca disparam *prunes* directamente, mas podem disparar acções que por sua vez conduzem a *prunes*.

4.2 O Network Simulator

O NS é um ambiente de simulação de eventos discretos que tem sido largamente utilizado pela comunidade científica para a simulação, estudo e desenvolvimento de protocolos de comunicação. Este ambiente inclui já suporte para a simulação de protocolos dos níveis de rede e transporte (por exemplo, o IP, o TCP e o UDP), bem como para diversos protocolos de encaminhamento e mesmo para aplicações (como por exemplo o HTTP e o FTP). Para além do desenvolvimento inicial na Universidade da Califórnia, em Berkeley, este simulador é o resultado do esforço contínuo da comunidade científica a nível mundial que tem vindo a fornecer diversas contribuições.

O NS está escrito em duas linguagens orientadas a objectos: o C++ e o OTcl. As tarefas muito frequentes (e que por isso exigem rapidez) são normalmente escritas em C++, enquanto que acções de controlo, que exigem flexibilidade e facilidade de interacção, são implementadas em OTcl. O NS inclui assim duas hierarquias de classes que se podem usar conforme o que se julgar mais conveniente. Uma classe numa das hierarquias tem normalmente uma classe equivalente na outra hierarquia. Uma instância de uma dessas classes reúne os métodos e as variáveis definidos em ambas as linguagens.

A interface com o utilizador é da responsabilidade de um interpretador de OTcl. As simulações são descritas em *scripts* OTcl que são interpretadas e executadas por este interpretador.

4.2.1 Encaminhamento Unicast no NS

A figura 4.1 mostra como é feito o encaminhamento de pacotes num nó que implementa o encaminhamento *unicast* no NS. O nó é constituído por um conjunto de "classificadores" por onde os pacotes vão passando e sendo "classificados" por forma a chegarem ao destino desejado. A classificação ocorre normalmente por endereço/porta de destino do pacote. Os destinos podem ser agentes que recebem os pacotes destinados ao próprio nó (endereço de destino = endereço do nó), ou ligações (*links*) para outros nós. Os endereços dos nós e os números de porta são inteiros de 32 bits [1].

O ponto de entrada no nó é identificado pela variável **entry_**, e pode ser obtido invocando o método **$node entry**. Este é o primeiro elemento que manipula os pacotes que chegam ao nó. No caso dos nós que implementam apenas o encaminhamento *unicast*, todos os pacotes são entregues a um classificador (um **Classifier/Hash/Dest** referenciado pela variável **classifier_**), que separa os pacotes de acordo com o endereço de destino, entregando os dirigidos ao próprio nó a um classificador de portas **Classifier/Port** referenciado pela variável **dmux_** , e os dirigidos a outros nós, às ligações respectivas.

[1] A partir da versão 2.1b6 do NS, tanto os endereços como as portas passaram a ser inteiros de 32 bits, acabando com a restrição das versões anteriores que impunha um único inteiro de 16 bits para identificar nó e porta (8 para endereço, 8 para porta). Com esta mudança, as topologias passaram a poder ter mais do que 256 nós, sem necessidade de comandos de expansão de endereços.

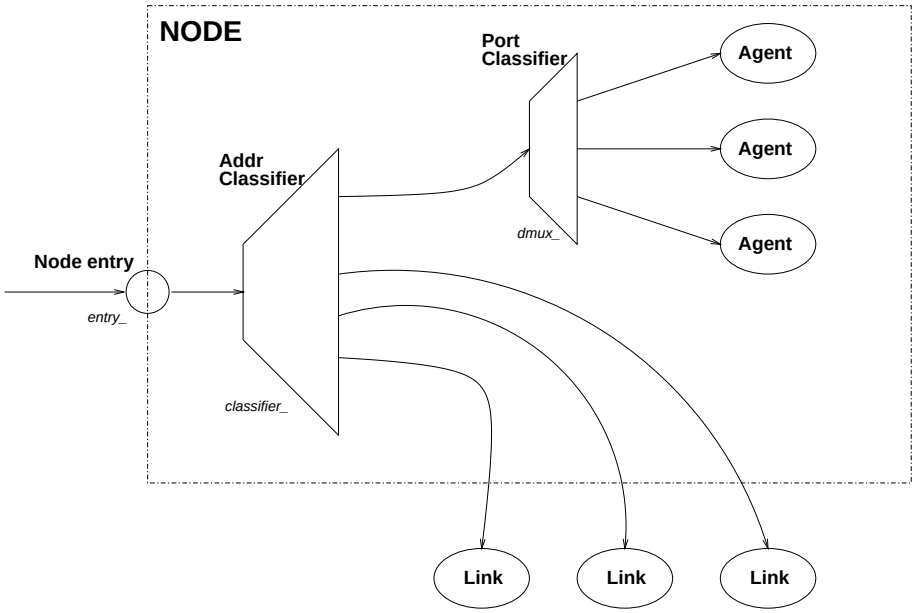


Figura 4.1: Encaminhamento unicast num nó

O classificador referenciado por **classifier_** guarda a tabela de encaminhamento *unicast* do nó. Cabe aos módulos que implementam os protocolos de encaminhamento, inserir e remover rotas neste classificador. A este fica a tarefa de armazenar as rotas em estruturas de fácil pesquisa, e de as consultar aquando da recepção de um novo pacote de dados, encaminhando-o de acordo com a informação aí constante.

É possível extender as funcionalidades de um nó acrescentando Módulos de Encaminhamento (*Routing Modules*) aos nós. Um Módulo de Encaminhamento é constituído por:

- Um agente de encaminhamento capaz de trocar informação com os seus pares (vizinhos);
- Uma lógica de encaminhamento que usa a informação recolhida pelos agentes de encaminhamento para calcular os vários caminhos;
- Um ou mais *classifiers* que residem dentro dos nós e que usam as rotas calculadas para processar o re-envio de pacotes.

É de notar que quando se desenvolve um novo protocolo de encaminhamento nem sempre é necessário desenvolver estas três componentes. Por exemplo, no caso do protocolo *unicast* LS (implementação integrada no NS de um protocolo de encaminhamento *unicast* de estado de ligação) apenas foi necessário desenvolver o agente de encaminhamento para que a informação de encaminhamento fosse trocada como ditam os protocolos de estado

de ligação e uma nova lógica de encaminhamento que basicamente implementa o algoritmo Dijkstra sobre a base de dados topológica mantida pelos agentes de encaminhamento em cada um dos nós. De resto, usa os mesmos classificadores que os outros protocolos de encaminhamento *unicast*. No entanto quando são criados novos classificadores é preferível criar um objecto designado por **RtModule** que gere estes três componentes de forma integrada.

No NS existem já implementados alguns módulos de encaminhamento, entre os quais se salientam os seguintes:

- o **RtModule/Base**: implementa todas as funcionalidades básicas comuns aos protocolos de encaminhamento *unicast*, por exemplo adiciona rota, remove rota, associa agente, dissocia agente, etc.
- o **RtModule/Mcast**: num futuro próximo deverá implementar as funcionalidades comuns aos protocolos de encaminhamento *multicast*. Na versão do NS utilizada este módulo limita-se a estabelecer os classificadores *multicast*.
- o **RtModule/Hier**: implementa o encaminhamento hierárquico.
- o **RtModule/Manual**: implementa o encaminhamento estático.

O módulo **RtModule/Base** é executado por defeito em todos os nós. Existe uma variável (de classe) da classe **Node**, designada por **module_list**, que contém todos os módulos que deverão ser suportados pelos nós. Esta variável é inicializada com a *string* "Base". Quando um novo nó é criado a variável **module_list** é percorrida para determinar que extensões se devem adicionar ao nó. Se só constar da lista a *string* "Base" o nó que é criado é do tipo do que foi mostrado na figura 4.1 e suporta apenas o encaminhamento *unicast*.

No NS, o encaminhamento *unicast* é implementado usando basicamente quatro classes principais. São elas:

- A classe **RouteLogic**. Esta classe representa basicamente a tabela de encaminhamento que é criada e mantida centralmente para qualquer simulação que use encaminhamento *unicast*. Inclui dois métodos para configurar o encaminhamento *unicast*: o método **register** que é invocado pelo método **rtproto** da classe **Simulator**, e o método **configure** que é invocado pelo método **run** também da classe **Simulator**.

  Além destes dois métodos, a classe **RouteLogic** tem o método **lookup** que recebe como argumento dois nós e devolve um terceiro nó que representa o próximo salto (*nexthop*) usado pelo nó que entrou como primeiro argumento para atingir o nó que entrou como segundo argumento.
- A classe **rtObject**. Esta classe é usada sempre que é usado encaminhamento dinâmico. Cada nó tem uma instância desta classe, referenciada pela variável **rtObject_**, que actua como coordenador entre os vários protocolos de encaminhamento

que estão a ser executados pelo nó. É o **rtObject** que calcula e instala na tabela de encaminhamento de cada nó as "melhores" rotas para todos os destinos possíveis. Para isso tem que analisar as rotas escolhidas por cada um dos agentes de encaminhamento instalados no nó e entre as várias disponíveis escolher a melhor. Sempre que há alterações na topologia ou no estado da rede é o **rtObject** que notifica os vários agentes de encaminhamento instalados nos nós para que ajam em conformidade.

- A classe **rtPeer**. Esta classe é usada pelos agentes de encaminhamento para armazenarem informação acerca dos seus pares. Além de armazenarem os endereços dos seus pares, os agentes de encaminhamento armazenam também a métrica e preferência de cada rota anunciada por cada um dos pares.

- A classe **Agent/rtProto**. Esta classe é a classe base a partir da qual são derivados todos os agentes de encaminhamento. Cada agente de encaminhamento deve definir um procedimento **init-all** para inicializar o funcionamento do protocolo. Adicionalmente, se o protocolo tiver um comportamento dinâmico e souber reagir a alterações da topologia, deverá definir o procedimento **compute-all** e **intf-changed**.

Além destas classes o encaminhamento *unicast* no NS é implementado recorrendo a extensões às classes **Simulator**, **Node**, **Link** e **Classifier**. Descrevem-se a seguir, muito sucintamente, as alterações realizadas em cada uma destas classes:

- Classe **Simulator**: foram acrescentados os métodos **rtproto**, e **get-routelogic**. O método **rtproto** estabelece quais os protocolos de encaminhamento *unicast* que se pretende que cada nó seja capaz de suportar. Recebe como argumento a designação de um protocolo (por exemplo, "LS" ou "DV") e uma lista de nós e inicializa o protocolo de encaminhamento em cada um dos nós que constitui a lista de nós. O método **get-routelogic** retorna um "apontador" para a tabela de encaminhamento global que é mantida centralmente.

- Classe **Node**: foram acrescentados os métodos **init-routing**, **add-routes**, **delete-routes** e **rtObject?**. O método **init-routing** é invocado em cada nó pelo **rtObject**. Este método guarda um apontador para o **rtObject** do nó numa variável de instância designada por **rtObject_**. Os métodos **add-routes** e **delete-routes** acrescentam e apagam, respectivamente, rotas para um determinado destino. O método **rtObject?** devolve o **rtObject** de um determinado nó.

- Classe **Link**: foram acrescentados métodos para suportar o custo das ligações. Todas elas têm agora uma variável de instância, designada por **cost_** e os métodos **cost** e **cost?** são utilizados para atribuir um valor a essa variável e ler o valor que ela tem associado, respectivamente.

- Classe **Classifier**: foram re-escritos e adaptados os métodos **install** e **installNext** que instalam novas entradas na tabela de *hashing* do classificador e foi acrescentado o método **adjacents** que devolve uma lista de pares com todos os elementos instalados no classificador.

4.2.1.1 Início do processo de encaminhamento unicast na simulação

No NS a estratégia de encaminhamento é especificada na *script* de simulação através do método **rtproto**. Este é um método da classe **Simulator**. Por exemplo, para se criar o agente de encaminhamento que implementa o protocolo LS em cada um dos nós da topologia é necessário incluir na *script* de simulação a seguinte linha:

```
$ns rtproto LS
```

O método **rtproto** ao ser executado com apenas um argumento: a *string* "LS", invoca o método **register** na variável **routingTable_**[2].

O método **register** da classe **RouteLogic** constrói uma variável de instância **rtprotos_** que não passa de um *array* em que o índice é o nome do protocolo (neste caso LS) e o valor a lista dos nós que executam esse protocolo.

Mais tarde, já no método **run** da classe **Simulator** é invocado o método **configure** da classe **RouteLogic**. Este método percorre a variável **rtprotos_** e para cada um dos protocolos que aparecem no índice deste *array* executa as inicializações necessárias. Neste caso, invoca o método **init-all** da classe **Agent/rtProto/LS**. Este é um método de classe que basicamente cria os **rtObjects** e os agentes de encaminhamento em cada um dos nós

4.2.1.2 Estratégias de encaminhamento unicast existentes no NS

Nas estratégias de encaminhamento dinâmico, os diferentes nós enviam e recebem mensagens e calculam as rotas com base nas mensagem que trocam entre si. A distribuição do NS usada inclui uma implementação de um protocolo de encaminhamento baseado em vectores de distância (o **rtProtoDV**) e de um outro de estado de ligação (o **rtProtoLS**).

4.2.1.3 O protocolo DV

O **rtProtoDV** é uma implementação para o NS de um protocolo de vectores de distância que calcula dinamicamente as rotas para todos os destinos possíveis de forma distribuída. O procedimento **init-all** cria uma instância da classe **rtObject** e um agente de encaminhamento do protocolo DV em cada um dos nós.

O construtor do agente de encaminhamento inicializa uma série de variáveis. Cada agente de encaminhamento mantém *arrays* indexados por nó destino onde mantém a métrica, a preferência e o próximo salto relativos à melhor rota encontrada para cada destino possível. Alem disso mantém uma lista com todos os seus pares. Cada par é uma instância da classe **rtPeer** e mantém um conjunto de *arrays* indexados por nó destino com a métrica e preferência de cada rota aprendida através dele.

[2] A variável **routingTable_** é uma variável do **Simulator** da classe **RouteLogic**

O procedimento **send-periodic-updates** invoca o procedimento **send-updates** com uma determinada periodicidade configurável. O procedimento **send-updates** é o procedimento responsável por enviar as actualizações para um conjunto seleccionado de pares. Além das actualizações periódicas, este procedimento é invocado sempre que se verificam mudanças nas rotas calculadas pelo nó.

Quando um nó recebe uma actualização, verifica se é diferente da última recebida do pelo par, e caso seja invoca o procedimento **compute-routes** para provocar o cálculo de novas rotas.

4.2.1.4 O protocolo LS

O **rtProtoLS** é uma implementação para o NS de um protocolo de estado de ligação que calcula dinamicamente rotas de forma distribuída. Pode dizer-se que se trata de uma versão simplificada do protocolo OSPF uma vez que:

- Cada nó envia LSAs para os seus vizinhos sempre que há alterações no estado das suas ligações ou, caso não as haja, periodicamente de 30 em 30 minutos.
- Ao receber um LSA, um nó envia um *acknowledge* para a origem e re-envia o LSA recebido por sua vez a todos os seus vizinhos, com excepção do vizinho por onde foi recebido o LSA.
- Em cada um dos nós a base de dados topológica é constituída por todos os LSAs recebidos dos vários nós que constituem a topologia.
- Utilizando essas bases de dados topológicas, em cada um dos nós são calculados os melhores caminhos para todos os destinos usando o algoritmo Dijkstra.
- Quando não é recebido o *acknowledge* de alguns dos nós, o nó que enviou o LSA torna a enviá-lo ao fim de um determinado tempo.

O protocolo **rtProtoLS** é implementado no NS através da classe OTcl **LS** e tem uma classe correspondente em C++: a classe **rtProtoLS**. A classe **LS** é uma subclasse da classe **Agent/rtProto**.

4.2.2 Encaminhamento Multicast no NS

No caso do encaminhamento *multicast*, a estratégia de encaminhamento (responsável pela troca de mensagens de controle entre os nós de forma a construir as árvores *multicast*), está quase toda implementada em OTcl, enquanto que a componente que processa o envio de tráfego *multicast* em cada nó está implementada em C++.

A figura 4.2 mostra como é feito o encaminhamento de pacotes num qualquer nó *multicast* do simulador NS.

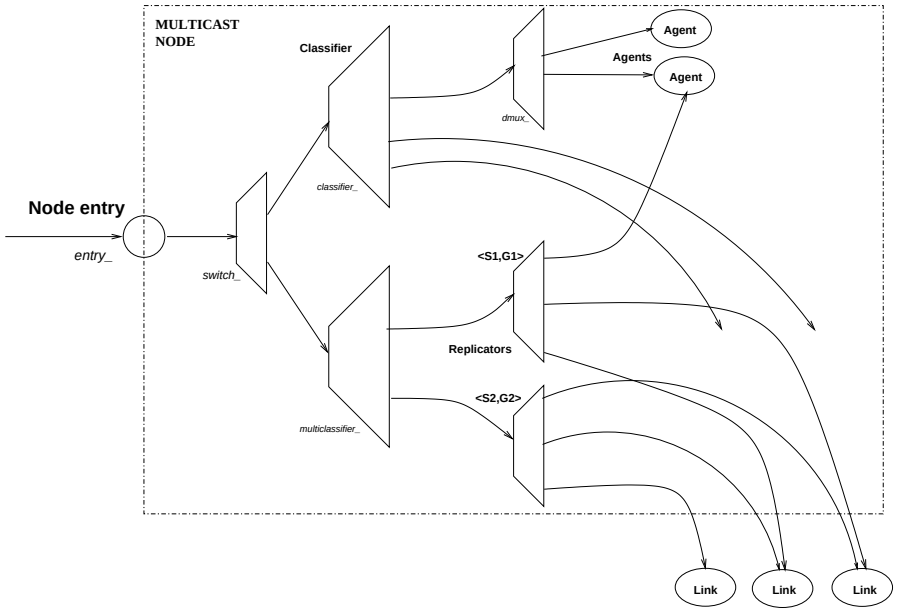


Figura 4.2: Encaminhamento multicast num nó

Os classificadores estão normalmente encadeados conforme ilustrado na figura 4.2. O ponto de entrada no nó é identificado pela variável **entry_**, e pode ser obtido invocando o método **$node entry**. Este é o primeiro classificador que manipula os pacotes que chegam ao nó. No caso dos nós *multicast*, este primeiro elemento é um **Classifier/Addr**, referenciado pela variável **switch_**, e que basicamente separa os pacotes *unicast* dos pacotes *multicast* com base no endereço destino. A heurística é muito simples: se o primeiro bit do endereço for 0, trata-se de um endereço *unicast*, se for 1 trata-se de um endereço *multicast*. Na prática o espaço de endereçamento é dividido a meio, quando a simulação envolve protocolos *multicast*.

Os pacotes classificados como *unicast* são depois entregues a um outro classificador (um **Classifier/Hash/Dest** referenciado pela variável **classifier_**), que os separa de acordo com o endereço de destino, entregando os dirigidos ao próprio nó a um classificador de portas **Classifier/Port** referenciado pela variável **dmux_**, e os dirigidos a outros nós ligações respectivas.

Quanto aos pacotes destinados a um determinado grupo *multicast* (G), passam então por um segundo classificador, do tipo **Classifier/Multicast**, referenciado pela variável **multiclassifier_**, que olha para o endereço de grupo (G) constante nos pacotes e procura encontrar entradas (S,G) (relativas a árvores centradas na fonte S) e/ou (*,G) (relativas a árvores partilhadas) associadas a esse endereço, encaminhando os pacotes por todos os interfaces de saída constantes nessa entrada. O trabalho de replicação por todos os interfaces é feito por um classificador-replicador, do tipo **Classifier/Replicator**, que se limita a manter uma lista de ligações ou agentes locais e entregar uma réplica dos pacotes

de dados a cada um deles.

Os classificadores referenciados por **classifier_** e **multiclassifier_**, guardam respectivamente a tabela de encaminhamento *unicast* e *multicast* do nó, bem como a lógica de encaminhamento associada.

Cabe aos módulos que implementam a estratégia de encaminhamento, inserir e remover rotas em cada um destes classificadores. A estes fica a tarefa de armazenar as rotas em estruturas de fácil pesquisa, e de as consultar a quando da recepção de um novo pacote de dados, encaminhando-os de acordo com a informação aí constante.

A tarefa de um classificador é examinar alguns campos de um pacote de dados, normalmente o endereço de destino, e mapear o valor desses campos num qualquer objecto de destino, que é o próximo destinatário do pacote de dados.

Os classificadores estão normalmente implementados em C++, porque as suas tarefas são muito frequentes (exigem rapidez) e porque são também muito bem determinadas e pouco sujeitas a acções de controlo.

O primeiro classificador da hierarquia de classificadores é o **Classifier** e basicamente implementa a funcionalidade de encaminhamento *unicast*. Tem basicamente três métodos nos quais baseia a sua actividade: **recv**, **find** e **classify**. O método **recv** é o primeiro a ser invocado sempre que um pacote chega ao classificador. Aliás o classificador recebe o pacote precisamente porque uma outra entidade lhe invocou o método **recv** com o pacote de dados como parâmetro.

O método **recv**, por sua vez, invoca o método **find** para descobrir qual o próximo destinatário do pacote de dados e invoca o método **recv** nesse destinatário, se ele não for nulo.

Quanto ao método **find**, começa por invocar o método **classify** para realmente classificar o pacote, localizando um *slot* na tabela de *hashing* que esteja de acordo com um campo do pacote de dados. Se encontrar não faz mais nada. Se não encontrar nenhum *slot*, verifica se existe um *default target* com uma indicação de um destinatário por omissão. Se não existir, invoca um método OTcl **no-slot** para que alguma decisão seja tomada, e tenta outra vez de acordo com o resultado devolvido pelo OTcl.

É o método **classify** que, com base numa função de sumariação aplicada ao endereço de destino do pacote, localiza uma entrada na tabela de encaminhamento e devolve o objecto associado a essa entrada.

O classificador *multicast* deriva do classificador *unicast* do qual herda algum do comportamento base, nomeadamente os métodos **find** e **recv**, só reescrevendo o método **classify**. Inclui duas novas tabelas de *hashing* (com cadeias nas colisões), uma para manter as entradas (S,G) e outra para manter as entradas (*,G). Implementa dois novos métodos auxiliares: **lookup** e **lookup_star** que procuram encontrar respectivamente entradas associadas (S,G) ou (*,G). Devolvem o objecto associado a essa entrada que normalmente é um replicador com uma lista de interfaces de saída.

Existe ainda um outro classificador *multicast*, usado na implementação BST (Bidirectional Shared Tree), que é quase idêntico ao classificador *multicast* standard, mas que reescreve ligeiramente o método **classify** e também o método **recv**.

A relação entre os diferentes classificadores está ilustrada na figura 4.3.

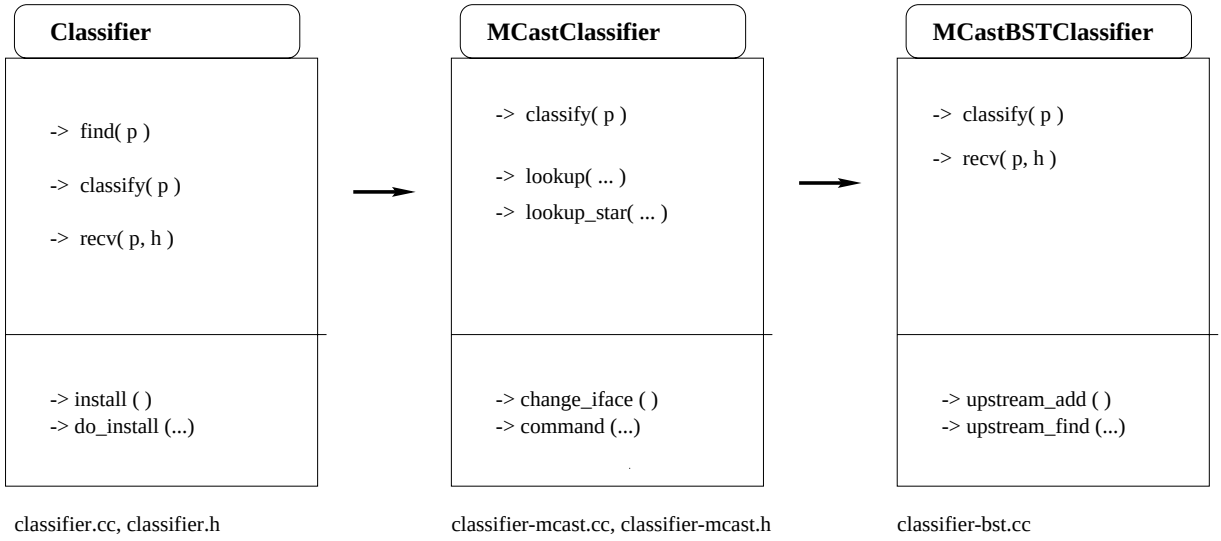


Figura 4.3: Classificadores unicast e multicast

4.2.2.1 Estratégias de encaminhamento multicast existentes no NS

A versão actual do simulador NS inclui quatro estratégias de encaminhamento *multicast*: centralizada (CtrMcast), modo-denso (DM), árvores-partilhadas (ST) e árvores-partilhadas-bidireccionais (BST), implementando cada delas um mecanismo distinto de cálculo das árvores de difusão com semelhanças óbvias com os principais protocolos de *multicast* actualmente existentes.

A estratégia centralizada utiliza um único agente de controlo (daí a sua designação) que calcula árvores de difusão centradas num qualquer nó da topologia e depois instala as entradas necessárias nas tabelas de encaminhamento de todos os nós seleccionados para fazerem parte da árvore. Podem-se assim construir árvores partilhadas, árvores centradas numa fonte, ou ambas, com os receptores a comutarem de umas árvores para as outras, com relativa facilidade. Esta estratégia tem muitas semelhanças com a estratégia PIM-SM, diferindo fundamentalmente na forma como se calcula e instala a informação de estado nos nós, pois no PIM-SM isso é feito à custa do envio explícito de mensagens de controlo *join* e *prune*. Esta diferença pode não ser aceitável em função da simulação e das medidas que se pretendam efectuar.

A estratégia de encaminhamento DM é uma implementação típica de um protocolo baseado na difusão não selectiva (*flooding*), mais adequada a topologias densas, à seme-

lhança dos protocolos bem conhecidos DVRMP e PIM-DM. O módulo DM pode aliás ser previamente configurado para se comportar como qualquer um desses dois protocolos, através da alteração de uma variável de classe. Por omissão, a estratégia seguida é a do PIM-DM.

A terceira estratégia incluída no simulador é o ST e constrói apenas árvores partilhadas centradas num RP. Trata-se de uma estratégia mais adequada a topologias esparsas, semelhante em alguns aspectos ao PIM-SM. Qualquer nó que se queira juntar ao grupo, envia uma mensagem de *join* em direcção ao RP, instalando entradas nas tabelas de encaminhamento de todos os nós ao longo do caminho que vai sendo percorrido. Para abandonar o grupo os nós enviam mensagens de *prune* ao RP, desfazendo o estado instalado ao longo do percurso. A implementação actual não reage a alterações dinâmicas do estado da rede durante a simulação.

A quarta estratégia disponível no simulador, designa-se por BST, e constrói árvores partilhadas bidireccionais centradas num RP. Tal como acontece no ST, a árvore partilhada é construída e destruída com base no envio explícito de mensagens de *join* e *prune* dos nós para o RP. Tal como a implementação ST, também esta não reage ao dinamismo da rede ao longo da simulação.

Analisando as quatro estratégias incluídas no NS conclui-se que embora duas delas (a centralizada e a ST) tenham algumas semelhanças com a estratégia de encaminhamento PIM-SM, nenhuma a implementa de facto. Na estratégia centralizada não são enviadas mensagens de controlo e a comutação entre árvores ocorre de forma brusca, enquanto a estratégia ST não constrói sequer árvores centradas nas fontes.

4.2.2.2 Início do processo de encaminhamento multicast na simulação

No NS as extensões *multicast* são activadas logo na altura em que se cria o simulador. Ao serem activadas as extensões *multicast* acrescenta-se o módulo **Mcast** à variável **module_list** (variável de classe da classe **Node**). Esta variável é usada sempre que se cria um novo nó para determinar que extensões se pretende adicionar aos nós. Neste caso quando um novo nó é criado é-lhe acrescentado o novo **classifier_** (o **multiclassifier_**, uma instância da classe **Classifier/Multicast/Replicator**), capaz de fazer com que o nó suporte o envio de tráfego *multicast*.

Além disso é colocado em cada nó uma instância da classe **mrtObject**. Este objecto é capaz de accionar métodos em cada um dos protocolos de encaminhamento que estão a correr num nó. Existe no máximo um protocolo de encaminhamento *multicast* associado a cada interface do nó. O objecto **mrtObject** mantém um *array* (**protocols_**), indexado por interface, com um apontador para cada um dos protocolos de encaminhamento activos. Através do método **all-mrprotos** da classe **mrtObject** consegue-se invocar os vários métodos específicos de cada um dos protocolos de encaminhamento *multicast* (por exemplo **join-group**, **leave-group**, etc).

Em seguida é necessário especificar qual é a estratégia usada para construir a árvore

multicast. Para isso usa-se o método **mrtproto**, que é um método da classe **Simulator**.

```
$ns mrtproto <estrategia> <lista de nos <lista de interfaces>>
```

É este método que vai desencadear o preenchimento da variável **protocols_** em cada **mrtObject** associado a cada nó. E é também a partir daqui que são criadas as várias instâncias dos protocolos de encaminhamento *multicast* que estão a correr em cada nó associados a cada interface.

Além de uma instância da classe **mrtObject** instância essa que contém o *array* **protocols_**, como já foi referido, cada nó tem ainda uma variável **replicator_** e uma variável **agent_**.

A variável **replicator_** é um *array* indexado por par "fonte:grupo", que contém objectos do tipo **Classifier/Replicator/Demuxer**. Cada um destes objectos contem todos os agentes que fizeram *join* ao grupo naquele nó bem como todas as interfaces de saída para onde tem de seguir determinado pacote *multicast* para percorrer a árvore de distribuição *multicast*.

A variável **agent_** é um *array* indexado por grupo que contém em cada entrada todos os agentes que fizeram *join* ao grupo naquele nó.

4.3 Implementação do PIM-SM no NS

Tendo-se concluído que nenhuma das estratégias existentes no NS implementa com suficiente proximidade a estratégia de encaminhamento PIM-SM, partiu-se para a sua implementação de raíz, usando como ponto de partida a estratégia ST, descrita na secção 4.2.2.

O primeiro passo foi identificar os módulos e funções que seria necessário escrever do zero e aqueles que poderiam ser simplesmente derivados, aproveitando o facto de o simulador ser orientado a objectos.

O encaminhamento *multicast* envolve duas actividades distintas que fazem uso de uma estrutura de dados comum que são as tabelas de encaminhamento. Uma das actividades consiste em manter (acrescentar e remover) as entradas dessas tabelas, e que sendo tipicamente uma actividade de controlo deve ser escrita em linguagem OTcl e outra que consiste na consulta dessas tabelas sempre que um pacote de dados precisa de ser encaminhado e que pela frequência com que se realiza deve ser implementada em C++. No primeiro caso as actividades são realizadas por um agente de controlo *multicast* a instalar em todos os nós da topologia enquanto no segundo caso são realizadas pelos classificadores também instalados em todos os nós da topologia.

Dado que a estratégia PIM-SM permite que os nós comutem de árvore de difusão a qualquer momento, as tabelas de encaminhamento podem conter, em simultâneo e para o

mesmo grupo G, entradas (*,G) relativas à árvore partilhada e entradas (S,G) relativas a árvore(s) centrada(s) na(s) fonte(s) S. Para além disso e por forma a evitar que a transição de árvore se faça bruscamente (e com perdas de dados), as entradas (S,G) são criadas num primeiro instante num estado inactivo e são mantidas nesse estado até que o primeiro pacote de dados chegue pela nova árvore. Este facto só por si é determinante para a maioria das decisões de implementação tomadas.

Por um lado torna-se óbvio que é necessário rever a estrutura de dados que implementa as tabelas encaminhamento nos nós. A nova estrutura deve permitir a coexistência de entradas (*,G) e (S,G) e, além disso, incluir as *flags* que caracterizam as rotas. Como as tabelas de encaminhamento são mantidas nos classificadores, é preciso pelo menos derivar uma nova sub-classe que inclua a nova estrutura de dados. Mas as implicações não são apenas essas, porque como é o classificador que recebe e replica os pacotes de dados dos vários grupos *multicast*, tem de ser ele também a detectar quando chega o primeiro pacote de dados que completa a transição da árvore partilhada para uma árvore centrada na fonte.

Por outro lado o agente de controlo *multicast* que recebe e envia as mensagens de controlo *join* e *prune* em cada nó, tem de ter pelo menos um novo método (**join-group** <**fonte**>) que possa ser invocado para dar início à comutação de árvore. Este novo método originará mensagens de controlo que podem ser dirigidas não só às fontes, mas também a todos os encaminhadores encontrados no caminho. Isto faz com que cada agente passe a receber mensagens dos seus vizinhos dirigidas ora ao RP ora a uma das fontes, e tem de reagir de maneira diferente e de acordo com o contexto local a cada uma delas. Esta constatações sugerem um nível de complexidade que obriga a implementar uma nova classe OTcl para o agente de encaminhamento PIM-SM.

Conclui-se assim pela necessidade de (1) implementar um novo agente *multicast* PIM-SM e (2) derivar um novo classificador PIM-SM, revendo as estruturas de dados da tabela de encaminhamento. Estas duas decisões de implementação, vão ser analisadas com mais detalhe nas secções seguintes.

4.3.1 Implementação de um agente PIM-SM

Relativamente ao agente de controlo PIM-SM, as suas funções são enviar, receber e processar mensagens de controlo *join* e *prune* de forma a construir, manter e destruir a árvore partilhada e as árvores centradas nas fontes.

Podemos organizar os estímulos a que o agente tem de responder em duas grandes categorias: (a) o nó que hospeda o agente abandona ou junta-se a um grupo ou a uma fonte desse grupo (b) o agente recebeu mensagens de *join* e *prune* dos agentes seus vizinhos destinados ao RP ou a uma das fontes. Para a primeira situação foram concebidos dois métodos que podem ou não ser invocados com uma fonte como parâmetro adicional: **join-group** [<**fonte**>] e **leave-group** [<**fonte**>]. Quando não é especificado nenhum parâmetro, significa que se trata de um operação de subscrição/abandono de um grupo na

totalidade, e ao especificar a fonte que se trata de um junção/abandono à árvore centrada nessa fonte. Para a segunda situação (b) foram concebidos igualmente dois métodos: **recv-join** e **recv-prune**, que são invocados consoante o agente recebeu uma mensagem de *join* ou de *prune*.

O agente deve reagir a cada invocação destes quatro métodos de acordo com o contexto actual do nó em que se encontra. Podemos também identificar quatro situações distintas, que se podem identificar consultando a tabela de encaminhamento e que são (a) o nó já está na árvore partilhada do grupo (b) o nó já está na árvore centrada na fonte (c) o nó ainda não está na árvore partilhada e (d) o nó ainda não está na árvore centrada na fonte. A reacção do agente a cada uma destas situações pode ser uma ou mais das seguintes actividades: não fazer nada, criar, remover ou alterar entradas na tabela de encaminhamento do nó e enviar mensagens de controlo aos nós vizinhos.

O agente PIM-SM implementado constitui uma nova classe OTcl (**PIM.tcl**). Esta classe, à semelhança das estratégias de encaminhamento *multicast* que integram o NS, deriva da classe **MCastProto**.

O comportamento global resultante, foi alinhado com a norma que descreve o protocolo PIM-SM. A seguir são descritos os principais métodos implementados pelo agente de encaminhamento PIM-SM.

- **Método join-group**

  Este método deverá ser invocado por um agente quando este decide juntar-se a um grupo ou quando decide comutar da árvore partilhada para uma árvore centrada na fonte.

  Se for invocado com apenas um argumento, o método depreende que o agente pretende juntar-se ao grupo (ou seja, à árvore partilhada) e invoca o método **join-rpt**.

  Se for invocado com dois argumentos quer dizer que o agente já faz parte do grupo e pretende apenas comutar da árvore partilhada para uma árvore centrada na fonte. Neste caso o método invoca o método **join-spt**.

- **Método join-rpt**

  Este método deverá ser invocado quando um agente decide juntar-se a um grupo. Podem ocorrer três situações distintas:

  - O nó ainda não faz parte da árvore partilhada, ou seja, ainda não existe nenhuma entrada (*,G) na tabela de encaminhamento *multicast* do nó. Nesse caso, a entrada tem que ser criada. A interface de entrada deverá ser preenchida com a melhor interface de saída para chegar ao RP, o RPT-Bit é preenchido com 1 e o SPT-Bit é preenchido com 0. Depois é necessário enviar um pedido de *join* para o próximo nó no caminho mais curto até ao RP.
  - Já existe uma entrada (*,G), mas a lista de interfaces de saída está vazia. Neste caso também é necessário enviar um pedido de *join* para o próximo nó no caminho mais curto até ao RP.

- Já existe uma entrada (*,G) e a lista de interfaces de saída não está vazia. Nesta caso, nem sempre é necessário enviar um *join*, como veremos adiante.

Em qualquer das situações o agente que fez o *join* ao grupo tem que ser incluído na lista de interfaces de saída (**oif_list**) da entrada (*,G). Isso já é feito no método **join-group** da classe **Node**, através da actualização da variável **replicator_**.

Depois é necessário verificar se existem entradas do tipo (S,G). Se existirem, o agente tem que ser acrescentado também às respectivas listas de interfaces de saída, o que também é feito pelo método **join-group** da classe **Node**.

Se existir a entrada (*,G) e algumas das entradas (S,G) tiverem o RPT-Bit a 1, quer dizer que já existem fontes que não estão a usar a árvore partilhada. Só neste caso, precisamos de propagar o *join* pela árvore partilhada acima até ao RP para garantirmos que esta situação é remediada para o novo membro do grupo, que como é óbvio deseja receber tráfego de todas as fontes.

Em qualquer das situações descritas, sempre que é necessário enviar um pedido de *join* para o nó vizinho no caminho mais curto até ao RP, o método **send-ctrl** deverá ser invocado com os seguintes argumentos:

- a string "join" para indicar que se trata de um pedido de junção a uma árvore;
- o endereço do RP do grupo;
- o endereço do grupo;
- o WC-Bit preenchido a 1;
- o RPT-Bit preenchido a 1;

A figura 4.4 representa a implementação do processo de junção á árvore partilhada. As diferentes tarefas, bem como os métodos responsáveis pela sua implementação estão numerados de acordo com uma lógica temporal.

- Método join-spt

Este método deverá ser invocado quando um agente decide comutar para uma árvore centrada na fonte. Neste caso, o agente poderá pretender continuar ligado à árvore partilhada excepto para uma determinada fonte. Podem ocorrer duas situações distintas:

- Não existe uma entrada (S,G) na tabela de encaminhamento do nó, sendo S a fonte para a qual o agente decide comutar, ou seja, o nó ainda não faz parte de uma árvore centrada na fonte S. Neste caso, é necessário criar a entrada (S,G). A interface de entrada deverá ficar igual à melhor interface para chegar até à fonte. A lista de interfaces de saída deve ficar igual à lista de interfaces de saída da entrada (*,G). O SPT-Bit da nova entrada deverá ser posto a zero, assim como o RPT-Bit. Depois de criada a entrada (S,G), deve ser enviado um pedido de *join* para o próximo nó no caminho mais curto até à fonte S.

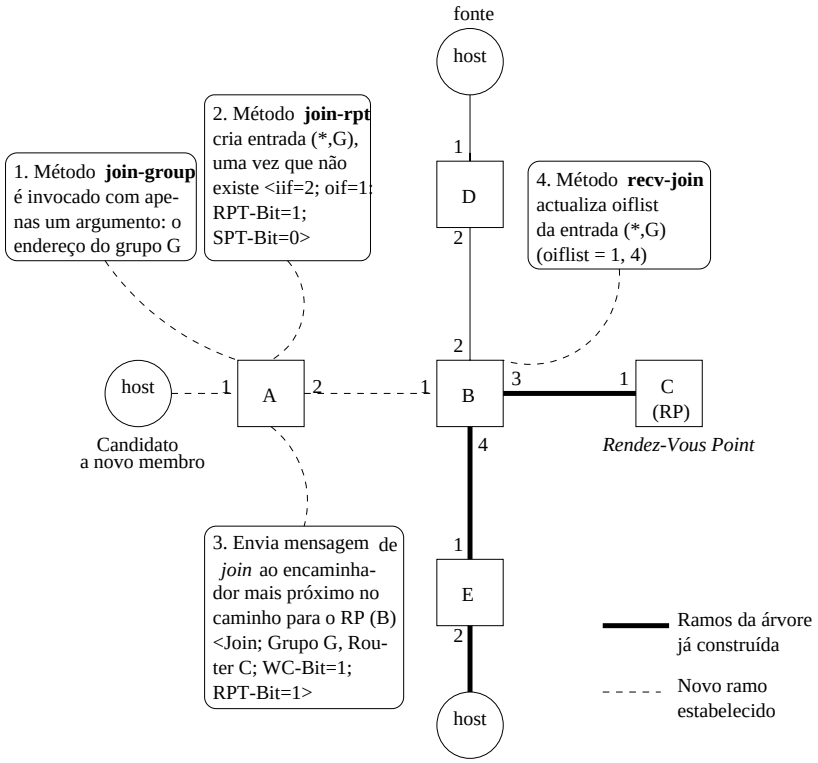


Figura 4.4: Processo de junção de um nó a uma árvore partilhada já parcialmente construída

- Já existe uma entrada do tipo (S,G). É necessário verificar se ela tem o RPT-Bit a 0 ou a 1. Se tiver o RPT-Bit a 1 é necessário pô-lo a 0 e substituir a interface de entrada pela melhor interface para chegar à fonte. Por fim é necessário enviar um pedido de *join* para o nó vizinho no melhor caminho até à fonte. Se a entrada (S,G) já existir e tiver o RPT-Bit a 0, só é necessário enviar um pedido de *join* para o nó vizinho no melhor caminho até à fonte se a lista de interfaces de saída estiver vazia.

Em qualquer das situações o agente que pretende comutar para uma árvore centrada na fonte tem que ser incluído na lista de interfaces de saída (**oif_list**) da entrada (*,G). Isso já é feito no método **join-group** da classe **Node**, através da actualização da variável **replicator_**.

Em qualquer das situações descritas sempre que é necessário enviar um pedido de *join* para o nó vizinho no caminho mais curto até à fonte, o método **send-ctrl** deverá ser invocado com os seguintes argumentos:

- a string "join" para indicar que se trata de um pedido de junção a uma árvore;
- o endereço da fonte para a qual se quer comutar;

- o endereço do grupo;
- o WC-Bit preenchido com 0;
- o RPT-Bit preenchido com 0;

A figura 4.5 representa a implementação do processo de transição da árvore partilhada para uma árvore centrada na fonte. Numa primeira fase a mensagem de *join* é enviada em direcção à fonte criando pelo caminho a informação de estado relativa ao novo ramo (figura 4.5(a). Numa segunda fase o novo ramo é tornado efectivo e é efectuado um *prune* da fonte em causa na árvore partilhada (figura 4.5(b)). As diferentes tarefas, bem como os métodos responsáveis pela sua implementação estão numeradas de acordo com uma lógica temporal.

- **Método recv-join**

  Este método é invocado sempre que um nó recebe uma mensagem de *join* à árvore partilhada ou um *join* a uma determinada fonte. Para distinguir um do outro é necessário testar o WC-Bit e o RPT-Bit. Se ambos estiverem a 1 trata-se de um pedido de *join* à árvore partilhada. Senão trata-se de um *join* a uma fonte.

  - Recepção de *join* à árvore partilhada

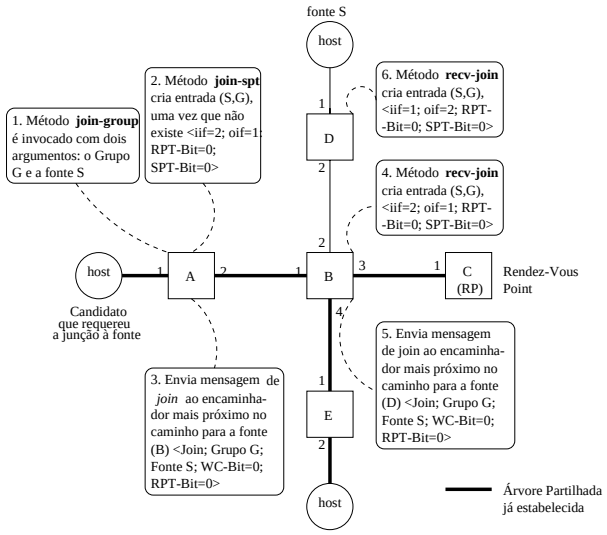
    Neste caso é necessário testar se o nó já pertence à arvore partilhada, ou seja se já existe uma entrada (*,G) na tabela de encaminhamento do nó.

    Se não existir, é necessário criá-la. A interface de entrada a colocar na nova entrada (*,G) deverá ser a melhor interface de saída para chegar ao RP. Na lista de interfaces de saída deve ser colocada a interface por onde chegou o pedido de *join*. Se a entrada (*,G) já existir mas a lista de interfaces de saída estiver vazia é necessário colocar na lista de interfaces de saída a interface por onde chegou o pedido de *join*. Em qualquer destes dois casos é necessário re-enviar o *join* para o nó vizinho em direcção ao RP.

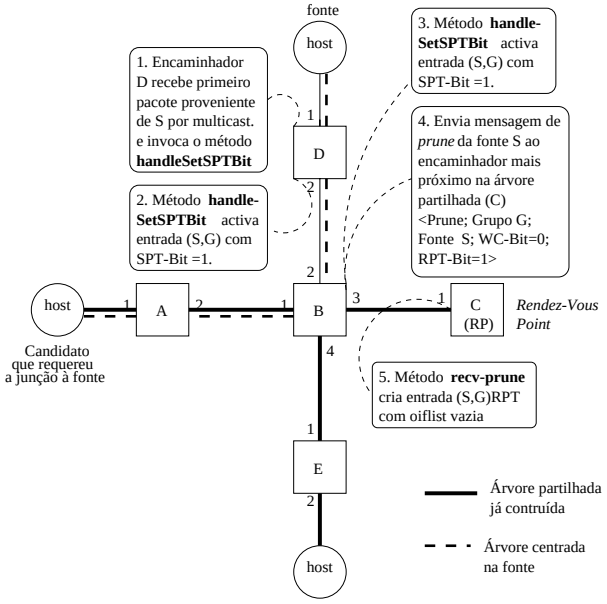
    Se já existir uma entrada (*,G) com a lista de interfaces de saída não vazia só é necessário acrescentar a interface donde veio o pacote de *join* à lista de interfaces de saída da entrada (*,G). Neste caso não se re-envia o pedido de *join*, a menos que haja alguma entrada (S,G) com o RPT-Bit a 1.

    Em qualquer das situações, é necessário verificar se existe alguma entrada do tipo (S,G) relativa ao mesmo grupo a que se fez o *join*. Se existir é necessário acrescentar a interface donde veio o pacote de *join* à lista de interfaces de saída da entrada (S,G), quer a entrada tenha o RPT-Bit a 1 quer não. Se a entrada (S,G) tiver o RPT-Bit a 1 é ainda necessário propagar o *join* em direcção ao RP para que os nós que estão acima na árvore partilhada transportem os pacotes provenientes de S até ao encaminhador.

    Ao acrescentarmos a nova interface a uma entrada (S,G) com o RPT-Bit a 1, a lista de interfaces de saída dessa entrada poderá ficar igual à lista de interfaces de saída da entrada (*,G). Nesse caso, a entrada (S,G) com RPT a 1 deverá ser apagada.



(a) *A mensagem de join é propagada*



(b) *O novo ramo torna-se efectivo*

Figura 4.5: Processo de comutação da árvore partilhada para uma árvore centrada na fonte

– Recepção de um *join* a uma fonte

Antes de mais é necessário testar se o nó já pertence ou não a uma árvore centrada na fonte, ou seja se já existe uma entrada (S,G) na tabela de encaminhamento do nó.

Se não existir, é necessário criá-la. A interface de entrada deverá ficar igual à melhor interface para chegar até à fonte. A lista de interfaces de saída deve ficar igual à lista de interfaces de saída da entrada (*,G), se esta existir, acrescentando a interface por onde chegou o pedido de *join*. O SPT-Bit da nova entrada deverá ser posto a zero, assim como o RPT-Bit. Depois de criada a entrada (S,G) o pedido de *join* deve ser re-enviado para o próximo nó no caminho mais curto até à fonte S.

Se não existir nenhuma entrada (*,G) a lista de interfaces de saída da nova entrada deverá ficar apenas com a interface por onde chegou o pedido de *join*. O SPT-Bit da nova entrada deverá na mesma ser posto a zero, assim como o RPT-Bit. O pedido de *join* deve ser re-enviado ao próximo nó no caminho mais curto até à fonte S.

Se já existir uma entrada do tipo (S,G), é necessário verificar se ela tem o RPT-Bit a 0 ou a 1. Se tiver o RPT-Bit a 1 é necessário pô-lo a 0 e substituir a interface de entrada pela melhor interface para chegar à fonte. Por fim é necessário re-enviar o pedido de *join* para o nó vizinho no melhor caminho até à fonte.

Se a entrada (S,G) já existir e tiver o RPT-Bit a 0 e a lista de interfaces de saída estiver vazia é necessário colocar na lista de interfaces de saída a interface por onde chegou o pedido de *join* e re-enviar o *join* para o nó vizinho em direcção à fonte. Se já existir uma entrada (S,G) com a lista de interfaces de saída não vazia é necessário, apenas, acrescentar a interface donde veio o pacote de *join* à lista de interfaces de saída da entrada (S,G). Neste caso não se re-envia o pedido de *join*.

- **Método handle-SetSPTBit**

  Este método é invocado pelo classificador PIM-SM quando pretende fazer o *prune* de uma fonte na árvore partilhada.

  Como já foi referido, quando um determinado agente quer comutar da árvore partilhada para uma árvore centrada numa fonte, é criada pelo método **join-source** uma entrada do tipo (S,G) com o SPT-Bit a 0. Este bit mantém-se a zero temporariamente até que o tráfego da fonte S começa a fluir através da árvore centrada na fonte. Quando isso acontece o bit é posto a 1 para tornar efectiva a entrada (S,G). Essa alteração do SPT-Bit de 0 para 1 é feita pelo classificador PIM-SM, assim que recebe o primeiro pacote vindo da fonte S. Depois é necessário fazer um *prune* daquela fonte na árvore partilhada para evitar que o tráfego proveniente daquela fonte chegue ao nó também através da árvore partilhada. Esse *prune* é diferente dos outros porque tem o RPT-Bit a 1, apesar de ter o endereço de uma fonte na lista de *prunes*, e o WC-Bit a 0. Este pedido deve ser re-enviado em direcção ao RP e não em direcção à fonte. Este *prune* só deverá ser enviado se a árvore pertencer à árvore partilhada.

- **Método leave-group**

  Este método é invocado quando um determinado agente associado a um nó decide abandonar um grupo, ou quando decide apenas deixar de receber tráfego através de uma árvore centrada numa fonte para a qual o agente comutou.

  Se for invocado com apenas um argumento, o método depreende que o agente pretende abandonar o grupo e invoca não só o método **leave-rpt** para abandonar a árvore partilhada, mas também o método **leave-spt** tantas vezes quantas as necessárias para ser removido de todas as árvores centradas nas fontes para as quais eventualmente tenha comutado.

  Se for invocado com dois argumentos quer dizer que o agente pretende apenas deixar de receber tráfego de uma determinada árvore centrada numa fonte para a qual comutou. Nesse caso o método invoca o método **leave-spt**.

- **Método leave-rpt**

  Para remover o agente da árvore partilhada é necessário apagá-lo da lista de interfaces da saída da entrada (*,G). Isto é feito pelo método **leave-group** da classe **Node**.

  Se a lista de interfaces de saída da entrada (*,G) ficar vazia, a entrada poderá ser apagada e deverá ser enviado um *prune* para o nó no caminho mais curto até ao RP. Um *prune* deste tipo tem o RPT-Bit e o WC-Bit ambos a 1. Alem disso o endereço que aparece na lista de *prunes* é o endereço do RP do grupo.

- **Método leave-spt**

  Este método é invocado pelo método **leave-group** para retirar determinado agente de todas as árvores centradas na fontes para as quais eventualmente tenha comutado.

  Para remover um agente de uma árvore centrada numa determinada fonte S é necessário apagá-lo da lista de interfaces da saída da entrada (S,G). O agente deverá ser apagado quer a entrada tenha o RPT a 1 ou a 0. Isto é feito pelo método **leave-group** da classe **Node**.

  Se a entrada (S,G) tiver o RPT-Bit a 1, a entrada (S,G) não deverá ser apagada, mesmo que a lista de interfaces de saída fique vazia. Se a lista de interfaces de saída ficar vazia dever-se-á enviar um *prune* da fonte S na árvore partilhada. Este *prune* é semelhante ao enviado pelo método **handle-SetSPTBit**, ou seja, apesar de ter o endereço de uma fonte na lista de *prunes* deve ser propagado em direcção ao RP. Para que isso aconteça o RPT-Bit deverá ser posto a 1 e o WC-Bit a 0.

  Se a entrada (S,G) tiver o RPT-Bit a 0 e a lista de interfaces de saída ficar vazia, a entrada é apagada e o *prune* deverá ser re-enviado para o nó no caminho mais curto até à fonte. Um *prune* deste tipo tem o RPT-Bit e o WC-Bit ambos a 0 e o endereço que aparece na lista de *prunes* é o endereço da fonte.

- **Método recv-prune**

  Este método é invocado sempre que um nó recebe uma mensagem de *prune*. Existem basicamente três *prunes* possíveis: o *prune* de um nó na árvore partilhada, o *prune* de

um nó numa árvore centrada na fonte e o *prune* de uma fonte na árvore partilhada. Para os distinguir é necessário testar o WC-Bit e o RPT-Bit.

- Recepção de um *prune* de um nó na árvore partilhada

  Um *prune* deste tipo tem o RPT-Bit e o WC-Bit ambos a 1. Além disso o endereço que aparece na lista de *prunes* é o endereço do RP do grupo. Ao receber um *prune* deste tipo deve-se verificar antes de mais se existe alguma entrada (*,G). Se existir, a interface por onde chegou o *prune* deve ser apagada da lista de interfaces de saída da entrada (*,G). Se a lista de interfaces de saída ficar vazia, a entrada poderá ser apagada e o *prune* deverá ser re-enviado para o nó no caminho mais curto até ao RP.

  Depois é necessário verificar se existem entradas (S,G) onde consta a interface por onde chegou o *prune*. Se existirem, a interface por onde chegou o *prune* deve ser apagada da lista de interfaces de saída. Se ao apagar a interface, a lista de interfaces de saída ficar vazia há que considerar duas situações possíveis:

  * Se a entrada (S,G) tiver o RPT-Bit a 0 então é necessário apagar a entrada e propagar o *prune* em direcção à fonte S com o WC-Bit e o RPT-Bit, ambos a zero.
  * Se a entrada (S,G) tiver o RPT-Bit a 1 então não se pode apagar a entrada (S,G), mas é necessário propagar o *prune*, desta vez em direcção ao RP com o RPT-Bit a 1 e o WC-Bit a 0.

- Recepção de um *prune* de um nó na árvore centrada na fonte

  Um *prune* deste tipo tem o RPT-Bit e o WC-Bit ambos a 0. O endereço que aparece na lista de *prunes* é o endereço da fonte. Ao receber um *prune* deste tipo deve-se verificar se existe alguma entrada (S,G). Se existir, à que considerar novamente duas situações possíveis:

  * Se a entrada (S,G) tiver o RPT-Bit a 0 então é necessário apagar a entrada e propagar o *prune* em direcção à fonte S com o WC-Bit e o RPT-Bit, ambos a zero.
  * Se a entrada (S,G) tiver o RPT-Bit a 1 então não se pode apagar a entrada (S,G), mas é necessário propagar o *prune*, desta vez em direcção ao RP com o RPT-Bit a 1 e o WC-Bit a 0.

- Recepção de um *prune* de uma fonte na árvore partilhada

  Um *prune* deste tipo tem o RPT-Bit a 1 e o WC-Bit a 0. O endereço que aparece na lista de *prunes* é o endereço da fonte da qual não se quer receber mais pacotes através da árvore partilhada. Ao receber um *prune* deste tipo deve-se verificar se existe alguma entrada (S,G). As duas situações a considerar são novamente as mesmas:

  * Se a entrada (S,G) tiver o RPT-Bit a 0 então é necessário apagar a entrada e propagar o *prune* em direcção à fonte S com o WC-Bit e o RPT-Bit, ambos a zero.

* Se a entrada (S,G) tiver o RPT-Bit a 1 então não se pode apagar a entrada (S,G), mas é necessário propagar o *prune*, desta vez em direcção ao RP com o RPT-Bit a 1 e o WC-Bit a 0.

Se existir uma entrada (*,G), mas não existir uma entrada (S,G), tem que ser criada uma entrada (S,G) com o RPT-Bit a 1. A lista de interfaces de saída é copiada da lista de interfaces de saída da entrada (*,G), apagando-se desta lista a interface por onde chegou o *prune*.

4.3.2 Implementação de um classificador PIM-SM

Como já foi referido, no NS as entidades responsáveis pelo re-envio dos pacotes nos nós são os classificadores. A tarefa de um classificador é examinar alguns campos de um pacote de dados, e mapear o valor desses campos num qualquer objecto de destino, que é o próximo destinatário do pacote de dados. No caso dos classificadores que implementam o re-envio de tráfego *multicast* pode ser necessário entregar o pacote a mais do que um objecto. Nesse caso é necessário recorrer a um outra entidade designada por classificador-replicador, que mantém uma lista de ligações ou agentes locais e entrega uma réplica dos pacotes de dados a cada um deles.

Com base nas classes já existentes descritas neste capítulo (secção 4.2), e tendo em vista a implementação do re-envio de tráfego de acordo com o PIM-SM (que é ligeiramente distinto do habitual, como já se referiu na secção 4.1.7 deste capítulo), as questões que se colocam são as seguintes:

- De onde derivar o novo classificador PIM-SM?
- Que métodos é necessário reescrever?

A hipótese mais lógica é derivar do **MCastClassifier** (o classificador *multicast* standard), com o cuidado de acrescentar os novos campos de dados na estrutura *hashnode*, porque o PIM-SM introduz mais informação de estado na tabela de encaminhamento (*Flags*, *Timers*, etc).

Quanto à segunda questão e tendo em conta que todo o trabalho de selecção de rotas *multicast* é feito pelo método **classify**, limitando-se os outros (os métodos **recv** e **find**) a fazer uso dele implementou-se todo o comportamento característico do PIM-SM directamente no método **classify**.

Como resultado, foram escritos os módulos **classifier-pim.cc** e **classifier-pim.h**. As estruturas de dados mantidas neste classificador estão ilustradas na figura 4.6. Os campos novos, que não existiam no **MCastClassifier**, estão ilustrados a cor diferente.

Trata-se basicamente de uma tabela de *hashing* com uma lista de colisões associada a cada posição. Cada posição da tabela de *hashing* é um apontador para um nó que representa entrada (S,G) ou (*,G), com todos os seus dados, e um apontador para o

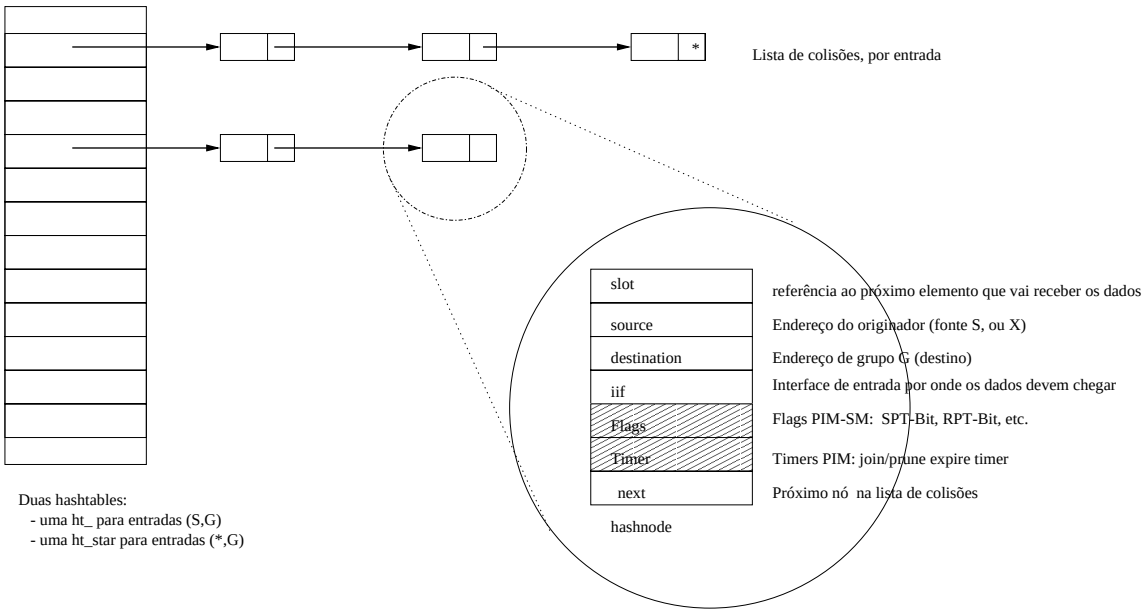


Figura 4.6: Estruturas de dados usadas no classificador PIM-SM

próximo *hashnode*, caso exista uma lista de colisões (mesmo valor devolvido pela função de sumariação).

Como a tabela de encaminhamento é mantida em tabelas de *hashing*, onde cada entrada representa uma rota, houve necessidade de derivar o registo de base por forma a incluir algumas *flags* de caracterização das rotas e temporizadores associados à rota. Uma das *flags* a incluir foi o SPT-Bit que classifica uma entrada (S,G) como inactiva (0) ou activa (1). A entrada é sempre criada como inactiva, quando um nó envia a mensagem de *join* à fonte e passa a activa quando chega o primeiro pacote de dados vindo dessa fonte.

Parte das funcionalidades do classificador de base puderam ser aproveitadas, nomeadamente as tabelas de *hashing* e respectivas funções de pesquisa e inserção, bem como os métodos usados na recepção dos pacotes pelo classificador. Foi rescrito apenas o método **classify**, que efectivamente classifica o pacote de dados extraindo do cabeçalho informação necessária para pesquisar na tabela de encaminhamento a rota de saída mais adequada. O algoritmo de classificação foi rescrito por forma a procurar em primeiro lugar entradas do tipo (S,G) relativas a árvores centradas na fonte e só depois entradas (*,G) relativas à árvore partilhada. Se existirem entradas (S,G) ainda inactivas (SPT-Bit ainda a zero) deve-se invocar o procedimento que finaliza de forma adequada a comutação da árvore partilhada para uma árvore centrada na fonte.

A tabela 4.1 procura sintetizar o comportamento implementado, em função das variáveis de entrada que estão em jogo: endereço de origem (SRC), endereço de destino (DST), interface de entrada do pacote de dados (IFACE), SPT-Bit, RPT-Bit. Para cada combinação destas variáveis de entrada, define-se qual a acção a desenvolver.

SRC	DST	IFACE	SPT-Bit	RPT-Bit	ACÇÃO
S	G	iif	1	x	Re-envia
S	G	iif	0	x	Re-envia Se (iif != IIF(*,G)) - Activa SPT-Bit - envia PRUNE (*,G)
S	G	não coincide	1	x	Descarta
S	G	não coincide	0	x	Acção baseada em (*,G)
*	G	iif	x	x	Descarta
*	G	não coincide	x	x	Descarta

Tabela 4.1: Comportamento do classificador PIM-SM mediante os valores das variáveis endereço origem, endereço destino, interface de entrada e *flags* SPT-Bit e RPT-Bit

4.4 Teste da Implementação

Para testar a implementação do PIM-SM no NS é necessário responder a duas questões fundamentais: o que avaliar e como obter os dados para efectuar essa avaliação.

Sendo o objectivo do PIM-SM a construção de árvores de difusão, interessará obter indicadores da qualidade da(s) árvore(s) geradas. Os dados necessários para obter esses indicadores podem ser registados pelo simulador durante a execução da simulação, quer pela activação de opções de *trace* já predefinidas, quer pela inclusão cirúrgica de novos registos de dados no código OTcl produzido. A combinação desses dois métodos melhora a qualidade dos resultados obtidos, e foi esse o procedimento seguido para obtenção dos resultados apresentados em 4.4.3.

4.4.1 Indicadores

A qualidade das árvore pode ser medida de várias formas. Uma das formas consiste em contar o número de ligações entre encaminhadores que fazem parte da árvore. No caso do grupo ter mais do que uma árvore (centrada no RP ou centradas em fontes) a contagem do número de ligações deve incluir todas elas, porque são de facto todas necessárias para a distribuição de dados no grupo. Naturalmente que a contagem do número de ligações não tem em conta o tipo de ligações nem o seu custo, mas é um bom indicador da qualidade da árvore.

Outra possibilidade é contar o número de réplicas que são feitas, ao longo da árvore, de cada pacote de dados. Esta medida seria equivalente à anterior se existisse apenas uma árvore, dado que os pacotes seriam replicados exactamente uma vez por cada ramo da árvore, o que corresponderia exactamente ao número de ligações. Quando existe mais do que uma árvore por grupo, o valor do número de réplicas corresponde às ligações das

várias árvores que são de facto usadas durante a transmissão de pacotes. Pode portanto ser diferente do número total de ramos que formam as diferentes árvores. Neste caso o número de réplicas constitui uma melhor métrica do que o número de ligações, uma vez que mede os recursos que estão a ser efectivamente usados.

O NS, já regista no ficheiro de *traces* todo o percurso de todos os pacotes, o que pode ser usado para obter o número médio de réplicas. Quanto à contagem do número de ligações que constituem a árvore, só pode ser feita incluindo no código do módulo **PIM.tcl** um registo de *trace* (recorrendo ao método **annotate** do objecto **McastProto**) sempre que uma entrada é adicionada ou removida da tabela de encaminhamento.

4.4.2 Cenários de Simulação

Para testar o PIM-SM usaram-se duas topologias: uma rede típica de um grande ISP com 18 nós e uma topologia de 100 nós gerada aleatoriamente.

4.4.2.1 Topologia de rede típica de um ISP

A primeira topologia de rede usada é uma rede típica de um grande ISP[65] e está ilustrada na figura 4.7.

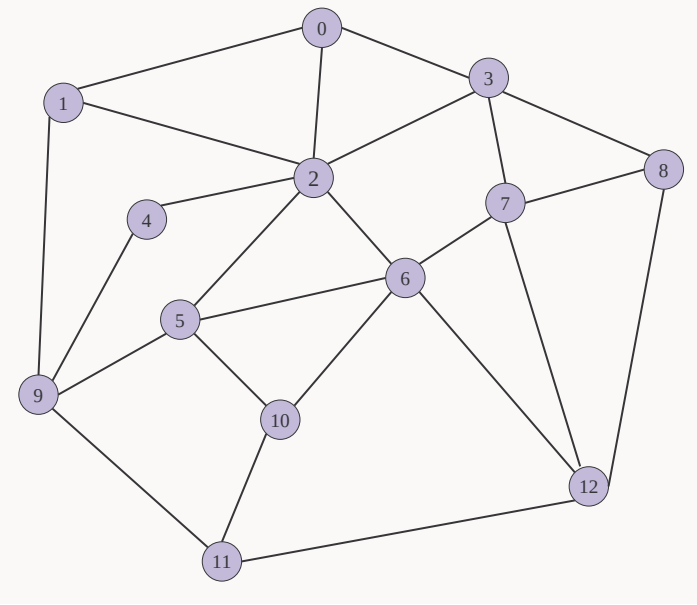


Figura 4.7: Topologia típica de um grande ISP

Esta topologia é constituída por 18 nós e 30 ligações simétricas de custo 1 entre eles. Todos os nós têm potenciais receptores (um associado a cada nó). Existem três fontes activas, nos nós 3, 9 e 15. O RP é escolhido aleatoriamente.

Com este cenário foram feitas duas experiências com o objectivo de comparar as árvores partilhadas com as árvores centradas nas fontes. Na primeira situação 50% os receptores aleatoriamente escolhidos entre os 18 disponíveis, juntam-se em instantes de tempo distintos à árvore partilhada e nenhum deles comuta de árvore. Na segunda experiência todos os receptores, que se juntaram ao grupo, ao fim de algum tempo comutam para uma árvore centrada na fonte associada ao nó 3. Mais tarde comutam também para uma árvore centrada na fonte associada ao nó 9 e por fim comutam para uma árvore centrada na fonte associada ao nó 15. Para cada experiência foram executadas 100 simulações independentes e os resultados apresentados representam a média dessas 100 simulações.

4.4.2.2 Topologia de rede com 100 nós

O segundo cenário de simulação usado foi constituído por uma segunda topologia, gerada usando o GT-ITM[66][3], um gerador de topologias de rede desenvolvido no Georgia Institute of Technology. A topologia gerada tem 100 nós e para criar as ligações entre eles usou-se o método aleatório puro com uma probabilidade de 0.33. Também neste caso o RP foi escolhido aleatoriamente entre todos os nós da topologia, mas desta vez optou-se por uma topologia assimétrica com os custos das ligações a serem escolhidos aleatoriamente dentro do intervalo [1, 10]. Existe um potencial candidato a novo membro em cada nó (100) e dez fontes fixas.

À semelhança do que aconteceu com o primeiro cenário foram feitas duas experiências. Na primeira experiência 10 dos candidatos a novos membros juntam-se ao grupo, um de cada vez (com intervalos de tempo de 0.25s) via árvore partilhada e não comutam para árvores centradas nas fontes. Na segunda experiência, ao fim de algum tempo os 10 membros do grupo já efectivos juntam-se a cada uma das fontes.

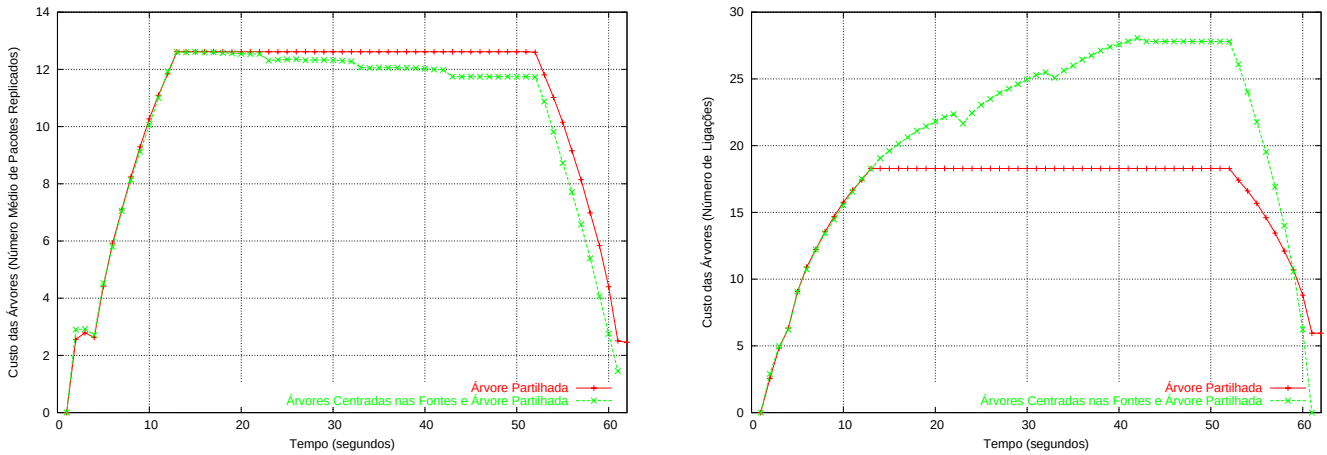
Foram executadas 100 simulações independentes para cada experiência e os resultados aqui apresentados representam a média dessas 100 simulações.

4.4.3 Análise de Resultados

Os resultados obtidos com o primeiro cenário de simulação (topologia típica de um ISP) estão representados nos gráficos que constam da figura 4.8.

O gráfico 4.8(a) mostra a qualidade das árvores em termos do número médio de réplicas feito por cada pacote de dados. Esta métrica é apresentada em função do tempo. As linhas estão praticamente sobrepostas (obviamente) até ao instante de tempo em que os membros do grupo iniciam a comutação para árvores centradas nas fontes. Nesses instantes de tempo nota-se claramente uma ligeira descida no valor da métrica. No que toca à métrica do número de ligações acontece precisamente o contrário como se pode observar no gráfico representado em 4.8(b). Este resultado é facilmente explicável se pensarmos que na situação em que os membros do grupo comutam para árvores centradas

[3]Georgia Tech - Internetworking Topology Models



(a) Número médio de réplicas

(b) Número de ligações

Figura 4.8: Resultados da Simulação para o primeiro cenário de teste - Topologia típica de um ISP

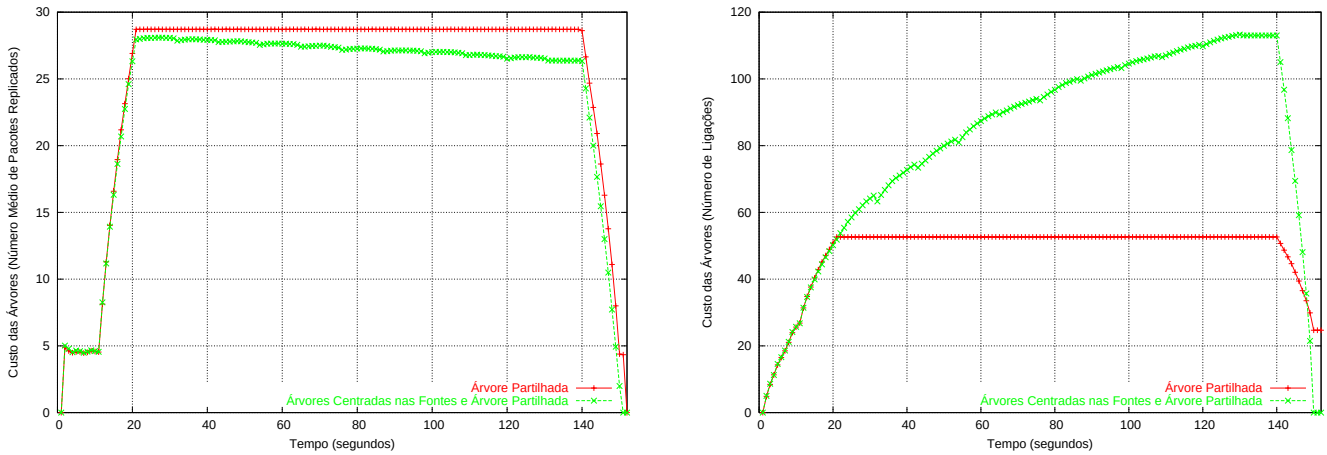
nas fontes irão existir mais árvores além da árvore partilhada, e por isso, mais ligações envolvidas. Todas as ligações envolvidas contribuem de igual modo para a métrica embora estas não sejam usadas simultaneamente por todos os pacotes, uma vez que nesta situação há pacotes que utilizam umas árvores e outros que utilizam outras, dependendo da fonte.

Os gráficos da figura 4.9 representam os resultados obtidos quando foi utilizado o segundo cenário de simulação (topologia aleatória assimétrica com 100 nós). A situação observada é idêntica à observada quando foi utilizado o primeiro cenário de simulação.

4.5 Conclusões

A contribuição apresentada neste capítulo refere-se à implementação e teste de um novo módulo para encaminhamento *multicast*, usando o Protocol Independent Multicast-Sparse Mode (PIM-SM). Embora o protocolo PIM-SM seja já bem conhecido e bastante utilizado, não existia até à data nenhuma implementação deste protocolo para o NS.

Da experiência resultante da implementação aqui apresentada dever-se-á realçar a dificuldade encontrada em incorporar o código desenvolvido, integrando-o simultaneamente nas hierarquias C++ e OTcl, como um novo módulo de encaminhamento *multicast* para o NS. Sendo um ambiente de simulação já bastante complexo e com múltiplas contribuições independentes, as interdependências entre os diferentes módulos nem sempre são fáceis de deduzir, até porque alguns dos módulos têm falhas importantes ao nível da respectiva documentação.



(a) Número médio de réplicas

(b) Número de ligações

Figura 4.9: Resultados da Simulação quando é usada a topologia aleatória de 100 nós

Para além desta implementação do PIM-SM, desenvolvida em C++ e OTcl e já integrada na hierarquia de classes OTcl deste ambiente de simulação, realizou-se ainda a simulação de algumas situações de encaminhamento de tráfego *multicast* num *backbone* com 18 nós e com uma topologia que proporciona múltiplos caminhos entre quaisquer dois dos seus nós. Simularam-se situações em que se utilizam árvores de distribuição *multicast*, quer árvores partilhadas quer árvores centradas na fonte, e analisaram-se os custos das diferentes árvores criadas, tanto em termos do número médio de réplicas de pacotes como em termos do número médio de ligações que o tráfego *multicast* atravessa. Estes mesmos testes foram repetidos com uma topologia de 100 nós, gerada aleatoriamente, tendo sido obtidos resultados equivalentes.

Finalmente será importante realçar que a implementação do PIM-SM não se constitui como um fim em si mesmo, mas antes como uma etapa preliminar e essencial no estudo de estratégias de encaminhamento *multicast* com requisitos de Qualidade de Serviço.

Capítulo 5

Árvores Directas no Encaminhamento Multicast

Este capítulo descreve um dos protocolos de encaminhamento *multicast* concebidos, implementados e avaliados no âmbito deste trabalho. Este novo protocolo, designado por DTMP (Directed Trees Multicast Protocol), é um protocolo muito semelhante ao protocolo PIM-SM nos seus princípios básicos: suporta simultaneamente árvores partilhadas e árvores centradas nas fontes, prevê mecanismos automáticos de comutação de uma árvore partilhada para uma árvore centrada na fonte sem perdas de pacotes durante o processo de comutação, e por fim, é independente do protocolo de encaminhamento *unicast* em utilização. Tem no entanto uma diferença fundamental na forma de construir as árvores de distribuição, quer sejam partilhadas quer sejam árvores centradas nas fontes. O protocolo DTMP constrói as árvores no sentido real usado pelo tráfego, ao contrário do PIM-SM que as constrói no sentido oposto. O objectivo é adaptar o encaminhamento *multicast* às redes assimétricas, ou seja a redes em que as ligações têm custos dinâmicos e diferentes dependendo do sentido em que se efectua o percurso.

5.1 Motivação

O encaminhamento do tráfego *multicast* é efectuado com base numa árvore ou num conjunto de árvores de distribuição. O tráfego é encaminhado desde a raíz da árvores até aos seus vértices, que coincidem com os destinatários (membros do grupo *multicast*). O objectivo dos protocolos de encaminhamento *multicast* é assim, construir a árvore de menor custo em função de um conjunto de parâmetros. Como foi referido no capítulo 2 este problema é um problema NP completo e é designado por **Problema da Árvore de Steiner**[67]. Foram propostas inúmeras heurísticas com o objectivo de construir eficientemente árvores de distribuição de tráfego *multicast*. Uma das heurísticas mais usadas pelo protocolos de encaminhamento *multicast* propõe a construção de uma árvore de distribuição de tráfego adicionando um participante de cada vez. Cada participante junta-se à árvore através de

um novo ramo que coincide com o melhor caminho que une o novo participante ao nó da árvore mais próximo. As árvores de distribuição de tráfego *multicast* construídas usando esta estratégia designam-se por árvores invertidas[1]. Os ramos são estabelecidos dos novos participantes em direcção à raíz da árvore ou seja no sentido oposto ao sentido usado pelo tráfego.

Esta estratégia é baseada no pressuposto de que as ligações entre os vários nós da rede são simétricas, ou seja, têm o mesmo custo em cada um dos sentidos. No entanto, nem sempre é esse o caso. Por exemplo, quando se pretende considerar requisitos de qualidade de serviço no encaminhamento, e escolher os "melhores" caminhos em função desses requisitos, o pressuposto de que as ligações entre os vários nós de uma rede são simétricas, não deve ser utilizado. Isto porque as ligações são quase sempre assimétricas em função da qualidade de serviço que disponibilizam. O mais natural, quando estamos perante Encaminhamento com QoS é termos ligações com custos diferentes em ambos os sentidos. Os custos das ligações no Encaminhamento com QoS são expressos em termos de requisitos de QoS, tais como, largura de banda disponível, atraso fim a fim, percentagem de perdas de pacotes, etc. Como é óbvio esses parâmetros variam, dependendo do sentido em que se efectua o percurso.

5.2 Trabalho relacionado

Como já foi referido no capítulo 2, grande parte dos protocolos de encaminhamento *multicast* assumem que as ligações entre quaisquer dois nós da rede são simétricas. As redes são modeladas através de grafos não orientados e as heurísticas utilizadas pretendem resolver o Problema da Árvore de Steiner em redes simétricas, problema esse que é NP_Completo.

Encontrar uma árvore de custo mínimo numa rede assimétrica, é também um problema NP-completo e designa-se por *Problema da Árvore Directa de Steiner*. Existem alguns estudos teóricos[68][69] que pretendem contribuir com aproximações para a resolução deste problema. No entanto, a maioria dos protocolos de encaminhamento *multicast* já desenvolvidos, como o DVMRP[34], o CBT[36] e o PIM-SM[7] são baseados no encaminhamento invertido. Apenas o MOSPF[35] consegue lidar com redes assimétricas pois a base de dados topológica é armazenada na forma de um grafo orientado. No PIM-SM os novos ramos da árvore de distribuição de tráfego são construídos à medida que as mensagens de *join* se propagam dos membros do grupo em direcção ao Ponto de Encontro (*Rendez-Vous Point*) ou às fontes. Se estivermos perante ligações assimétricas o caminho seguido pelas mensagens de *join* pode não ser o caminho mais curto que o tráfego deveria seguir em direcção ao destinatário, e em consequência disso as árvores de distribuição de tráfego (quer as árvores partilhadas como as árvores centradas nas fontes) podem ser "más" árvores. O protocolo CBT em redes assimétricas sofre do mesmo tipo de problema.

Em [70] é proposto um novo mecanismo para construção de árvores directas, baseado

[1] *reverse-path trees*, em Inglês

no CBT. O processo de *join* é semelhante ao usado pelo CBT. Um novo membro junta-se a um grupo através da propagação de uma mensagem de *join-request* em direcção ao *Core Node* estabelecido. Quando o *Core Node* recebe um *join-request* responde com uma mensagem de *join-ack* destinada ao originador do pedido de junção, mensagem essa que poderá não seguir o mesmo caminho que a mensagem de *join-request*. É ao longo desta última trajectória que é estabelecido o ramo da árvore que ligará o novo membro ao grupo. Da mesma forma que o CBT, também esta nova variante do protocolo poderá concentrar demasiado o tráfego em poucas ligações, aumentando mais a carga na rede e a probabilidade de congestão do que os protocolos de encaminhamento *multicast* que recorrem a árvores centradas nas fontes.

O REUNITE[62] implementa a distribuição do tráfego *multicast* com base na infra-estrutura de encaminhamento *unicast*, mais concretamente, usando árvores *unicast* recursivas. A ideia principal do REUNITE é baseada na constatação de que na sua maioria, os encaminhadores que constituem uma árvore de distribuição de tráfego *multicast*, limitam-se a re-enviar os pacotes *multicast* para uma única interface, ou seja, só uma minoria dos encaminhadores é que são responsáveis pela ramificação da árvore de distribuição. Para tirar partido deste facto e diminuir a complexidade do protocolo, o REUNITE propõe a utilização de duas tabelas: a MCT (*Multicast Control Table*) e a MFT (*Multicast Forwarding Table*). Os encaminhadores de onde não partem novos ramos da árvore de distribuição não consultam nenhuma das tabelas *multicast* para fazerem o re-envio dos pacotes, mantendo apenas a tabela MCT com informação relativa ao grupo. Pelo contrário, os encaminhadores de onde partem novos ramos da árvore mantém na MFT os endereços *unicast* dos destinatários e é através da consulta dessa tabela que os pacotes são duplicados e correctamente re-enviados para esses destinos. Além disso o REUNITE utiliza apenas os endereços *unicast* dos membros do grupo, por isso diz-se que no REUNITE o *multicast* é implementado usando *unicast* recursivo. A fonte envia dados em *unicast* dirigidos ao primeiro membro que se juntou ao grupo. No encaminhadores onde não existe MFT, o pacote é simplesmente re-enviado por *unicast*, nos nós em que existe a tabela MFT, quer dizer que é um encaminhador de onde partem ramos da árvore, a tabela é consultada e é efectuada uma copia do pacote por cada entrada na tabela MFT, sendo a copia re-enviada por *unicast* endereçada ao destinatário que consta da tabela.

O REUNITE apresenta algumas vantagens em relação aos protocolos de encaminhamento *multicast* tradicionais: permite um desenvolvimento progressivo da infra-estrutura *multicast* graças ao esquema de endereçamento e re-envio de tráfego utilizados. A utilização de endereçamento *unicast* faz com que o protocolo seja capaz de suportar encaminhadores exclusivamente *unicast* na árvore de distribuição. Por outro lado o REUNITE apresenta alguns problemas quando usado em redes assimétricas. Potencialmente constrói as árvores de distribuição de forma directa, uma vez que a árvore é construída através da acção de uma mensagem (*tree message* que é enviada pela fonte em direcção aos membros do grupo. A mensagem de *join* não introduz informação de estado nos encaminhares, a menos que passe por algum que já pertença á árvore de distribuição. Nesses casos, a mensagem de *join* não continua o seu caminho e é enviada uma *tree message* a partir desse nó em

direcção ao novo membro com o objectivo de criar um novo ramo. Assim, o caminho entre a fonte e o novo membro pode não ser o menor caminho possível, logo a árvore de distribuição construída pelo REUNITE pode não ser uma árvore de caminhos mais curtos, embora seja de facto criada de forma directa e não invertida.

Em [63] são propostas algumas alterações ao protocolo REUNITE com o objectivo de resolver este e outros problemas. É proposto um novo protocolo: o Hop by Hop Multicast Routing Protocol (HBHP), muito semelhante ao REUNITE. São usadas na mesma as duas tabelas MCT e MFT mas no HBHP as mensagens de *join* são sempre re-enviadas até à fonte a menos que seja encontrado um nó da árvore donde já esteja a sair um ramo especificamente destinado ao receptor que fez o *join*. Desta forma o HBHP consegue construir uma árvore de caminhos mais curtos por cada fonte. Todavia sofre dos mesmo problemas que os protocolos baseados unicamente em árvores centradas em fontes.

Tanto o protocolo REUNITE como o HBHP foram já descritos no capítulo 3, por serem considerados como bons candidatos à implementação de encaminhamento *multicast* com QoS.

5.3 Descrição do funcionamento do DTMP

O novo protocolo DTMP (Directed Trees Multicast Protocol) é baseado no protocolo PIM-SM, diferindo dele num aspecto fundamental: a forma como são construídas as árvores de distribuição de tráfego *multicast*.

As árvores de distribuição de tráfego *multicast* podem ser de dois tipos: árvores partilhadas, ou árvores centradas nas fontes. As árvores partilhadas são centradas num nó pré-definido da topologia de rede, designado na especificação do PIM-SM por *Rendez-Vous Point - RP*, e são partilhadas pelas diferentes fontes que a usam para atingir todos os membros do grupo. As árvores centradas nas fontes, são centradas numa única fonte e usadas apenas por essa fonte para atingir os diferentes membros do grupo. Os protocolos baseados unicamente em árvores centradas nas fontes necessitam de construir uma árvore por cada fonte activa. Em relação a esta, a estratégia das árvores partilhadas poupa recursos, mas pode concentrar demasiado o tráfego nas mesmas ligações.

Quase todos os protocolos optam por uma ou outra estratégia, mas o PIM-SM propõe a utilização simultânea dos dois tipos de árvores. No PIM-SM os receptores começam por juntar-se a uma árvore partilhada, centrada num ponto pré-definido, e depois podem comutar para árvores centradas nas fontes, se acharem necessário ou conveniente. Esta mesma ideia foi usada no protocolo DTMP no sentido de lhe conferir a flexibilidade apresentada pelo protocolo PIM-SM.

Sinteticamente podemos descrever o DTMP como sendo um protocolo muito semelhante ao PIM-SM, com excepção da forma como são construídas as árvores. No PIM-SM as árvores *multicast* são *Reverse Path Trees*, ou seja, são construídas com base nos melhores caminhos inversos, isto é, desde os destinatários até ao RP ou fonte conforme se trate

de uma árvore partilhada ou centrada na fonte, respectivamente. Como já foi referido, esta estratégia não é adequada nas situações em que as ligações que constituem a topologia de rede são assimétricas ou seja no caso em que o custo do caminho entre dois nós varia conforme o sentido em que é feito o percurso. Nesse caso as árvores deverão ser construídas com base nos caminhos mais curtos entre a raíz (RP ou fonte) e os destinatários, e não ao contrário. É este o principal objectivo do DTMP: adaptar o PIM-SM às redes assimétricas construindo um protocolo em tudo semelhante, mas capaz de construir árvores directas.

5.3.1 Construção da árvore partilhada

À semelhança do que acontece no PIM-SM, no DTMP as árvores de distribuição *multicast* são construídas e mantidas através de pedidos explícitos de junção e abandono dos vários membros do grupo.

Na figura 5.1 está ilustrado o processo de junção de novos membros a um grupo *multicast* usando o protocolo DTMP.

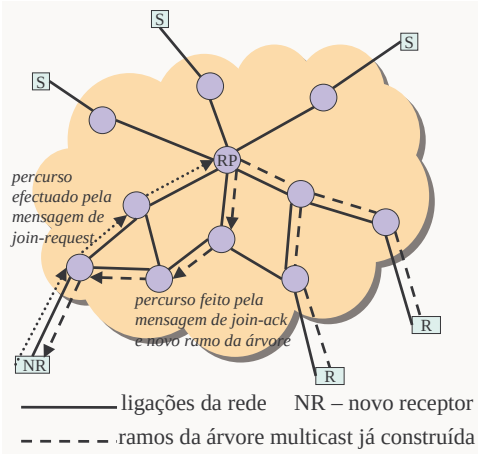


Figura 5.1: Construção da árvore partilhada segundo o protocolo DTMP

Inicialmente é proposta a utilização de uma árvore partilhada no sentido de possibilitar que os receptores se juntem ao grupo sem conhecerem previamente quem são as fontes e onde estão localizadas. O pedido de junção ao grupo é efectuado de forma explícita usando mensagens específicas, designadas por mensagens de *join-request*, enviadas em direcção a um *Rendez-Vous Point*. Ao contrário do que acontece no protocolo PIM-SM, no DTMP a mensagem de *join-request* não desencadeia qualquer tipo de acção ao longo do seu percurso. Os encaminhadores ao receberem uma mensagem deste tipo limitam-se a reenviá-la através do melhor caminho rumo ao RP ou à fonte, conforme se trate de um pedido de junção ao grupo ou de um pedido de junção a uma fonte. Ou seja, neste fase não é introduzida nenhuma informação de estado no encaminhadores.

Quem desencadeia o processo de junção ao grupo é naturalmente o candidato a novo

membro, através da já referida mensagem de *join-request*. Esta mensagem contém os seguintes dados:

- tipo de mensagem (join); Este campo especifica de que tipo de mensagem se trata: *join*, *ack* ou *prune*.
- endereço do Grupo (G);
- endereço do encaminhador nomeado (originador do *join-request*)
- lista de join (endereço do RP);
- lista de prune (nulo);
- WC-Bit (1); Quando está activa (ou seja, com o valor 1) esta *flag* significa que o encaminhador pretende receber o tráfego proveniente de todas as fontes registadas para o grupo G.
- RPT-Bit (1); Quando está activa, esta *flag* indica que se trata de um pedido de junção à árvore partilhada.

O *Rendez-Vous Point* ao receber uma mensagem de *join-request* responde com uma mensagem de aceitação, designada por mensagem de *join-aknowledge*. Esta mensagem é enviada de volta para o nó que requereu a junção ao grupo e deverá seguir pelo melhor caminho desde o *Rendez-Vous Point* rumo a este nó, caminho esse que tratando-se de uma topologia de rede assimétrica pode ser diferente do utilizado pela mensagem de *join-request*.

A mensagem de *join-acknowledge* é constituída pelos seguintes dados:

- tipo de mensagem (ack);
- endereço do Grupo (G);
- endereço do encaminhador que está a re-enviar a mensagem de *acknowledge*;
- lista de ack (endereço do encaminhador nomeado, ou seja, do encaminhador que originou a mensagem de *join-request*);
- lista de prune (nulo);
- WC-Bit (1);
- RPT-Bit (1);

É esta segunda mensagem que desencadeia o processo de construção de novos ramos da árvore de distribuição. Os encaminhadores ao receberem a mensagem de *join-acknowledge* actualizam as suas tabelas de encaminhamento *multicast*, inserindo juntamente com a

entrada relativa ao grupo (entrada (*,G)), as interfaces de entrada e saída usadas pela mensagem de *join-acknowledge*.

As entradas (*,G), segundo o DTMP, deverão ser constituídas pelos seguintes dados:

- endereço do Grupo (G);
- endereço da fonte (*); Quando este campo está preenchido com um "*" significa que se trata de uma entrada da árvore partilhada (*,G).
- interface de entrada (iif = interface por onde chegou a mensagem de *join-acknowledge*);
- endereço do vizinho que enviou a mensagem de *join-acknowledge*;
- lista das interfaces de saída (oiflist = interface usada para atingir o sistema terminal que requereu a junção ao grupo).
- *flags* que caracterizam o tipo de entrada:
  - RPT-Bit (1); Quando está activa, esta *flag* indica que esta entrada corresponde a informação de estado relativa à árvore partilhada.
  - SPT-Bit (0); Numa entrada (*,G) esta *flag* não tem significado.

Quando não existirem membros directamente ligados, nem nenhum interface de saída para outras redes na lista de interfaces de saída, a entrada (*,G) deve ser apagada.

5.3.2 Utilização da árvore partilhada

À semelhança do que acontece no protocolo PIM-SM, quando uma fonte deseja começar a enviar tráfego para um grupo, inicialmente deve fazê-lo através do RP do grupo. A fonte "encapsula" cada pacote de dados numa mensagem do tipo *register* e envia essa mensagem por *unicast* até ao RP. O RP "desencapsula" a mensagem e re-envia o pacote de dados através da árvore partilhada a todos os membros do grupo.

5.3.3 Construção da árvore centrada na fonte

Depois de receber pacotes de dados de determinada fonte, um receptor pode, se assim o desejar, juntar-se a uma árvore centrada nessa fonte. O processo de junção a uma árvore centrada na fonte é semelhante ao processo de junção à árvore partilhada. O receptor envia um pedido explícito de junção à fonte através de uma mensagem de *join-request*, que contém os seguintes dados:

- tipo de mensagem (join);
- endereço do Grupo (G);

- lista de join (endereço da fonte S);
- lista de prune (nulo);
- WC-Bit (0);
- RPT-Bit (0);

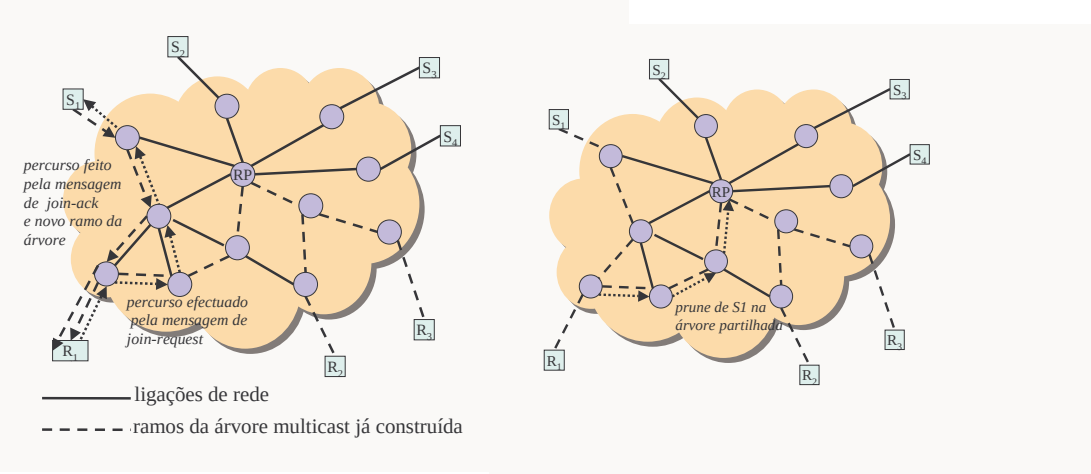
Essa mensagem viaja até à fonte sem introduzir qualquer tipo de informação de estado nos encaminhadores que estão no caminho. Ao aceitar o pedido de junção de um receptor, a fonte origina uma mensagem de *join-acknowledge* que envia em direcção ao receptor que efectuou o pedido de junção à fonte. Esta mensagem contém os seguintes dados:

- tipo de mensagem (ack);
- endereço do Grupo (G);
- endereço do próprio encaminhador (o que está a re-enviar a mensagem de *acknowledge*);
- lista de ack (endereço do encaminhador nomeado);
- lista de prune (nulo);
- WC-Bit (0);
- RPT-Bit (0);

É esta mensagem de resposta que, ao longo do seu caminho, o melhor caminho entre a fonte e o novo receptor, vai introduzir informação de estado nos encaminhadores de forma a criar um novo ramo na árvore centrada na fonte. Uma vez construído, todos os encaminhadores ao longo deste ramo, deverão deixar de receber pacotes desta fonte através da árvore partilhada, uma vez que já os estão a receber através da árvore centrada na fonte, para evitar a duplicação desnecessária de pacotes. Para isso é proposto um mecanismo semelhante ao proposto no protocolo PIM-SM e que se designa por "prune de uma fonte da árvore partilhada". A concretização deste mecanismo é bastante mais complexa no caso do DTMP, uma vez que as árvores construídas são directas e não invertidas.

Na figura 5.2 está ilustrado o processo de junção de membros de um grupo *multicast* a uma fonte usando o protocolo DTMP.

Para cada grupo *multicast*, podem coexistir árvores *multicast* centradas nas fontes, identificadas por pares (S,G), e uma árvore partilhada identificada por (*,G).



(a) R1 decide comutar para uma árvore centrada em S1

(b) Prune de S1 na árvore partilhada

Figura 5.2: Construção de uma árvore centrada na fonte segundo o protocolo DTMP

5.3.4 Utilização da árvore centrada na fonte

Quando um determinado encaminhador decide comutar para uma árvore centrada numa fonte são criadas entradas (S,G) com o SPT-Bit a zero em todos os encaminhadores no caminho entre a fonte S e o nó que originou o pedido de junção à árvore.

O facto do SPT-Bit estar a zero quer dizer que o envio de pacotes deve continuar a ser feito através das entradas (*,G) se elas existirem. Um encaminhador ao receber o primeiro pacote proveniente da fonte através da árvore centrada na fonte (comprova isso testando a interface por onde chegou o pacote, que terá que coincidir com a interface de entrada - iif, que consta na entrada (S,G)), põe o SPT-Bit a 1. Além disso faz um *prune* daquela fonte na árvore partilhada para garantir que não recebe pacotes duplicados da fonte, via árvore centrada na fonte e via árvore partilhada. Este *prune* só deverá ser feito se o nó pertencer à árvore partilhada.

A partir do momento que a entrada (S,G) tem o SPT-Bit a 1 deixam de ser re-enviados os pacotes provenientes da fonte S, que chegarem via árvore partilhada.

5.3.5 Destruição de árvores no DTMP

Quando os membros de um grupo *multicast* pretendem abandoná-lo é necessário implementar mecanismos adicionais para remover informação de estado nos encaminhadores e eventualmente "podar" ramos das árvores de distribuição de tráfego *multicast*.

Como as árvores são construídas de forma directa e não de forma invertida como no caso do PIM-SM, o mecanismo de abandono usado no PIM-SM teve que ser re-pensado.

Uma hipótese seria implementar um mecanismo de actualização periódica da informação de estado. Nesse caso, quando um membro pretendesse abandonar o grupo simplesmente teria que parar de enviar as mensagens de *join-refresh* periódicas. Se os encaminhadores deixassem de receber as mensagens de *acknowledge-refresh* durante um determinado período de tempo a informação era dada como obsoleta e simplesmente removida. Na implementação do DTMP optou-se antes por implementar a funcionalidade de abandono do grupo através de mensagens explicitas de *prune-request*.

A destruição das árvores de distribuição *multicast* ou simplesmente de alguns dos seus ramos é o resultado de pedidos de abandono explícitos por parte dos membros do grupo.

Um pedido de abandono de um membro implica a saída do membro da árvore partilhada, ou seja, a "poda" de todos os ramos que estão na árvore partilhada com o único propósito de fazer chegar o tráfego *multicast* a este membro. Se durante a sessão, o membro comutou para uma árvore ou mais árvores centradas nas fontes então torna-se também necessário "podar" os ramos dessas árvores que ligam o membro do grupo às respectivas fontes.

Um pedido de abandono dá origem a um mensagem de *prune-request* na árvore partilhada e uma mensagem de *prune-request* por cada árvore centrada na fonte à qual o membro do grupo se tenha juntado.

O processo tem obrigatoriamente que ser diferente do processo de "poda" previsto no PIM-SM, uma vez que no DTMP as árvores são directas e não árvores invertidas. Assim, as mensagens de *prune-request* no DTMP são enviadas em direcção ao vizinho imediatamente acima na árvore. Por isso é que é necessário manter o seu endereço nas entradas da tabela de encaminhamento.

Um encaminhador ao receber uma mensagem deste tipo começa por apagar da lista de interfaces de saída da respectiva entrada ((*,G) ou (S,G), conforme se trate de um *prune* na árvore partilhada ou de um *prune* numa árvore centrada numa fonte), a interface usada para atingir o nó que originou a mensagem de *prune-request*. Depois disso, caso a lista de interfaces da saída tenha ficado vazia, re-envia a mensagem de *prune-request* em direcção ao nó vizinho que consta da entrada (*,G) ou (S,G) conforme se trate de um *prune* na árvore partilhada ou de um *prune* numa árvore centrada numa fonte, respectivamente. Além disso, as entradas deverão ser apagadas, o que alivia os encaminhadores de informação de estado desnecessária, e destrói definitivamente os ramos das árvores *multicast*.

As mensagens de *prune-request* podem também ser originadas na situação em que os nós comutam da árvore partilhada para árvores centradas na fontes. Como já foi referido nessas situações é necessário evitar que os pacotes que já estão a ser transmitidos por árvores centradas nas fontes continuem a ser difundidos através da árvore partilhada. Para isso utilizam-se também mensagens de *prune-request*, se bem que especiais: são mensagens de *prune-request* de fontes na árvore partilhada e distinguem-se dos outros *prunes* de fontes por terem a *flag* RPT-Bit activa. É esta *flag* que indica aos diferentes encaminhadores que se trata de um *prune* de uma fonte da árvore partilhada, e desta forma, se for necessário re-enviar a mensagem de *prune-request* esse re-envio é feito em direcção ao vizinho que

consta na entrada (*,G) e não ao que consta na entrada (S,G).

As mensagens de *prune-request* são constituídas pelos seguintes dados:

- tipo de mensagem (prune); Este campo especifica de que tipo de mensagem se trata, neste caso deverá ser preenchido com *prune*.
- endereço do Grupo (G);
- lista de join (nulo);
- lista de prune; Se se tratar de um *prune* de uma fonte na árvore partilhada, ou apenas de um *prune* numa árvore centrada na fonte este campo deverá ser preenchido com a fonte em causa, senão deverá conter o endereço do RP.
- O WC-Bit e o RPT-Bit são preenchidos a 0 ou a 1 de acordo com as seguintes condições:
  - WC-Bit(1), RPT-Bit(1) caso se trate de um *prune* na árvore partilhada
  - WC-Bit(0), RPT-Bit(0) caso se trate de um *prune* numa árvore centrada na fonte
  - WC-Bit(0), RPT-Bit(0) caso se trate do *prune* de uma fonte na árvore partilhada.

5.3.6 Informação de estado mantida nos nós

Para cada grupo *multicast*, podem coexistir árvores *multicast* centradas nas fontes, identificadas por pares (S,G), e uma árvore partilhada identificada por (*,G). Para cada uma destas árvores, o estado a manter é o seguinte:

- *S* - endereço da fonte ou "*" se se trata da árvore partilhada
- *G* - endereço do grupo
- *iif* - interface de entrada por onde os pacotes de dados são esperados (é preciso verificar este campo antes de os encaminhar, para evitar ciclos!)
- *neighbor* - endereço do nó que está imediatamente acima na árvore *multicast* (usado para efectuar os *prunes*).
- *oif_list* - lista de todos os interfaces de saída por onde os pacotes de dados devem ser encaminhados
- *Flags* - *flags* de caracterização da rota, nomeadamente o SPT-Bit e o RPT-Bit.

O encaminhamento *multicast* é feito com base nesta informação. A informação de estado é produzida pela acção da estratégia de encaminhamento descrita nas secções anteriores. Na secção seguinte descreve-e o uso que o DTMP faz dessa informação para proceder ao encaminhamento de pacotes *multicast*.

5.3.7 Re-envio dos pacotes de dados em cada nó

Os pacotes de dados são encaminhados de uma forma semelhante ao proposto pelo protocolo PIM-SM, já detalhado na secção 4.1.7.

5.4 Implementação do DTMP

Para implementar o protocolo DTMP[8], usou-se o Network Simulator (NS)[5]. Como já foi referido na secção 4.2.2, o NS inclui implementações de alguns protocolos de encaminhamento *multicast*. Entre outras, inclui uma versão de um protocolo muito semelhante ao PIM-SM, com a desvantagem de ser totalmente centralizada e por essa razão não permitir a avaliação de algumas métricas.

Por esse motivo foi implementada uma versão distribuída do protocolo PIM-SM, implementação essa que foi descrita no capítulo 4. Essa implementação serviu de ponto de partida para a concretização do protocolo DTMP no NS e também para comparar os resultados obtidos com os do PIM-SM.

A implementação do DTMP desenvolvida no NS é constituída por (1) um agente de encaminhamento que implementa a construção das árvores de distribuição *multicast* de acordo com a estratégia seguida pelo DTMP e por (2) um novo classificador que implementa o re-envio do tráfego segundo o DTMP. Estas duas componentes são descritas com maior detalhe na próxima secções deste capítulo.

5.4.1 Implementação de um agente de encaminhamento DTMP

A seguir são descritos da forma mais detalhada possível os métodos implementados. Este métodos são basicamente os mesmo que foram construídos para implementar o agente de encaminhamento PIM-SM, embora a maior parte deles tenha sido re-escrita no sentido de implementar a estratégia de encaminhamento do protocolo DTMP. Foram acrescentados alguns métodos, nomeadamente, o método **recv-ack** que é o método responsável pela criação de novos ramos na árvore de distribuição *multicast*.

- **Método join-group**

  Este método deverá ser invocado por um agente quando este decide juntar-se a um grupo ou quando decide comutar da árvore partilhada para uma árvore centrada na fonte.

Se for invocado com apenas um argumento, o método depreende que o agente pretende juntar-se ao grupo (ou seja, à árvore partilhada) e invoca o método **join-rpt**.

Se for invocado com dois argumentos quer dizer que o agente já faz parte do grupo e pretende apenas comutar da árvore partilhada para uma árvore centrada na fonte. Neste caso o método invoca o método **join-spt**.

- **Método join-rpt**

  Este método deverá ser invocado quando um agente decide juntar-se a um grupo. Podem ocorrer três situações distintas:

  - O nó ainda não faz parte da árvore partilhada, ou seja, ainda não existe nenhuma entrada (*,G) na tabela de encaminhamento *multicast* do nó. Nesse caso, a entrada tem que ser criada embora fique marcada como inactiva até ser recebida a mensagem de *join-acknowledge*. A interface de entrada (campo *iif*) é preenchida com -1, e o campo *neighbor* com a identificação do próprio nó, uma vez que só depois de ser recebida a mensagem de *join-acknowledge* é que estes campos poderão ser preenchidos correctamente. O RPT-Bit é preenchido com 1 e o SPT-Bit é preenchido com 0. Depois de criada a nova entrada é necessário enviar um pedido de *join-request* para o RP.
  - Já existe uma entrada (*,G), mas a lista de interfaces de saída está vazia. Neste caso também é necessário enviar um pedido de *join* para o RP.
  - Já existe uma entrada (*,G) e a lista de interfaces de saída não está vazia. Nesta caso, nem sempre é necessário enviar um *join*, como veremos adiante.

  Em qualquer das situações o agente que fez o *join* ao grupo tem que ser incluído na lista de interfaces de saída (*oif_list*) da entrada (*,G). Isso já é feito no método **join-group** da classe **Node**, através da actualização da variável **replicator_**.

  Depois é necessário verificar se existem entradas do tipo (S,G). Se existirem, o agente tem que ser acrescentado também às respectivas listas de interfaces de saída, o que também é feito pelo método **join-group** da classe **Node**.

  Se existir a entrada (*,G) e algumas das entradas (S,G) tiverem o RPT-Bit a 1, quer dizer que já existem fontes que não estão a usar a árvore partilhada. Só neste caso, precisamos de propagar o *join-request* pela árvore partilhada acima até ao RP para garantirmos que esta situação é remediada para o novo membro do grupo, que como é óbvio deseja receber de todas as fontes através da árvore partilhada.

  Em qualquer das situações descritas, sempre que é necessário enviar um pedido de *join* para o nó vizinho no caminho mais curto até ao RP, o método **send-ctrl** deverá ser invocado com os seguintes argumentos:

  - a string "join" para indicar que se trata de um pedido de junção a uma árvore;
  - o endereço do RP do grupo;
  - o endereço do grupo;

– endereço do nó que vai fazer o *join* (candidato a novo membro do grupo);
– o WC-Bit preenchido a 1;
– o RPT-Bit preenchido a 1;

O processo de *join* à árvore partilhada está esquematizado na figura 5.3, onde as variáveis e as *flags* têm o mesmo significado do que as definidas no PIM-SM[7].

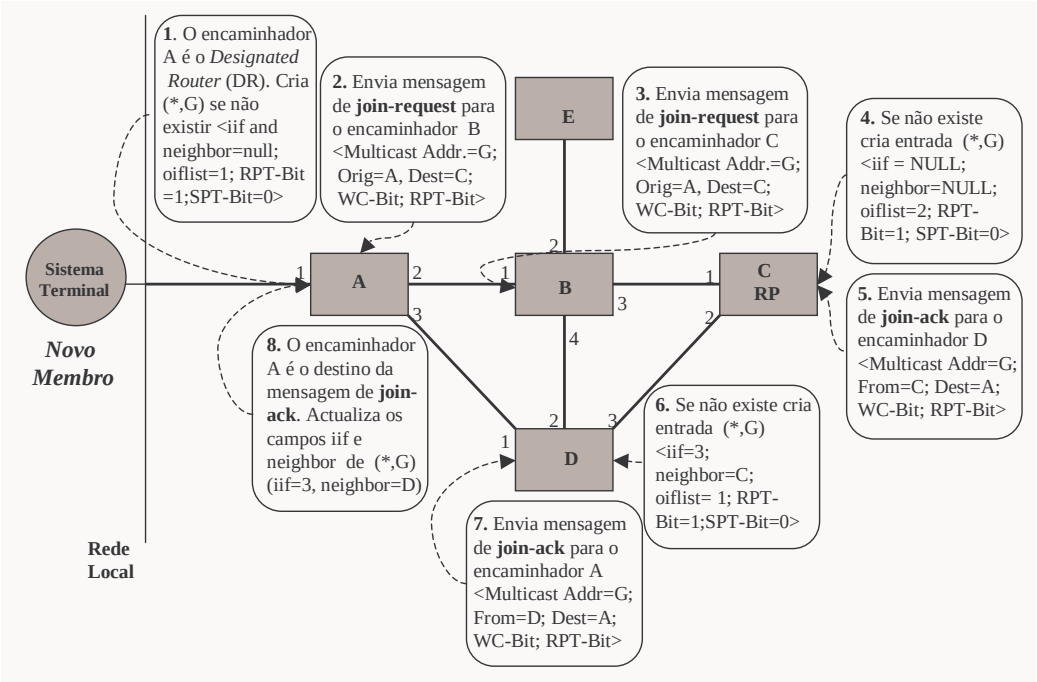


Figura 5.3: Implementação da construção da árvore partilhada segundo o protocolo DTMP. As acções estão numeradas pela ordem em que ocorrem

- **Método join-spt**

  Este método deverá ser invocado quando um agente decide comutar para uma árvore centrada na fonte. Neste caso, o agente poderá pretender continuar ligado à árvore partilhada excepto para uma determinada fonte. Podem ocorrer duas situações distintas:

  – Não existe uma entrada (S,G) na tabela de encaminhamento do nó, sendo S a fonte para a qual o agente decide comutar, ou seja, o nó ainda não faz parte de uma árvore centrada na fonte S. Neste caso, é necessário criar a entrada (S,G). A interface de entrada (campo *iif*) deverá ficar igual a -1 e o campo *neighbor* igual à identificação do próprio nó, isto até chegar a mensagem de *join-acknowledge* pois só nessa altura é que este campos poderão ser correctamente preenchidos. A lista de interfaces de saída deve ficar igual à lista de interfaces de saída da

entrada (*,G). O SPT-Bit da nova entrada deverá ser posto a zero, assim como o RPT-Bit. Depois de criada a entrada (S,G) deve ser enviado um pedido de *join-request* para o próximo nó no caminho mais curto até à fonte S.

- Já existe uma entrada do tipo (S,G). É necessário verificar se ela tem o RPT-Bit a 0 ou a 1. Se tiver o RPT-Bit a 1 é necessário pô-lo a 0, alterar o campo relativo à interface de entrada (*iif*) para -1 e o campo *neighbor* para a identificação do próprio nó e por fim é necessário enviar um pedido de *join-request* para o nó vizinho no melhor caminho até à fonte. Se a entrada (S,G) já existir e tiver o RPT-Bit a 0, só é necessário enviar um pedido de *join* para o nó vizinho no melhor caminho até à fonte se a lista de interfaces de saída estiver vazia.

Em qualquer das situações descritas sempre que é necessário enviar um pedido de *join-request* para o nó vizinho no caminho mais curto até à fonte, o método **send-ctrl** deverá ser invocado com os seguintes argumentos:

- a string "join" para indicar que se trata de um pedido de junção a uma árvore;
- o endereço da fonte para a qual se quer comutar;
- o endereço do grupo;
- o endereço do nó que pretende juntar-se à fonte
- o WC-Bit preenchido com 0;
- o RPT-Bit preenchido com 0;

O processo de comutação de uma árvore partilhada para uma árvore centrada na fonte está esquematizado na figura 5.4.

- **Método recv-join**

  Se se tratar de um encaminhador que está algures no caminho entre o nó que originou o *join* e o RP ou a fonte, este método limita-se a reenviar o pedido de *join*.

  Se foi o RP (ou a fonte) que recebeu o *join* então é necessário criar ou actualizar a entrada (*,G) (ou a entrada (S,G)), e invocar o método **send-ctrl** com os seguintes argumentos:

  - a string "ack" para indicar que se trata de uma mensagem de *join-acknowledge*
  - o endereço do RP do grupo ou endereço da fonte conforme se trate de um pedido de junção ao grupo ou de um pedido de junção a uma fonte, respectivamente;
  - o endereço do candidato a novo membro ou endereço do nó que pretende comutar para uma árvore centrada na fonte;
  - o endereço do grupo;
  - o WC-Bit preenchido a 1 ou a 0 conforme se trate de um pedido de junção ao grupo ou de um pedido de junção a uma fonte, respectivamente;

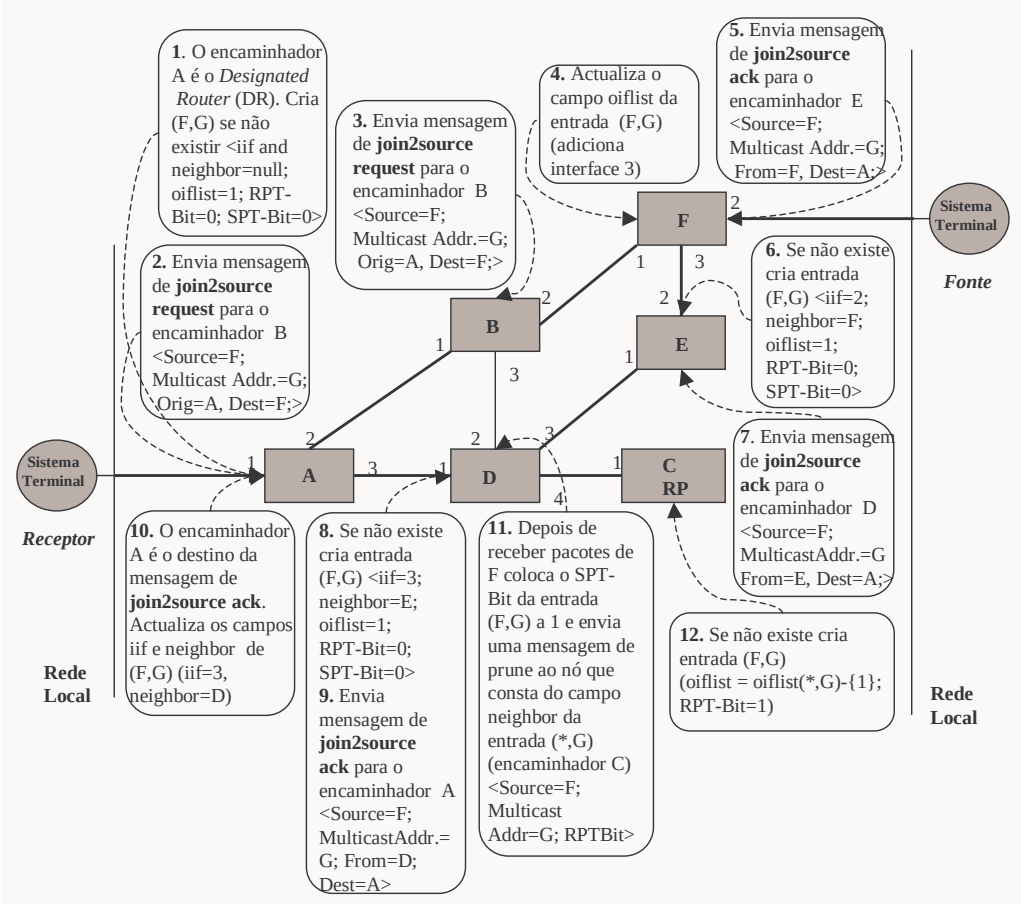


Figura 5.4: Processo de comutação da árvore partilhada para uma árvore centrada na fonte

– o RPT-Bit preenchido a 1 ou a 0 conforme se trate de um pedido de junção ao grupo ou de um pedido de junção a uma fonte, respectivamente;

- **Método recv-ack**

  Este método é invocado sempre que um nó recebe uma mensagem de *join-acknowledge* proveniente do RP ou de uma determinada fonte. Para distinguir um do outro é necessário testar o WC-Bit e o RPT-Bit. Se ambos estiverem a 1 trata-se de um *acknowledge* proveniente do RP e que conduzirá à construção de um ramo da árvore partilhada. Senão trata-se de um *acknowledge* a um pedido de junção a uma fonte.

  – Recepção de um *join-acknowledge* proveniente do RP

  Neste caso é necessário testar se o nó já pertence à árvore partilhada, ou seja se já existe uma entrada (*,G) na tabela de encaminhamento do nó.

  Se não existir, é necessário criá-la. A interface de entrada a colocar na nova

entrada (*,G) deverá ser a interface por onde chegou a mensagem de *join-acknowledge*. Na lista de interfaces de saída deve ser colocada a interface usada para atingir o nó que originou o *join* à árvore partilhada através do melhor caminho.

Se a entrada (*,G) já existir só é necessário acrescentar a melhor interface para atingir o nó que originou o *join* à lista de interfaces de saída da entrada (*,G).

Em qualquer dos casos é necessário re-enviar a mensagem de *join-acknowledge* para o nó vizinho em direcção ao nó que originou o *join*.

Depois é necessário verificar se existe alguma entrada do tipo (S,G) relativa ao mesmo grupo a que se fez o *join*. Se existir é necessário acrescentar a melhor interface para atingir o nó que originou o *join* à lista de interfaces de saída da entrada (S,G), quer a entrada tenha o RPT-Bit a 1 quer não.

Ao acrescentarmos a nova interface a uma entrada (S,G) com RPT a 1, a lista de interfaces de saída dessa entrada poderá ficar igual à lista de interfaces de saída da entrada (*,G). Nesse caso, a entrada (S,G) com RPT a 1 deverá ser apagada.

– Recepção de um *join-acknowledge* proveniente de uma fonte

Antes de mais é necessário testar se o nó já pertence ou não a uma árvore centrada na fonte, ou seja se já existe uma entrada (S,G) na tabela de encaminhamento *multicast* do nó.

Se não existir, é necessário criá-la. A interface de entrada deverá ficar igual à interface por onde chegou a mensagem de *join-acknowledge*. A lista de interfaces de saída deve ficar igual à lista de interfaces de saída da entrada (*,G), se esta existir, acrescentando a interface usada para atingir o nó que originou o *join* à fonte através do melhor caminho. O SPT-Bit da nova entrada deverá ser posto a zero, assim como o RPT-Bit. Depois de criada a entrada (S,G) a mensagem de *join-acknowledge* deve ser re-enviada para o próximo nó no caminho mais curto até ao nó que originou o *join* à fonte.

Senão existir nenhuma entrada (*,G) a lista de interfaces de saída da nova entrada deverá ficar apenas com a interface usada para atingir o nó que originou o *join* à fonte através do melhor caminho. O SPT-Bit e o RPT-Bit da nova entrada deverão na mesma ser postos a zero e a mensagem de *join-acknowledge* deve ser re-enviado para o próximo nó no caminho mais curto até ao nó que originou o *join* à fonte.

Se já existir uma entrada do tipo (S,G), é necessário verificar se ela tem o RPT-Bit a 0 ou a 1. Se tiver o RPT-Bit a 1 é necessário pô-lo a 0, substituir a interface de entrada pela interface por onde chegou a mensagem de *join-acknowledge*, bem como o campo *neighbor* pelo último nó que re-enviou a mensagem de *join-acknowledge*, e acrescentar à lista de interfaces de saída a interface usada para atingir o nó que originou o *join* à fonte através do melhor caminho. Por fim é necessário re-enviar mensagem de *join-acknowledge* para o nó vizinho no melhor caminho até o nó que originou o *join* à fonte.

Se a entrada (S,G) já existir e tiver o RPT-Bit a 0 é necessário colocar na lista de interfaces de saída a interface usada para atingir o nó que originou o *join* à fonte através do melhor caminho e re-enviar o *join* para o nó vizinho em direcção ao nó que originou o *join* à fonte.

- **Método handle-SetSPTBit**

  Este método é invocado pelo classificador do DTMP quando pretende fazer o *prune* de uma fonte na árvore partilhada.

  Como já foi referido, quando um determinado agente quer comutar da árvore partilhada para uma árvore centrada numa fonte, é criada pelo método **recv-ack** uma entrada do tipo (S,G) com o SPT-Bit a 0. Este bit mantém-se a zero temporariamente até que o tráfego da fonte S começa a fluir através da árvore centrada na fonte. Quando isso acontece o bit é posto a 1 para tornar efectiva a entrada (S,G). Essa alteração do SPT-Bit de 0 para 1 é feita pelo classificador do DTMP, assim que recebe o primeiro pacote vindo da fonte S pela interface esperada. Depois é necessário fazer um *prune* daquela fonte na árvore partilhada para evitar que o tráfego proveniente daquela fonte chegue ao nó também através da árvore partilhada. Esse *prune* é diferente dos outros porque tem o RPT-Bit a 1, apesar de ter o endereço de uma fonte na lista de *prune*, e o WC-Bit a 0. Este pedido deve ser re-enviado em direcção ao RP e não em direcção à fonte. Este *prune* só deverá ser enviado se a árvore pertencer à árvore partilhada.

- **Método leave-group**

  Este método deverá ser invocado quando um determinado agente associado a um nó decide abandonar um grupo ou quando decide apenas deixar de receber tráfego através de uma árvore centrada numa fonte para a qual o agente comutou.

  Se for invocado com apenas um argumento, o método depreende que o agente pretende abandonar o grupo e invoca não só o método **leave-rpt** para abandonar a árvore partilhada, mas também o método **leave-spt** tantas vezes quantas as necessárias para ser removido de todas as árvores centradas nas fontes para as quais eventualmente tenha comutado.

  Se for invocado com dois argumentos quer dizer que o agente pretende apenas deixar de receber tráfego de uma determinada árvore centrada numa fonte para a qual comutou. Nesse caso o método invoca o método **leave-spt**.

- **Método leave-rpt**

  Para remover o agente da árvore partilhada é necessário apagá-lo da lista de interfaces da saída da entrada (*,G). Isto é feito pelo método **leave-group** da classe **Node**.

  Se a lista de interfaces de saída da entrada (*,G) ficar vazia, a entrada poderá ser apagada e deverá ser gerada uma mensagem de *prune-request*. Um *prune* deste tipo tem o RPT-Bit e o WC-Bit ambos a 1. Alem disso o endereço que aparece na lista de *prunes* é o endereço do RP do grupo.

No caso do PIM-SM, esta mensagem de *prune-request* seria enviada para o próximo nó no caminho mais curto para o RP. No entanto como já foi referido na secção 5.3.5 deste capítulo o mecanismo de *prune* no DTMP teve que ser alterado pelo facto deste construir árvores directas e não invertidas. Assim, a mensagem de *prune-request* é enviada em direcção ao vizinho imediatamente acima na árvore de distribuição *multicast*. A identificação deste nó consta das próprias entradas da tabela de encaminhamento *multicast*, neste caso em concreto o campo *neighbor* da entrada (*,G).

- **Método leave-spt**

  Este método é invocado pelo método **leave-group** para retirar determinado agente de todas as árvores centradas nas fontes para as quais eventualmente tenha comutado.

  Para remover um agente de uma árvore centrada numa determinada fonte S é necessário apagá-lo da lista de interfaces da saída da entrada (S,G). O agente deverá ser apagado quer a entrada tenha o RPT a 1 ou a 0. Isto é feito pelo método **leave-group** da classe **Node**.

  Se a entrada (S,G) tiver o RPT-Bit a 1, a entrada (S,G) não deverá ser apagada, mesmo que a lista de interfaces de saída fique vazia. Se a lista de interfaces de saída ficar vazia dever-se-à enviar um *prune* da fonte S na árvore partilhada. Este *prune* é semelhante ao enviado no método **handle-SetSPTBit**, ou seja, apesar de ter o endereço de uma fonte na lista de *prune* deve ser propagado ao longo da árvore partilhada. Para que isso aconteça o RPT-Bit deverá ser posto a 1 e o WC-Bit a 0.

  Se a entrada (S,G) tiver o RPT-Bit a 0 e a lista de interfaces de saída ficar vazia, a entrada poderá ser apagada e o *prune* deverá ser re-enviado ao longo da árvore centrada na fonte ou seja, a mensagem de *prune-request* é enviada para o vizinho imediatamente acima na árvore de centrada na fonte (o campo *neighbor* da entrada (S,G)). Um *prune* deste tipo tem o RPT-Bit e o WC-Bit ambos a 0 e o endereço que aparece na lista de *prunes* é o endereço da fonte.

- **Método recv-prune**

  Este método é invocado sempre que um nó recebe uma mensagem de *prune-request*. Existem basicamente três *prunes* possíveis: o *prune* de um nó na árvore partilhada, o *prune* de um nó numa árvore centrada na fonte e o *prune* de uma fonte na árvore partilhada. Para os distinguir é necessário testar o WC-Bit e o RPT-Bit.

  - Recepção de um *prune* de um nó na árvore partilhada

    Um *prune* deste tipo tem o RPT-Bit e o WC-Bit ambos a 1. Além disso o endereço que aparece na lista de *prunes* é o endereço do RP do grupo. Ao receber um *prune* deste tipo é necessário verificar antes de mais se existe alguma entrada (*,G). Se existir é necessário actualizar a lista de interfaces de saída da entrada (*,G), apagando uma interface de forma a "podar"o ramo respectivo. No caso do PIM-SM a interface a apagar é a interface por onde chegou o *prune*. No caso do DTMP, como as árvores de distribuição de tráfego são construídas

de forma directa, a interface a apagar da lista de interfaces de saída deverá ser a interface usada pelo encaminhador para atingir o nó que enviou o *prune*. No caso de estarmos perante uma rede assimétrica, essa interface não tem de ser a mesma por onde chegou a mensagem de *prune-request*.

Se, depois de apagar esta interface, a lista de interfaces de saída ficar vazia, a entrada deverá ser apagada e a mensagem de *prune-request* deverá ser re-enviado ao longo da árvore partilhada, ou seja, deverá ser re-enviada para o nó que está imediatamente acima dele na árvore partilhada (campo *neighbor* da entrada (*,G)).

Depois é necessário verificar se existem entradas (S,G) onde consta a interface apagada da entrada (*,G). Se existirem, esta interface também deve ser apagada da lista de interfaces de saída das entradas (S,G). Se ao apagar a interface, a lista de interfaces de saída ficar vazia há que considerar duas situações possíveis:

* Se a entrada (S,G) tiver o RPT-Bit a 0 então é necessário apagar a entrada e propagar o *prune* ao longo da árvore centrada na fonte S (ou seja, re-enviar a mensagem de *prune-request* para o vizinho identificado através do campo *neighbor* da entrada (S,G)), tendo o cuidado de colocar o WC-Bit e o RPT-Bit, ambos a zero.
* Se a entrada (S,G) tiver o RPT-Bit a 1 então não se pode apagar a entrada (S,G), mas é necessário propagar o *prune*, desta vez ao longo da árvore partilhada (ou seja, enviar a mensagem de *prune-request* para o vizinho identificado através do campo *neighbor* da entrada (*,G)), tendo o cuidado de colocar o RPT-Bit a 1 e o WC-Bit a 0.

– Recepção de um *prune* de um nó na árvore centrada na fonte

Um *prune* deste tipo tem o RPT-Bit e o WC-Bit ambos a 0. O endereço que aparece na lista de *prunes* é o endereço da fonte. Ao receber um *prune* deste tipo deve-se verificar se existe alguma entrada (S,G). Se existir a interface usada para atingir o nó que enviou o *prune* deve ser apagada da lista de interfaces de saída da entrada (S,G). Depois têm que ser consideradas duas situações distintas:

* Se a entrada (S,G) tiver o RPT-Bit a 0 então é necessário apagar a entrada e propagar o *prune* ao longo da árvore centrada na fonte S (ou seja, re-enviar a mensagem de *prune-request* para o vizinho identificado através do campo *neighbor* da entrada (S,G)), tendo o cuidado de colocar o WC-Bit e o RPT-Bit, ambos a zero.
* Se existir uma entrada (*,G) e a entrada (S,G) tiver o RPT-Bit a 1, então não se pode apagar a entrada (S,G), mas é necessário propagar o *prune*, desta vez ao longo da árvore partilhada (ou seja, enviar a mensagem de *prune-request* para o vizinho identificado através do campo *neighbor* da entrada (*,G)), tendo o cuidado de colocar o RPT-Bit a 1 e o WC-Bit a 0.

– Recepção de um *prune* de uma fonte na árvore partilhada

Um *prune* deste tipo tem o RPT-Bit a 1 e o WC-Bit a 0. O endereço que aparece na lista de *prunes* é o endereço da fonte da qual não se quer receber mais pacotes através da árvore partilhada. Ao receber um *prune* deste tipo deve-se verificar se existe alguma entrada (S,G). Se existir, a interface usada para atingir o nó que enviou o *prune* deve ser apagada da lista de interfaces de saída da entrada (S,G). Depois têm que ser consideradas a mesmas situações que já foram consideradas no caso dos outros *prunes*, ou seja:

* Se a entrada (S,G) tiver o RPT-Bit a 0 então é necessário apagar a entrada e propagar o *prune* ao longo da árvore centrada na fonte S (ou seja, re-enviar a mensagem de *prune-request* para o vizinho identificado através do campo *neighbor* da entrada (S,G)), tendo o cuidado de colocar o WC-Bit e o RPT-Bit, ambos a zero.
* Se existir uma entrada (*,G) e a entrada (S,G) tiver o RPT-Bit a 1, então não se pode apagar a entrada (S,G), mas é necessário propagar o *prune*, desta vez ao longo da árvore partilhada (ou seja, enviar a mensagem de *prune-request* para o vizinho identificado através do campo *neighbor* da entrada (*,G)), tendo o cuidado de colocar o RPT-Bit a 1 e o WC-Bit a 0.

Se existir uma entrada (*,G), mas não existir uma entrada (S,G), tem que ser criada uma entrada (S,G) com o RPT-Bit a 1. A lista de interfaces de saída é copiada da lista de interfaces de saída da entrada (*,G), apagando-se desta lista a interface usada para atingir o nó que enviou o *prune*.

5.4.2 Implementação de um classificador DTMP

O classificador DTMP implementado é basicamente igual ao classificador PIM-SM, já descrito na secção 4.3.2. Deriva também da classe **MCastClassifier** (classe do classificador *multicast* standard), mas além de lhe acrescentar as flags (RPT-Bit e SPT-Bit) acrescenta-lhe também o campo *neighbor* que contém a identificação do nó que está imediatamente acima na árvore de distribuição de tráfego.

5.5 Teste da Implementação

Para implementar e avaliar o DTMP usou-se o Network Simulator, versão 2.18b. Os resultados obtidos foram comparados com resultados obtidos em situações idênticas pela implementação do protocolo PIM-SM desenvolvida e já descrita no capítulo. 4

Tanto o DTMP como o PIM-SM têm como objectivo a construção de árvores de distribuição de tráfego *multicast*. Como tal na comparação entre estes dois protocolos interessará obter indicadores da qualidade da(s) árvore(s) geradas e indicadores da sobrecarga introduzida pelo protocolo na geração dessa(s) mesma(s) árvores.

Existem diferentes alternativas para avaliar a qualidade das árvores de distribuição

multicast. Uma das mais simples consiste em contar o número de ligações da topologia que pertencem à árvore. Outra possibilidade consiste em contar o número de réplicas de pacotes que são originadas pelos diferentes nós quando re-enviam os pacotes *multicast* ao longo da árvore de distribuição de tráfego. Cada nó não deverá receber pacotes duplicados de nenhuma das fontes, pelo que o número de réplicas originadas deverá corresponder ao número de interfaces de saída diferentes usadas para atingir os diferentes destinatários. Desta forma estes duas métricas deverão representar o mesmo valor desde que seja utilizada uma única árvore. No entanto, quando o PIM-SM e o DTMP são utilizados, é habitual aparecer mais do que uma árvore, uma vez que os membros do grupo habitualmente comutam para árvore centradas nas fontes ao fim de algum tempo. Por essa razão, a métrica que conta o número de réplicas é melhor para medir a qualidade das árvores pois conta os recursos que estão a ser usados efectivamente. Outra diferença entre estas duas métricas está relacionada com a forma como os diferentes valores são calculados. Enquanto o número de réplicas só pode ser determinado durante a transferência de dados, o número de ligações pode ser calculado na altura em que as entradas são inseridas ou apagadas das tabelas de encaminhamento *multicast*.

Nenhuma destas duas métricas tem em consideração as características das ligações, e não podem ser usadas para determinar como é que o processo de construção das árvores lida com as assimetrias das ligações. Assim, em vez de se utilizar apenas o número de réplicas, é usada uma métrica que combina este número com o custo associado a cada ligação que cada réplica atravessa. Este foi a primeira métrica que foi calculada. Calculou-se também o número total de ligações para determinar se o DTMP constrói árvores maiores ou menores do que o PIM-SM.

A avaliação das árvores geradas é normalmente feita recorrendo a gráficos que mostram como varia o custo médio da árvore com o número de membros que fazem parte do grupo. Isto porque sempre que um membro sai ou entra do grupo a árvore modifica-se em conformidade e muda de custo. Dado que os membros podem juntar-se ou abandonar o grupo a qualquer momento, a cardinalidade do mesmo vai mudando ao longo do tempo, podendo haver valores de custo diferentes para o mesmo número de elementos. Esta situação leva a que se tenha de considerar um valor de custo médio da árvore, calculado com base em todos os valores de custo registados para um dado número de membros.

É de realçar no entanto que qualquer uma destas métricas analisa a árvore ou árvores de distribuição de tráfego *multicast* como um todo. Outro aspecto que pode ser importante considerar é a análise dos caminhos individuais que ligam as raízes das árvores (RP ou fontes) a cada um dos receptores. Em relação aos caminhos é possível obter não só a média dos tamanhos dos diferentes caminhos que constituem a árvore, como também a média dos respectivos custos. Estas duas métricas também foram calculadas em função do número de membros que fazem parte do grupo.

Outro aspecto que interessa medir é a sobrecarga introduzida pelos protocolos na geração das árvores. Dada a natureza dos protocolos em causa, PIM-SM e DTMP, pode-se usar como indicador a sobrecarga de mensagens de controlo que os nós da rede têm de processar para construir e manter as árvores de distribuição *multicast*. As mensagens

de controlo são de diferentes tipos (*join*, *prune* e *ack*), e podem ser contadas à entrada ou à saída de cada nó. O resultado das contagens apresenta-se normalmente em valores médios por nó em função do número de operações que alteram de alguma forma a árvore de difusão: abandono do grupo, adesão de novos membros e comutações da árvore partilhada para árvores centradas na fonte. O ficheiro de *trace* normal do NS permite calcular estes valores.

5.5.1 Cenários de Simulação

Para testar o DTMP e compará-lo com o PIM-SM usaram-se as mesmas duas topologias que se usaram para testar a implementação do PIM-SM: uma rede típica de um grande ISP com 18 nós e uma topologia de 100 nós gerada aleatoriamente. Principalmente no primeiro cenário de simulação foram introduzidas algumas variantes para ser possível concluir qual o verdadeiro impacto das assimetrias nos resultados obtidos.

5.5.1.1 Topologia de rede típica de um ISP

A primeira topologia de rede usada é uma rede típica de um grande ISP[65] e está ilustrada na figura 5.5.

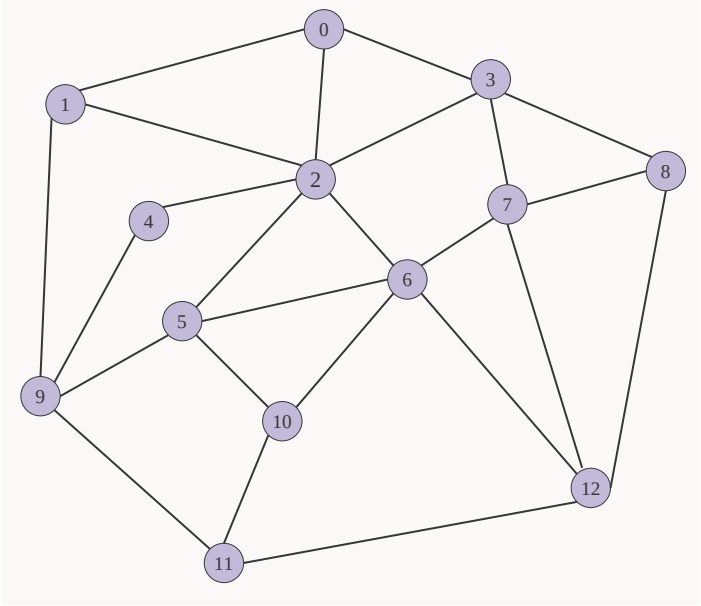


Figura 5.5: Topologia de rede típica de um ISP

Esta topologia inclui 18 nós e 30 ligações. Associado a cada ligação existem dois custos de utilização, uma em cada direcção. Cada custo é um inteiro escolhido aleatoriamente em diferentes intervalos. Neste primeiro cenário de simulação é considerado apenas um grupo

e três fontes fixas nos nós 3, 9 e 15. É assumido que existe um potencial membro do grupo em cada nó da topologia. Para cada simulação o RP é escolhido aleatoriamente entre todos os nós da topologia. No inicio de cada simulação não existe nenhum membro no grupo. Após um período inicial, em que as três fontes começam a transmitir, 50% dos potenciais membros aleatoriamente escolhidos entre os 18 possíveis, juntam-se em instantes de tempo distintos à árvore partilhada com raíz no RP.

Depois de estarem todos os membros ligados, todos eles comutam para árvores centradas nas fontes, em instantes de tempo diferentes. Começam por comutar todos para uma árvore centrada na fonte 3, depois comutam também para uma árvore centrada na fonte 9 e finalmente comutam para uma árvore centrada na fonte 15. Este cenário, com uma árvore partilhada e três árvores centradas nas fontes é mantido até ao fim da simulação. Antes da simulação terminar, todos os membros abandonam o grupo.

Foram feitas diversas simulações e análises com este cenário. Na primeira experiência o custo das ligações é um inteiro escolhido aleatoriamente no intervalo [1, 5]. Na segunda experiência é um inteiro escolhido no intervalo de [1, 10] e finalmente na terceira experiência é um inteiro escolhido no intervalo de [1, 20]. Também foi efectuada um estudo usando esta topologia com ligações simétricas de custo 1, para garantir que na presença de topologias simétricas o DTMP não piorava, de forma significativa, os resultados obtidos pelo PIM-SM.

Para cada experiência foram executadas 100 simulações independentes para cada protocolo e os resultados obtidos representam a média dessas 100 simulações.

5.5.1.2 Topologia de rede com 100 nós

O segundo cenário de simulação usado é exactamente igual ao segundo cenário de simulação usado para testar a implementação do PIM-SM.

Também neste caso foi usado o GT-ITM[66][2], um gerador de topologias de rede desenvolvido no Georgia Institute of Technology, para gerar uma topologia de 100 nós. Para criar as diferentes ligações entre os nós da topologia foi usado o método aleatório puro com uma probabilidade de 0.33.

O RP é sempre escolhido aleatoriamente entre todos os nós da topologia e os custos das ligações são também escolhidos aleatoriamente dentro do intervalo [1, 10]. Existe um potencial candidato a novo membro em cada um dos 100 nós e dez fontes fixas. Dez dos candidatos a novos membros, escolhidos aleatoriamente juntam-se ao grupo um de cada vez (em intervalos de tempo distintos) via árvore partilhada e ao fim de algum tempo os 10 membros do grupo já efectivos, juntam-se a cada uma das fontes.

Foram executadas 100 simulações independentes para cada protocolo e os resultados aqui apresentados representam a média dessas 100 simulações.

[2]Georgia Tech - Internetworking Topology Models

5.5.2 Análise de Resultados

Os resultados obtidos das simulações são apresentados figuras 5.6, 5.7, 5.8, 5.9, 5.10 e 5.11

A figura 5.6, ilustra o custo das árvores construídas pelos dois protocolos (PIM-SM e DTMP) quando é utilizada a primeira topologia (topologia típica de um grande ISP). O custo das árvores reflecte a qualidade das árvores construídas e dessa forma fornece um bom meio de comparação entre os diferentes mecanismos de construção. Como foi referido na secção anterior existem várias formas de medir esse custo. As curvas apresentadas nas figuras 5.6(a), 5.6(c) e 5.6(e) mostram os resultados obtidos para a primeira métrica utilizada (número de réplicas por ligação). As curvas apresentadas nas figuras 5.6(b), 5.6(d) e 5.6(f) mostram os resultados obtidos quando é calculada a segunda métrica: número de ligações na topologia que fazem parte das árvores.

Estes resultados demonstram que o DTMP constrói árvores de menor custo do que as construídas pelo PIM-SM, sem aumentar o seu tamanho de forma significativa. Estes resultados são visíveis em qualquer das topologia assimétricas mas como é óbvio tornam-se mais evidentes à medida que as assimetrias são mais significativas. O máximo do ganho do DTMP em relação ao PIM-SM nestas experiências foi de 7%, quando os custos das ligações variam no intervalo [1,5], 12% quando os custos das ligações variam no intervalo [1,10] e 18% quando os custos das ligações variam no intervalo [1,20].

A figura 5.7 mostra a média do custo das árvores de distribuição de tráfego *multicast* construídas pelos dois protocolos (PIM-SM e DTMP) quando é usada a mesma topologia de rede (topologia de 18 nós típica de um ISP) com custos de 1 em todas as ligações.

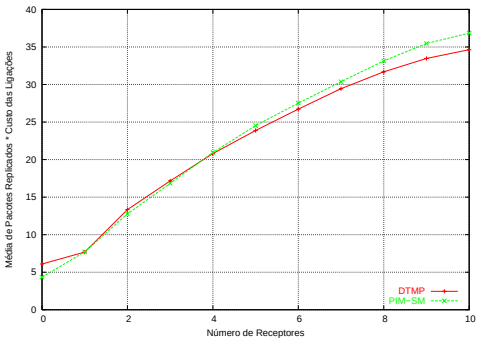
Estes resultados mostram que o DTMP, em topologias simétricas, piora os resultados obtidos pelo protocolo PIM-SM. Neste caso perdeu-se no máximo 12% pela utilização do DTMP em relação ao PIM-SM.

A figura 5.8 mostra a média do custo das árvores de distribuição de tráfego *multicast* construídas pelos dois protocolos (PIM-SM e DTMP) quando é usado o segundo cenário de simulação (topologia de 100 nós gerada aleatoriamente).

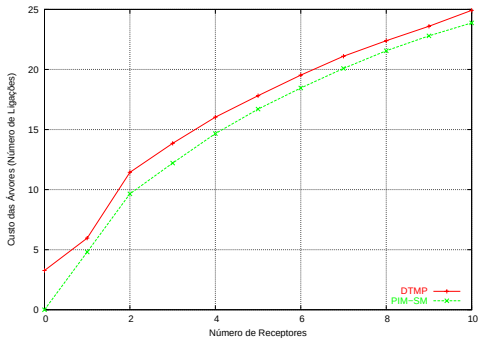
Com este segundo cenário a vantagem do DTMP sobre o protocolo PIM-SM ainda se torna mais evidente. Ou seja, o custo das árvores é significativamente menor, enquanto que o seu tamanho é pouco maior. Este facto indica-nos que o DTMP é capaz de construir melhores árvores do que o PIM-SM sem aumentar de forma significativa o seu tamanho independentemente da topologia e do número de nós presentes. Para este segundo cenário o máximo do ganho do DTMP sobre o PIM-SM foi de 21%.

Na tentativa de descobrir o que distinguia as árvores construídas pelo DTMP das árvores construídas pelo PIM-SM analisaram-se os diferentes caminhos (fim a fim) que separavam as raízes das diferentes árvores das respectivas folhas (membros do grupo), quer em termos de tamanho, quer em termos de custo. A figura 5.9 mostra estas duas métricas obtidas quando se utiliza a topologia de 18 nós típica de um ISP.

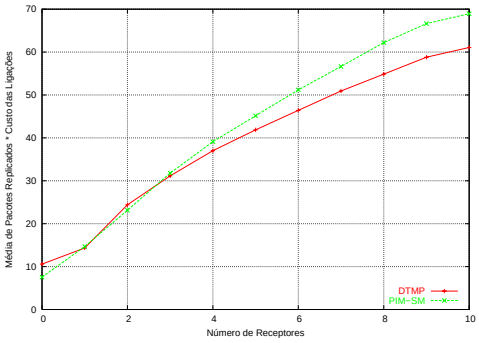
Em relação aos caminhos fim a fim nota-se a mesma tendência, ou seja, apesar dos



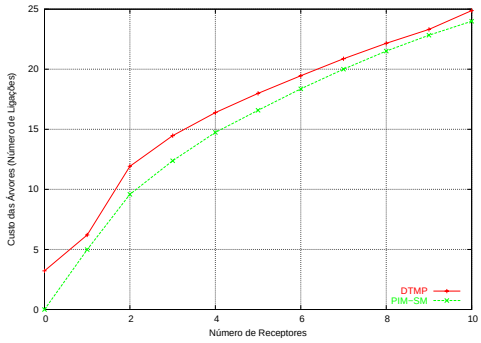
(a) Custo das árvores - Custo das Ligações a variar em [1, 5]



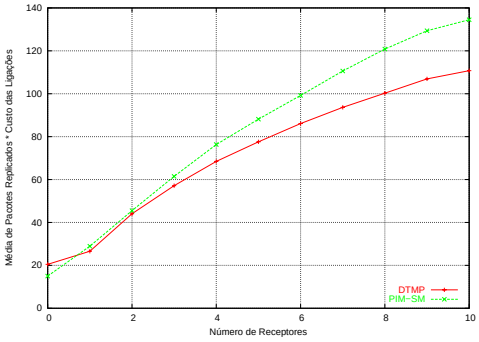
(b) Número de ligações - Custo das Ligações a variar em [1, 5]



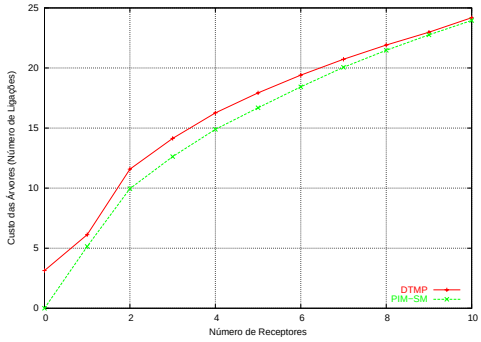
(c) Custo das árvores - Custo das Ligações a variar em [1,10]



(d) Número de ligações - Custo das Ligações a variar em [1,10]

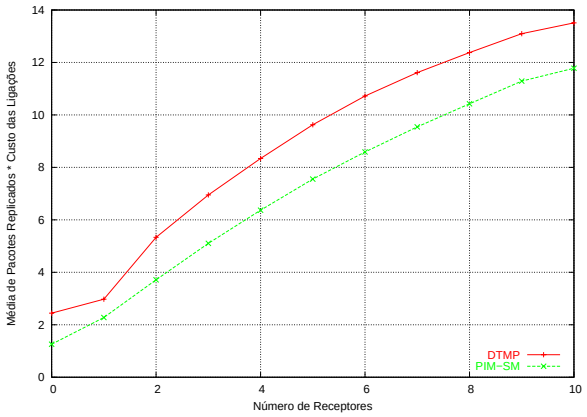


(e) Custo das árvores - Custo das Ligações a variar em [1, 20]

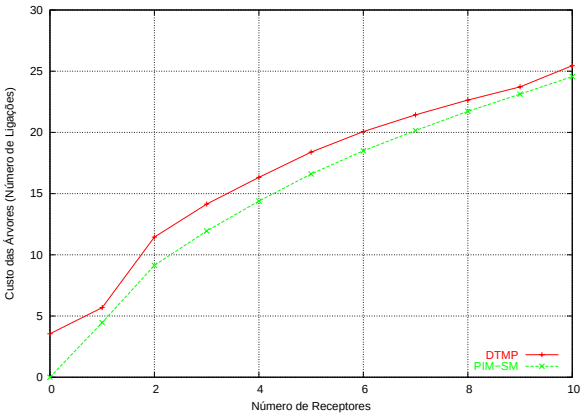


(f) Número de ligações - Custo das Ligações a variar em [1, 20]

Figura 5.6: Custo e número médio das ligações obtidos nas simulações efectuadas com topologia de rede de 18 nós típica de um ISP

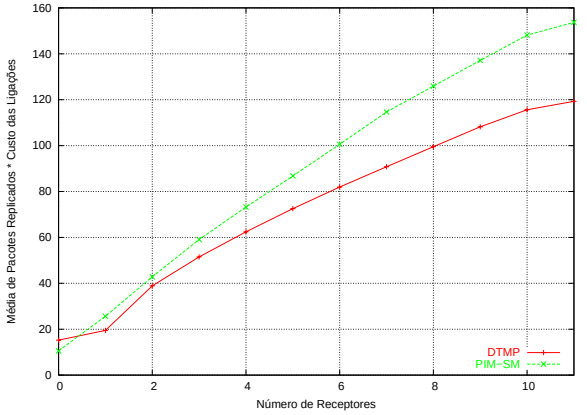


(a) Custo das árvores

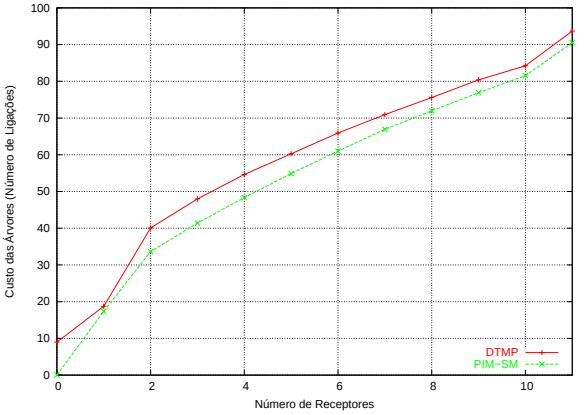


(b) Custo das árvores

Figura 5.7: Custos e número médio das ligações quando é usada uma topologia simétrica

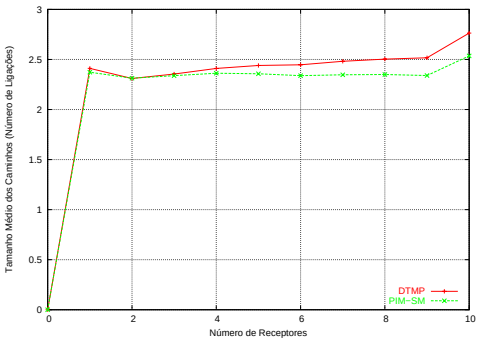


(a) Custo das árvores

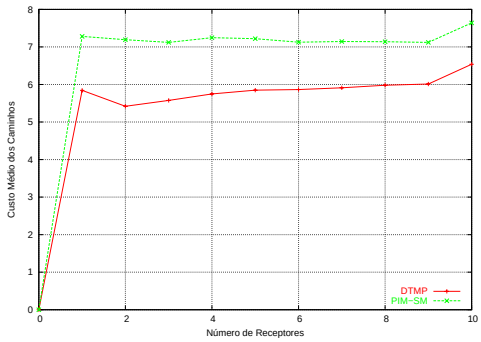


(b) Custo das árvores

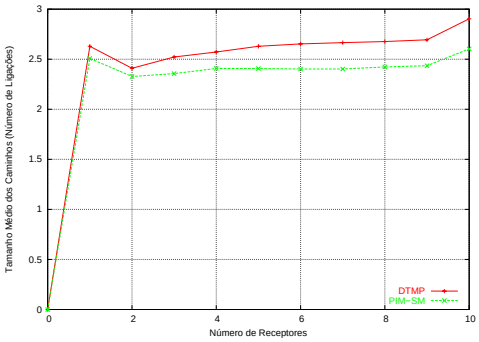
Figura 5.8: Custos e número médio das ligações quando é usada a topologia de rede de 100 nós gerada aleatoriamente



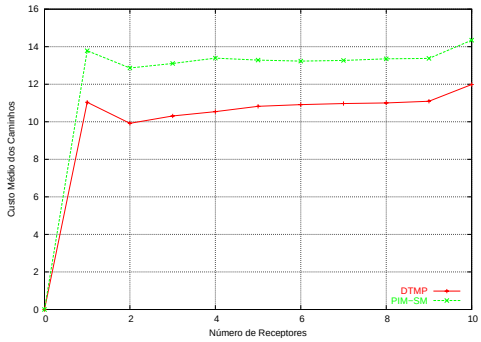
(a) Tamanho dos caminhos - Custo das Ligações a variar em [1, 5]



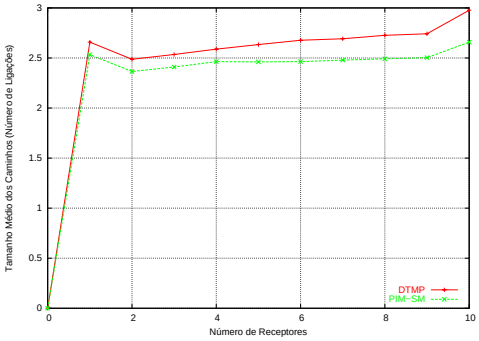
(b) Custo dos caminhos - Custo das Ligações a variar em [1, 5]



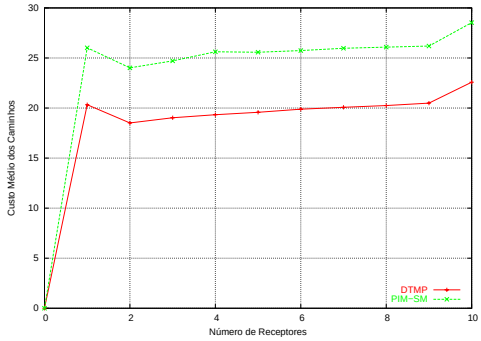
(c) Tamanho dos caminhos - Custo das Ligações a variar em [1,10]



(d) Custo dos caminhos - Custo das Ligações a variar em [1,10]



(e) Tamanho dos caminhos - Custo das Ligações a variar em [1, 20]

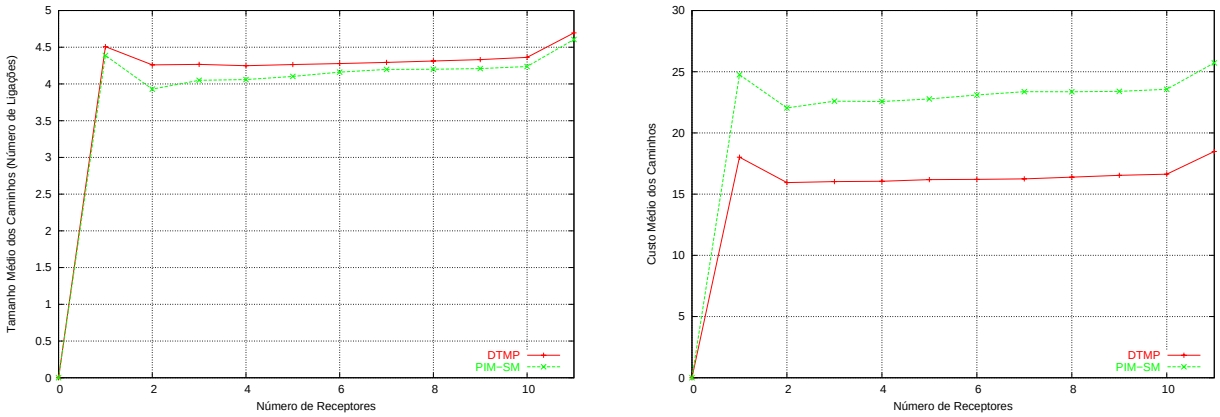


(f) Custo das caminhos - Custo das Ligações a variar em [1, 20]

Figura 5.9: Tamanho e custo dos caminhos fim a fim quando é usada a topologia de rede de 18 nós típica de um ISP

custos dos caminhos serem significativamente menores quando se usa o DTMP, o respectivo tamanho é maior. Isto leva-nos a concluir que as árvores construídas pelo DTMP são constituídas por caminhos mais "longos" do que as construídas pelo PIM-SM.

A figura 5.10 representa as mesmas métricas agora quando é utilizada a topologia de rede de 100 nós gerada aleatoriamente. Como se pode observar os resultados são idênticos, isto é, coerentes independentemente da topologia.



(a) Tamanho dos caminhos

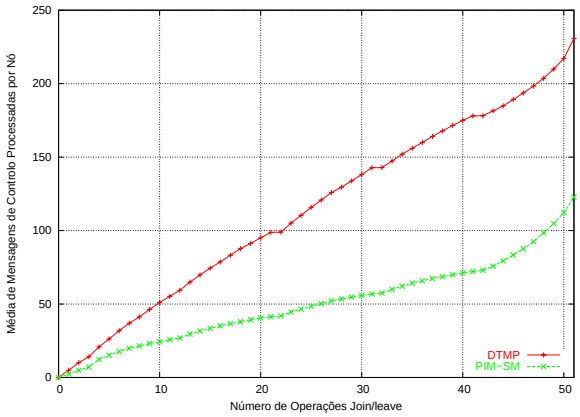
(b) Custo dos caminhos

Figura 5.10: Tamanho e custo dos caminhos fim a fim quando é usada a topologia de rede de 100 nós gerada aleatoriamente

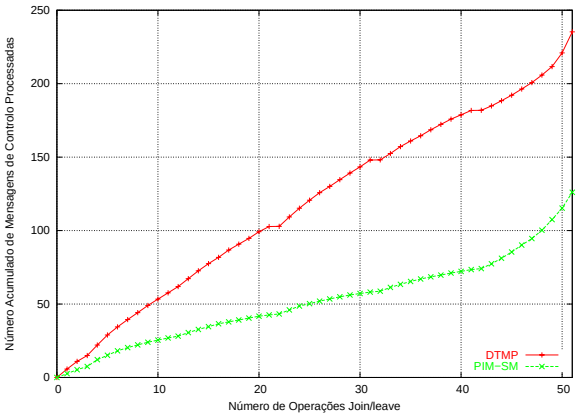
Finalmente, a figura 5.11 reflecte a sobrecarga, medida em termos de mensagens de controle introduzidas, que as operações de *join* e *prune* representam quando se utiliza o DTMP e o PIM-SM. Como já era esperado, de acordo com esta métrica o DTMP tem aparentemente um pior desempenho uma vez que necessita de mais mensagens de controle para construir as árvores. Isto deve-se apenas às operações de junção ao grupo. De acordo com o DTMP, além de serem introduzidas as mensagens de *ack*, que no protocolo PIM-SM não são utilizadas, as mensagens de *join-request* são sempre re-enviadas até atingirem a raíz da árvore, enquanto que no PIM-SM só necessitam de atingir um nó na árvore, neste caso o mais próximo do candidato a novo membro.

5.6 Conclusões

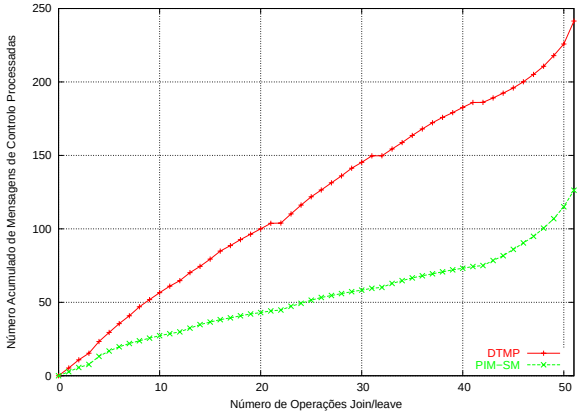
Neste capítulo é apresentada uma descrição do protocolo DTMP - um protocolo que implementa a construção de árvores de distribuição de tráfego *multicast* directas. O protocolo é de alguma forma inspirado no protocolo PIM-SM que é um protocolo de encaminhamento *multicast* amplamente usado na Internet. O PIM-SM, como a maioria dos protocolos de encaminhamento *multicast* constrói árvores invertidas, ou seja no sentido inverso ao



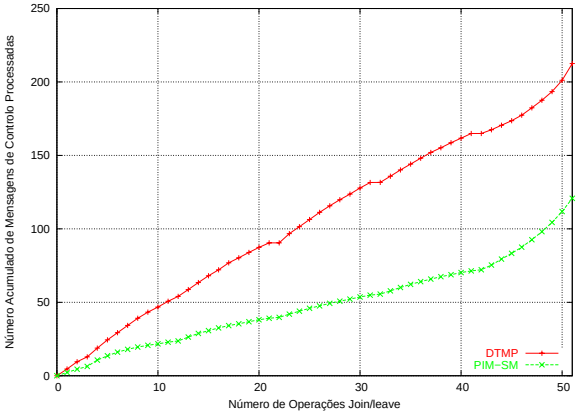
(a) Mensagens de Controle - Custo das Ligações a variar em [1,5]



(b) Mensagens de Controle - Custo das Ligações a variar em [1,10]



(c) Mensagens de Controle - Custo das Ligações a variar em [1,20]



(d) Mensagens de Controle - Topologia Simétrica

Figura 5.11: Número médio de mensagens de controle processadas por cada nó quando é usada a topologia de rede de 18 nós típica de um ISP

sentido usado pelo tráfego. Este facto pode levar a que se construam "más" árvores de distribuição na presença de redes assimétricas, da mesma forma que se criam problemas na adaptação destes protocolos ao encaminhamento com QoS, uma vez que habitualmente as ligações têm diferentes características num e noutro sentido.

O DTMP permite o estabelecimento tanto de árvores partilhadas como de árvores centradas nas fontes. Os candidatos a novos membros começam por se juntar à árvore partilhada cujo centro é um nó pré-definido da topologia, também designado por *Rendez Vous Point* à semelhança do PIM-SM. Depois de terem recebido um determinado número de pacotes de dados provenientes de uma fonte, um receptor pode requerer a junção a uma árvore centrada na fonte. O protocolo DTMP suporta um processo simples de construção de árvores centradas nas fontes, incluindo os mecanismos necessários para "podar" ramos (então desnecessários) da árvore partilhada.

O protocolo proposto foi implementado e avaliado usando o Network Simulator. Os resultados da simulação demonstraram que em presença de assimetrias o DTMP é uma abordagem prometedora permitindo a construção de árvores directas que têm em conta as restrições assimétricas impostas ao encaminhamento. O DTMP foi testado com diferentes protocolos de encaminhamento *unicast*, tanto de vector de distância como de estado de ligação, tendo em todos os casos revelado o mesmo tipo de resultados coerentes,

Convém realçar o facto do DTMP ser independente do protocolo de *unicast* subjacente, à semelhança do que acontece com o protocolo PIM-SM. Este facto facilita a integração do DTMP com a base instalada, tornando-se fácil usá-lo como uma extensão do PIM-SM no sentido de suportar árvores directas. À semelhança do PIM-SM, também o DTMP não tem problemas de ciclos desde que o protocolo de encaminhamento *unicast* garanta esta característica. Adicionalmente, se o protocolo de encaminhamento *unicast* subjacente for capaz de fornecer informação de encaminhamento dependente de QoS, o DTMP poderá construir diferentes árvores de distribuição *multicast* sensíveis às necessidades de Qualidade de Serviço do tráfego que as atravessa.

Capítulo 6

Encaminhamento Unicast por Classes de Serviço

Este capítulo descreve o protocolo CoSLSP (Class-of-Service Link State Protocol), um novo protocolo de encaminhamento *unicast* por classes de serviço. O CoSLSP é um protocolo de estado de ligação, inspirado no protocolo OSPF, que procura calcular rotas distintas para os mesmos destinos, mediante os requisitos de Qualidade de Serviço das diferentes classes de serviço presentes num determinado domínio. Apesar de não fazer parte do principal objectivo deste trabalho, que está mais focado no encaminhamento *multicast*, não foi possível deixar o encaminhamento *unicast* de lado. Isto porque, os protocolos de encaminhamento *multicast* constroem as árvores de distribuição de tráfego com base nos melhores caminhos descobertos pelos protocolos de encaminhamento *unicast*. Para ser possível obter diferentes árvores de distribuição de tráfego *multicast*, dependendo dos requisitos das diferentes classes de serviço, tornou-se assim necessário conceber e implementar uma estratégia de encaminhamento *unicast* por classes de serviço. Essa estratégia foi instanciada neste novo protocolo, o CoSLSP, protocolo que constitui o assunto principal deste capítulo.

6.1 Motivação

O encaminhamento de tráfego na Internet tem seguido até ao momento, o modelo de serviço dito do "melhor esforço". Os protocolos de encaminhamento preocupam-se fundamentalmente com a conectividade e os caminhos são calculados tendo por base apenas o endereço destino. Tipicamente, é considerada uma única métrica estática que se designa por custo e está associada a cada uma das ligações que constitui um caminho. O custo associado a um caminho corresponde, tipicamente, ao somatório dos custos de todas as ligações que o constituem. Em algoritmos de vector de distâncias, se todas as ligações

tiverem o custo unitário, esta métrica é designada por "número de saltos"[1] e corresponde precisamente ao número de ligações que constitui o caminho. Neste caso diz-se que o melhor caminho corresponde ao caminho mais curto, ou seja ao caminho com menor número de "saltos". Os protocolos de encaminhamento procuram então, construir as tabelas de encaminhamento tendo como objectivo minimizar o custo de cada um dos caminhos.

Este modelo é demasiado simplista quando se pretende dotar a rede da capacidade de fornecer Qualidade de Serviço (QoS). Nesse caso pode não interessar enviar todo o tráfego destinado a determinado nó através do mesmo caminho pois apesar de eventualmente ser o mais curto pode não satisfazer os requisitos de QoS e existirem caminhos alternativos que por sua vez os satisfazem. Para ser possível satisfazer um conjunto de requisitos de Qualidade de Serviço os protocolos de encaminhamento têm que obedecer a um modelo mais complexo, pelo menos se se pretender implementar a Qualidade de Serviço ao nível do encaminhamento. Nesse caso uma rede tem que ser caracterizada por um conjunto diversificado de métricas, por exemplo a largura de banda disponível em cada ligação, o atraso fim-a-fim ou a percentagem de perdas verificada em cada interface. O problema fundamental do encaminhamento com QoS é assim encontrar um caminho capaz de satisfazer múltiplas restrições em termos destas ou de outras métricas de Qualidade de Serviço.

O problema do encaminhamento com QoS é muito difícil de resolver por diferentes motivos. Em primeiro lugar, o facto de estarmos perante múltiplas restrições transforma o problema do encaminhamento num problema NP-completo, se pelo menos duas delas forem independentes[71]. Em segundo lugar, o encaminhamento com QoS baseia-se em informação de estado relativa à disponibilidade de recursos nos nós da rede e nas suas ligações. O facto de estarem constantemente a aparecer novos fluxos e a terminar outros provoca grandes flutuações na disponibilidade dos recursos. Manter esse tipo de informação de estado actualizada e coerente em todos os encaminhadores é um processo dispendioso e, em termos de recursos, nem sempre possível.

Existem duas abordagens diferentes para implementar Qualidade de Serviço ao nível do encaminhamento: encaminhamento por fluxo a pedido e encaminhamento por classes de serviço. O encaminhamento por fluxo a pedido é uma estratégia que se adequa melhor ao modelo de Serviços Integrados[2]. Como foi referido no capítulo 3, o modelo IntServ tem como objectivo fornecer garantias de Qualidade de Serviço a cada fluxo individual que atravessa a rede, através da alocação de recursos. Este modelo apresenta como principal vantagem a capacidade de fornecer garantias de serviço através da reserva de recursos, mas por outro lado também possui grandes limitações, sendo a maior o facto de não ser escalável. Cada um dos encaminhadores necessita de manter informação de estado por cada fluxo activo o que traz problemas de escalabilidade em ambientes operacionais. Além disso, é necessária capacidade de processamento adicional em cada um dos encaminhadores no momento do estabelecimento de novas conexões. Esta sobrecarga pode ser significativa, além de que o tempo de estabelecimento de uma nova conexão pode ultrapassar o o tempo de transmissão de todos os pacotes pertencentes ao fluxo.

[1] *number of hops* ou *hop-count* segundo a terminologia Inglesa

Com o objectivo de ultrapassar grande parte das limitações do modelo IntServ foi proposto um novo modelo, designado por Modelo de Serviços Diferenciados, ou de forma abreviada modelo DiffServ[3]. À semelhança do modelo IntServ, o modelo DiffServ foi descrito com maior detalhe no capítulo 3. O principal objectivo do modelo DiffServ é disponibilizar os benefícios de suportar um número limitado de diferentes níveis de Qualidade de Serviço, evitando as limitações do modelo IntServ. Para isso o tráfego é agrupado em classes de serviço, cada uma das quais com diferentes requisitos de QoS. As garantias de QoS são dadas a cada um dos agregados de tráfego, e não aos fluxos individuais. Em oposição ao modelo IntServ não é mantida qualquer informação de estado por fluxo individual, sendo eliminado o tempo de estabelecimento de uma nova conexão.

A maioria das implementações do modelo DiffServ utiliza técnicas internas a cada um dos encaminhadores para obter diferentes níveis de diferenciação do tráfego. Tipicamente, antes de entrarem num domínio Diffserv os pacotes são marcados numa das classes de serviço disponíveis dependendo dos seus requisitos de QoS. Os encaminhadores de um domínio DiffServ, mediante esta marcação dão a cada um dos pacotes um tratamento diferenciado. São utilizadas normalmente diferentes filas de espera para diferentes classes de serviço. Depois é possível configurar de forma independente cada uma destas filas, e mesmo utilizar algoritmos de escalonamento que atribuem diferentes prioridades às filas de espera de forma a obter um tratamento diferenciado para cada uma delas. Um determinado pacote ao ser colocado na fila de espera de uma determinada classe de serviço irá obter o tipo de tratamento que está atribuído a essa classe-de serviço. No entanto este tratamento diferenciado é sentido apenas no interior de cada um dos encaminhadores, não havendo ao nível do encaminhamento qualquer interferência. A utilização do encaminhamento no sentido de diferenciar o tráfego ou pelo menos potenciar essa diferenciação é uma técnica pouco explorada...

No entanto, por não estar prevista a manutenção de informação de estado por fluxo, o cálculo de caminhos por fluxo, a pedido, não é uma estratégia de encaminhamento adequada em redes DiffServ. Em vez disso faz sentido pensar numa estratégia de cálculo de rotas por classes de serviço.

Neste capítulo é apresentado um novo protocolo de encaminhamento *unicast*, o CoSLSP (Class-of-Service Link State Protocol), que implementa o cálculo dos caminhos por classes de serviço. Além do modelo conceptual subjacente e da sua implementação, são também mostrados os resultados obtidos através da sua simulação.

6.2 Trabalho Relacionado

Têm sido propostos diferentes protocolos de encaminhamento com QoS nos últimos anos. Basicamente, esses protocolos combinam as diferentes métricas de QoS usando diferentes estratégias com o objectivo comum de resolver o problema do encaminhamento com QoS. Como já foi referido na secção anterior este problema é NP-Completo quando se procura combinar pelo menos duas métricas de QoS independentes.

Em [45], descrito com maior detalhe no capítulo 3, é proposto um algoritmo de encaminhamento com QoS que pretende contribuir com uma solução para o problema de encontrar um caminho sujeito a restrições em termos do atraso fim a fim e da largura de banda disponível. O problema é resolvido em duas fases, primeiro começa-se por eliminar todos as ligações que não satisfaçam os requisitos da largura de banda e depois é calculado o caminho mais curto em termos de atraso fim a fim, usando para isso o algoritmo de Dijkstra. Se esse caminho obedecer à restrição imposta no atraso fim a fim, então encontrou-se um caminho exequível. Para a concretização deste algoritmo todos os nós da topologia necessitam de possuir o conhecimento completo da topologia e do estado de todas as suas ligações.

Em [72] são propostas diferentes extensões ao protocolo OSPF com o objectivo de implementar encaminhamento com Qualidade de Serviço com recurso a um protocolo de Reserva de Recursos. O protocolo que daí resulta é o QOSPF. O algoritmo de encaminhamento subjacente é muito semelhante ao proposto em [45], descrito em cima.

Em [51] é proposta uma estratégia de encaminhamento distribuído para estabelecer caminhos sujeitos a restrições no atraso fim-a-fim. Esta estratégia não requer a manutenção de informação de estado nos nós da topologia. Segundo esta proposta o nó fonte envia mensagens de controle por todos os caminhos possíveis. Estas mensagens vão acumulando o valor da métrica à medida que a mensagem vai sendo re-enviada nó a nó. Os nós intermédios ao receberem uma mensagem deste tipo verificam o valor acumulado da métrica e re-enviam a mensagem para todos os seus vizinhos se esse valor satisfizer as restrições especificadas. Se uma mensagem chegar ao destino significa que foi encontrado um caminho exequível que nesse caso é estabelecido. A grande vantagem desta estratégia é precisamente não requerer a manutenção de informação de estado global em cada um dos nós da topologia. Mas, por outro lado introduz uma grande sobrecarga na comunicação, uma vez que as mensagens de controle são espalhadas pela rede sempre que se inicia um novo fluxo.

Todas as estratégias até aqui referidas são baseadas no pressuposto de que os caminhos com Qualidade de Serviço têm que ser descobertos por cada pedido, ou seja sempre que se pretende iniciar um novo fluxo é efectuado um pedido em que são explicitados os seus requisitos de QoS, e só depois disso é que o processo de encaminhamento procura descobrir e estabelecer uma rota que os satisfaça. Assim sendo, neste contexto o objectivo do encaminhamento com QoS é satisfazer, individualmente, os requisitos de pedidos efectuados por cada fonte de tráfego antes de começar a emitir. Estas estratégias implicam normalmente um alto custo no processo de encaminhamento devido não só à sobrecarga das actualizações na informação de estado, mas também devido ao processo de sinalização necessário para estabelecer individualmente caminhos por cada pedido.

Outra alternativa é fazer um cálculo prévio de diferentes caminhos, mediante os requisitos de QoS de diferentes classes de serviço, instalando depois os vários caminhos descobertos na tabela de encaminhamento. Neste caso os requisitos de QoS de múltiplos fluxos individuais são mapeados numa única classe. O tamanho das tabelas de encaminhamento pode ser maior do que o das tabelas de encaminhamento usadas no encaminhamento

que se baseia apenas no endereço destino, mas muito menor do que as que têm uma entrada por cada fluxo activo. Além disso desaparece a necessidade de procurar um caminho exequível sempre que se inicia um novo fluxo.

Existem ainda poucas propostas para encaminhamento por classes de serviço. Em [73] é apresentada uma estratégia de encaminhamento por classes de serviço. As diferentes classes têm os seus requisitos de Qualidade de Serviço expressos em termos da sua sensibilidade a perdas e a atrasos. Periodicamente todos os encaminhadores difundem uma métrica pela rede que basicamente reflecte o impacto que os atrasos e as perdas estão a ter em cada uma das classes, nas diferentes interfaces do encaminhador. Com base nesta métrica são calculadas as rotas que conseguirão satisfazer os requisitos de cada uma das diferentes classes.

Em [74] é apresentado um método designado por PERD (Per-class Routing on Per-class Dissemination), que extende o QOSPF[72] segundo o principio do encaminhamento por classes de serviço. Os recursos de rede disponíveis para cada classe de serviço são mantidos numa Base de Dados extendida do Estado das Ligações. Quando ocorre uma mudança nos recursos disponíveis para uma classe essa mudança é difundida para toda a rede numa espécie de LSA. As rotas são calculadas para cada uma das classes de serviço tendo em conta os seus requisitos e a disponibilidade de recursos para cada classe.

6.3 Descrição da estratégia concebida para implementar o encaminhamento por classes de serviço

A ideia básica subjacente desta proposta é disponibilizar uma rota por classe de serviço disponível, capaz de satisfazer os seus requisitos de QoS. Esta proposta distingue-se das outras apresentadas na secção 6.2 deste capítulo pelo facto de, além do objectivo de estabelecer as rotas que melhor se adequam aos diferentes requisitos da classes de serviço, ter também como meta conseguir optimizar a utilização dos recursos de rede. Como já foi referido esse é o objectivo do encaminhamento tradicional, que procura encontrar o caminho de menor custo ou o caminho mais curto...

Assim sendo, em vez de se limitar a procurar diferentes caminhos que satisfaçam os diferentes requisitos de QoS das diferentes classes de serviço, o protocolo proposto procura encontrar o caminho de menor custo que consegue satisfazer determinados requisitos de QoS.

O objectivo principal não é mais uma proposta sobre como combinar diferentes métricas de QoS. Em vez disso, um novo algoritmo baseado no algoritmo de Dijkstra é apresentado, capaz de descobrir (caso exista) o caminho mais curto capaz de satisfazer os requisitos de QoS de uma determinada classe.

6.3.1 Algoritmo para o cálculo de Rotas

O algoritmo de Dijkstra é um algoritmo simples que calcula de uma vez só os caminhos mais curtos para todos os destinos possíveis.

Já foi descrito de forma mais ou menos exaustiva no capítulo 2. No entanto, e porque o algoritmo para cálculo de rotas usado pelo CoSLSP é uma variante do algoritmo de Dijkstra, optou-se por voltar nesta secção a referir, embora de forma mais abreviada, o funcionamento deste algoritmo, bem como retomar o exemplo já apresentado no capítulo 2. Na figura 6.1 é apresentada uma topologia muito simples, e na tabela 6.1 as várias iterações do algoritmo Dijkstra para calcular as rotas para todos os destinos a partir do nó 1.

O algoritmo Dijkstra calcula de forma incremental uma árvore de caminhos mais curtos, desde um determinado nó para todos os destinos. O processo de cálculo começa por acrescentar o próprio nó à árvore dos caminhos mais curtos, e todos os seus vizinhos a uma lista, designada por lista de candidatos. Cada candidato tem associado um custo e o próximo salto. Neste caso o custo corresponde precisamente ao custo da ligação que separa o nó do respectivo vizinho, e o próximo salto é a identificação do próprio vizinho. Em cada iteração é escolhido da lista de candidatos o candidato com menor custo. Esse candidato é acrescentado à árvore de caminhos mais curtos. Todos os vizinhos do candidato escolhido são analisados. Podem acontecer três situações distintas:

- O vizinho já consta da árvore de caminhos mais curtos. Nesse caso não se faz mais nada, passando-se a analisar o vizinho seguinte.
- O vizinho não consta da árvore de caminhos mais curtos, nem da lista de candidatos. Nesse caso é inserido na lista de candidatos, adicionando-se ao custo do candidato escolhido o custo da ligação que separa esse candidato do vizinho. O campo com o próximo salto é mantido.
- O vizinho não consta da árvore de caminhos mais curtos, mas consta da lista de candidatos. Nesse caso adiciona-se ao custo do candidato escolhido o custo da ligação que separa esse candidato do vizinho e, só se o resultado dessa soma for inferior ao custo do candidato já existente na lista de candidatos é que o novo vizinho é inserido na lista de candidatos e o outro é apagado. No caso de ser inserido o novo vizinho, o campo próximo salto fica igual ao do candidato escolhido.

6.3.1.1 Determinação de rotas pelo CoSLSP

O algoritmo para cálculo das rotas usado pelo CoSLSP é basicamente um algoritmo de Dijkstra modificado. A ideia subjacente consiste em calcular não umas mas múltiplas rotas, uma por classe, dependendo dos respectivos requisitos de Qualidade de Serviço. Em particular, neste trabalho optou-se por usar como métrica de QoS a percentagem de perdas

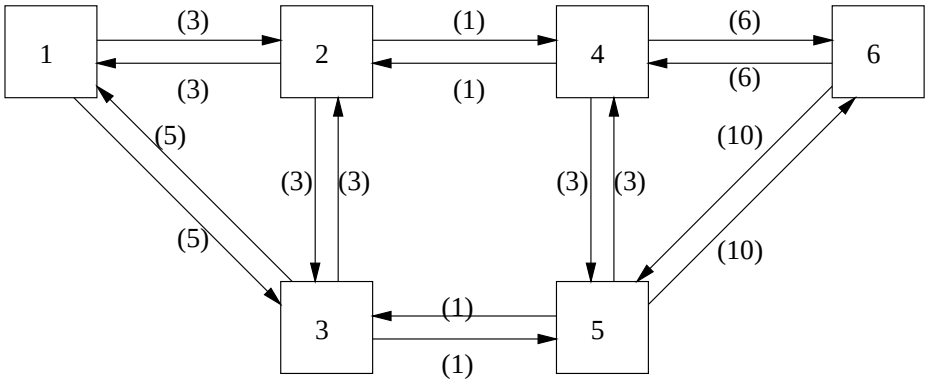


Figura 6.1: Pequena topologia usada como exemplo

Iteração	Candidato escolhido	Rota inserida na RT	Lista de Candidatos
1		1(0,1)	2(3, 2) 3(5,3)
2	2(3, 2)	2(3, 2)	4(4, 2) 3 (5, 3)
3	4(4, 2)	4(4, 2)	3(5, 3) 5(7, 2) 6(10, 2)
4	3(5, 3)	3(5, 3)	5(6, 3) 6(10, 2)
5	5(6, 2)	5(6, 2)	6(10, 2))
6	6(10, 2)	6(10, 2)	

Tabela 6.1: Aplicação do algoritmo de Dijkstra

ocorrida em cada ligação. Os custos das diferentes ligações também são considerados. Assim, por cada ligação (neste caso unidireccional) é possível obter a seguinte informação:

- Custo da ligação;
- Número de Classes, que corresponde ao número de filas físicas;
- Por cada classe, a taxa de perdas que se verifica em cada momento;

Cada nó da topologia recolhe periodicamente esta informação acerca das suas ligações, e é esta informação que é agrupada para formar os LSAs, que depois são divulgados por toda a rede.

As estruturas de dados usadas pouco diferem das apresentada para a explicação do algoritmo Dijkstra. A lista de candidatos além de conter a identificação do nó destino, o custo e o próximo salto, contém também uma lista dos nós visitados e a taxa de perdas acumulada. A tabela de encaminhamento contém além do nó destino, do custo e do próximo salto, a taxa de perdas e a classe a que corresponde o caminho.

A regra que compõe a taxa de perdas acumulada é dada pela seguinte fórmula: $Tp_{(c)} = 1 - ((1 - Tp_{(i,j)}) * (1 - Tp_{(j,k)}) * ... * (1 - Tp_{(j,k)}))$, sendo c um determinado caminho, $Tp_{(c)}$ a taxa de perdas verificada no caminho c, e $Tp_{(i,j)}$ a taxa de perdas verificada na ligação que une os nós i e j.

Para uma determinada classe é necessário saber qual o limite máximo de perdas tolerado. Nesta fase, este foi o único requisito de QoS considerado.

O algoritmo proposto começa por adicionar o próprio nó à tabela de encaminhamento e todos os seus vizinhos à lista de candidatos, exactamente como acontece no algoritmo de Dijkstra. Depois, também à semelhança do que acontece no Dijkstra, em cada iteração é escolhido entre os candidatos aquele que tem menor custo. Ao ser escolhido, o candidato é retirado da lista de candidatos. Para cada candidato escolhido é necessário verificar se este já consta ou não da tabela de encaminhamento. Se não consta e a taxa de perdas não ultrapassa o máximo tolerado, então é acrescentado à tabela de encaminhamento. Em qualquer dos casos é necessário ir buscar à base de dados topológica todos os vizinhos desse candidato que não pertençam à lista de nós visitados, juntamente com os atributos que caracterizam a ligação que separa o candidato do respectivo vizinho. A cada um desses vizinhos corresponderá um novo candidato que será acrescentado à lista de candidatos depois de serem actualizados a taxa de perdas, o custo do caminho, a lista de visitados e o próximo salto. Estes novos atributos são obtidos a partir dos mesmos atributos do candidato que está a ser processado e dos atributos da ligação que separa o candidato do dito vizinho, da seguinte forma:

- O custo do novo caminho é obtido somando ao custo do caminho do candidato o custo da ligação entre esse candidato e o nó vizinho.

- A taxa de perdas do novo caminho é obtida através da fórmula para a taxa de perdas apresentada a cima..
- A nova lista de visitados é igual à do candidato acrescentado-lhe o nó vizinho.
- O próximo salto é simplesmente copiado.

O algoritmo continua até a lista de candidatos estar vazia, ou a tabela de encaminhamento estar completa. No fim é necessário verificar se a tabela de encaminhamento está ou não completa porque pode acontecer que não tenham sido descobertos caminhos para todos os vizinhos que respeitem os requisitos de determinada classe. Nesse caso o caminho escolhido será o mais curto.

Para implementar esta ultima funcionalidade é necessário ir guardando numa tabela de encaminhamento auxiliar os caminhos mais curtos assim que eles forem encontrados. Desta forma, bastará no fim do algoritmo copiar desta tabela os caminhos que não constem da tabela de encaminhamento principal.

6.3.1.2 Descrição matemática do algoritmo para cálculo das rotas usado pelo CoSLSP

Sendo $G = (N, L)$ um grafo que representa um rede com N nós e L ligações. Sejam c_{ij} e u_{ij} dois números reais associados a cada ligação $l(i, j)$ representando respectivamente o custo da ligação e o valor da métrica considerada. Quando a ligação $l(i, j)$ não existe no grafo, então $c_{ij} = \infty$ e $u_{ij} = \infty$ (ou $u_{ij} = 0$, dependendo da métrica considerada). Dado um caminho $c = (i, j, k, \ldots, l, m)$ e uma constante U para o valor máximo aceitável na métrica considerada, o problema é encontrar o caminho mais curto c entre i e m de tal forma que o valor acumulado para a métrica considerada seja inferior a U.

Supondo que o nó 1 é o nó fonte e o nó m o nó destino, o algoritmo seguinte encontra o caminho mais curto entre o nó 1 e o nó m que satisfaz a restrição U. Seja U_i o valor acumulado para a restrição U e C_i o custo total de um caminho entre o nó 1 e o nó i.

Começar
$\forall_{ij} : u_{ij} = \infty \; se \; c_{ij} = \infty$;
$L = \{1\}$;
Encontrar Conjunto $K : C_k = c_{1k} \; \forall k \neq 1$;
Enquanto $K \neq \{\}$ **fazer**;
Encontrar $k \in K : C_k = \min_{i \in K} C_i$;
Se $U_k > U$ **então** $K = K \setminus \{k\}$;
Senão
$L \cup \{k\}$;
Se $m \in L$ *entao foi encontrado o caminho. Fim.*;
Encontrar Conjunto $J : \forall_{j \in J} C_j = C_k + c_{kj}$;

$K = K \cup J$;
Fim EnquantoFazer
Fim

6.3.1.3 CoSLSP: exemplo de aplicação

Para ilustrar melhor este algoritmo optou-se por mostrar um exemplo. A topologia usada é precisamente topologia que se usou para exemplificar o algoritmo de Dijkstra, acrescentaram-se apenas a taxa de perdas nas ligações. Para não se tornar demasiado complicado ir-se-á considerar uma só classe que pretende ter como limite máximo de perdas uma taxa de 0.3.

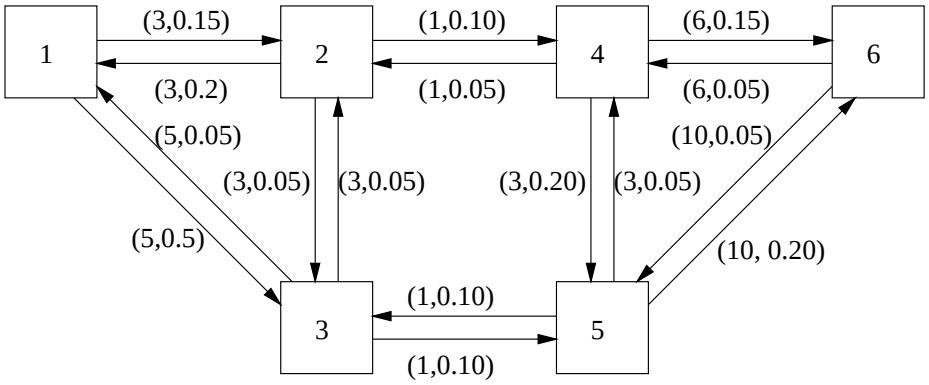


Figura 6.2: Pequena topologia com perdas nas ligações

As várias iterações do algoritmo para o nó 1 estão representadas na tabela 6.2.

Iteração	Candidato escolhido	Rota inserida na RT	Lista de Candidato
1		1(0, 0, 1)	2(3, 0.15, 2, [1,2]) 3 (5, 0.05, 3, [1,3])
2	2(3, 0.15, 2, [1,2])	2(3, 0.15, 2)	3(6, 0.20, 2, [1,2,3]) 4(4, 0.24, 2, [1,2,4]) 3 (5, 0.05, 3, [1,3])
3	4(4, 0.24, 2, [1,2,4])	4(4, 0.24, 2)	5(7, 0.28, 2, [1,2,4,5]) 6(10, 0.35, 2, [1,2,4,6]) 3(5, 0.05, 3, [1,3]) 3(6, 0.20, 2, [1,2,3])

Iteração	Candidato escolhido	Rota inserida na RT	Lista de Candidato
4	3(5, 0.05, 3, [1,3])	3(5, 0.05, 3)	2(8, 0.10, 3, [1,3,2]) 5(6, 0.15, 3, [1,3,5]) 3(6, 0.20, 2, [1,2,3]) 5(7, 0.28, 2, [1,2,4,5]) 6(10, 0.35, 2, [1,2,4,6])
5	3(6, 0.20, 2, [1,2,3])	Já existe na RT	5(7, 0.28, 2, [1,2,3,5]) 2(8, 0.10, 3, [1,3,2]) 5(6, 0.15, 3, [1,3,5]) 5(7, 0.28, 2, [1,2,4,5]) 6(10, 0.35, 2, [1,2,4,6])
6	5(6, 0.15, 3, [1,3,5])	5(6, 0.15, 3)	4(9, , 3,[1,3,5,4]) 6(16, 0,19, 3,[1,3,5,6]) 5(7, 0.28, 2, [1,2,3,5]) 5(7, 0.28, 2, [1,2,4,5]) 2(8, 0.10, 3, [1,3,2]) 6(10, 0.35, 2, [1,2,4,6])
7	5(7, 0.28, 2, [1,2,3,5])	Já existe na RT	4(10, , 2, [1,2,3,5,4]) 6(17, 0.31, 2, [1,2,3,5,6]) 5(7, 0.28, 2, [1,2,4,5]) 2(8, 0.10, 3, [1,3,2]) 4(9, , 3, [1,3,5,4]) 6(10, 0.35, 2, [1,2,4,6]) 6(16, 0,19, 3, [1,3,5,6])
8	5(7, 0.28, 2, [1,2,4,5])	Já existe na RT	3(8, , 2, [1,2,4,5,3]) 6(17, 0.31, 2, [1,2,4,5,6]) 2(8, 0.10, 3, [1,3,2]) 4(9, , 3, [1,3,5,4]) 4(10, , 2, [1,2,3,5,4]) 6(10, 0.35, 2, [1,2,4,6]) 6(16, 0,19, 3, [1,3,5,6]) 6(17, 0,31, 2, [1,2,3,5,6])
9	3(8, , 2, [1,2,4,5,3]	Já existe na RT	2(8, 0.10, 3, [1,3,2]) 4(9, , 3, [1,3,5,4]) 4(10, , 2, [1,2,3,5,4]) 6(10, 0.35, 2, [1,2,4,6]) 6(16, 0,19, 3, [1,3,5,6]) 6(17, 0,31, 2, [1,2,3,5,6]) 6(17, 0,31, 2, [1,2,4,5,6])

Iteração	Candidato escolhido	Rota inserida na RT	Lista de Candidato
10	2(8, 0.10, 3, [1,3,2])	Já existe na RT	4(9, , 3, [1,3,2,4])
			4(9, , 3, [1,3,5,4])
			4(10, , 2, [1,2,3,5,4])
			6(10, 0.35, 2, [1,2,4,6])
			6(16, 0,19, 3, [1,3,5,6])
			6(17, 0,31, 2, [1,2,3,5,6])
			6(17, 0,31, 2, [1,2,4,5,6])
11	4(9, , 3, [1,3,5,4])	Já existe na RT	5(12, , 3, [1,3,2,4,5])
			6(15, , 3, [1,3,2,4,6])
			4(9, , 3,[1,3,5,4])
			4(10, , 2, [1,2,3,5,4])
			6(10, 0.35, 2, [1,2,4,6])
			6(16, 0,19, 3, [1,3,5,6])
			6(17, 0,31, 2, [1,2,3,5,6])
			6(17, 0,31, 2, [1,2,4,5,6])
12	4(9, , 3,[1,3,5,4]	Já existe na RT	2(10, , 3, [1,3,5,4,2]
			6(15, , 3, [1,3,5,4,6]
			4(10, , 2, [1,2,3,5,4])
			6(10, 0.35, 2, [1,2,4,6])
			5(12, , 3, [1,3,2,4,5])
			6(15, , 3, [1,3,2,4,6])
			6(16, 0,19, 3, [1,3,5,6])
			6(17, 0.31, 2, [1,2,3,5,6])
			6(17, 0,31, 2, [1,2,4,5,6])
13	2(10, , 3, [1,3,5,4,2]	Já existe na RT	4(10, , 2, [1,2,3,5,4])
			6(10, 0.35, 2, [1,2,4,6])
			5(12, , 3, [1,3,2,4,5])
			6(15, , 3, [1,3,2,4,6])
			6(15, , 3, [1,3,5,4,6])
			6(16, 0,19, 3, [1,3,5,6])
			6(17, 0,31, 2, [1,2,3,5,6])
			6(17, 0,31, 2, [1,2,4,5,6])
14	4(10, , 2, [1,2,3,5,4])	Já existe na RT	6(16, , 2, [1,2,3,5,4,6)
			6(10, 0.35, 2, [1,2,4,6])
			5(12, , 3, [1,3,2,4,5])
			6(15, , 3, [1,3,2,4,6])
			6(15, , 3, [1,3,5,4,6])
			6(16, 0,19, 3, [1,3,5,6])
			6(17, 0.31, 2, [1,2,3,5,6])
			6(17, 0.31, 2, [1,2,4,5,6])

Iteração	Candidato escolhido	Rota inserida na RT	Lista de Candidato
15	6(10, 0.35, 2, [1,2,4,6]	Não respeita os requisitos	5(20, , 2, [1,2,4,6,5])
			5(12, , 3, [1,3,2,4,5])
			6(15, , 3, [1,3,2,4,6])
			6(15, , 3, [1,3,5,4,6])
			6(16, , 2, [1,2,3,5,4,6)
			6(16, 0,19, 3, [1,3,5,6])
			6(17, 0.31, 2, [1,2,3,5,6])
			6(17, 0.31, 2, [1,2,4,5,6])
16	5(12, , 3, [1,3,2,4,5]	Já existe na RT	6(16, 0.27, 3, [1,3,2,4,5,6])
			6(15, 0.31, 3, [1,3,2,4,6])
			6(15, 0.42, 3, [1,3,5,4,6])
			6(16, 0.39, 2, [1,2,3,5,4,6)
			6(16, 0.19, 3, [1,3,5,6])
			6(17, 0.31, 2, [1,2,3,5,6])
			6(17, 0.31, 2, [1,2,4,5,6])
			5(20, , 2, [1,2,4,6,5])
17	6(15, 0.31, 3, [1,3,2,4,6]	Não respeita os requisitos	5(25, , 3, [1,3,2,4,6,5])
			6(15, 0.42, 3, [1,3,5,4,6])
			6(16, 0.39, 2, [1,2,3,5,4,6)
			6(16, 0.19, 3, [1,3,5,6])
			6(16, 0.27, 3, [1,3,2,4,5,6])
			6(17, 0.31, 2, [1,2,3,5,6])
			6(17, 0.31, 2, [1,2,4,5,6])
			5(20, , 2, [1,2,4,6,5])
18	6(15, 0.42, 3, [1,3,5,4,6])	Não respeita os requisitos	6(16, 0.39, 2, [1,2,3,5,4,6)
			6(16, 0.19, 3, [1,3,5,6])
			6(16, 0.27, 3, [1,3,2,4,5,6])
			6(17, 0.31, 2, [1,2,3,5,6])
			6(17, 0.31, 2, [1,2,4,5,6])
			5(20, , 2, [1,2,4,6,5])
			5(25, , 3, [1,3,2,4,6,5])
19	6(16, 0.39, 2, [1,2,3,5,4,6)	Não respeita os requisitos	6(16, 0.19, 3, [1,3,5,6])
			6(16, 0.27, 3, [1,3,2,4,5,6])
			6(17, 0.31, 2, [1,2,3,5,6])
			6(17, 0.31, 2, [1,2,4,5,6])
			5(20, , 2, [1,2,4,6,5])
			5(25, , 3, [1,3,2,4,6,5])
20	6(16, 0.19, 3, [1,3,5,6])	6(16, 0.19, 3)	RT Completa

Tabela 6.2: Aplicação do algoritmo do CoSLSP

6.3.2 Informação de Estado

A estrutura da tabela de encaminhamento *unicast* tem que ser alterada no sentido de incluir alguns campos adicionais. A chave deixa de ser apenas a identificação do nó destino e passa a ser um par constituído pela identificação do nó destino e pela identificação da classe de serviço.

Além da informação relativa ao custo do caminho, é necessário também incluir o valor acumulado da métrica considerada. Assim, cada entrada na tabela de encaminhamento *unicast* deverá ser constituída pelos seguintes campos de informação:

- *dst* - endereço do nó destino
- *classId* - identificador da Classe de Serviço
- *cost* - custo total do caminho
- *metric* - valor acumulado da métrica no caminho
- *nextHop* - identificador do próximo salto

6.3.3 O Processo de Re-Envio de Pacotes

Ao receber um pacote, o encaminhador extrai o endereço destino, e a classe de serviço em que o pacote foi marcado, construindo a chave de pesquisa na tabela de encaminhamento com base nestes dois valores. O pacote é re-enviado para o nó identificado através do campo *nexthop* da entrada na tabela de encaminhamento que coincide com o par constituído por este par de valores.

6.4 Implementação do CoSLSP

Para implementar o protocolo CoSLSP[9] usou-se a versão 2.18b do Network Simulator(NS)[5]. Como foi referido no capítulo 4 o NS inclui implementações de alguns protocolos de encaminhamento *unicast*. Além disso inclui uma implementação de DiffServ desenvolvida pela Nortel, que também foi modificada de forma a integrar e facilitar o teste da implementação do CoSLSP desenvolvida. Esta secção foi assim dividida em três secções principais: inicialmente será descrita a implementação de DiffServ da Nortel, bem como as alterações efectuadas a esse nível de forma a suportar o protocolo CoSLSP. Depois descreve-se a implementação do protocolo propriamente dito. Por fim, na secção 6.5 apresentam-se os testes realizados e os resultados obtidos.

6.4.1 Implementação do DiffServ no NS

Neste trabalho foi usada a implementação de DiffServ da Nortel incorporada na versão 2.1b8 do NS. Basicamente, a Nortel implementa o esquema proposto pelo *Assured Forwarding Per-Hop-Behavior - AF PHB*[75], ou seja 4 classes, cada uma delas com 3 níveis de precedência.

A classe que é extendida é a classe **Queue** que implementa no NS as filas de espera. Basicamente é criado um novo tipo de fila de espera que deriva da classe **Queue**, à semelhança de outros tipos de filas, nomeadamente, DropTail, CBQ e RED. Esta nova fila é implementada através da classe **dsREDQueue** que é uma subclasses da classe **Queue**, que por sua vez deriva da classe **Conector**.

A classe **dsREDQueue** implementa as seguintes funcionalidades:

- Múltiplas filas físicas (*Queues RED*) numa só ligação;
- Múltiplas filas virtuais por cada fila física com parâmetros distintos;
- Capacidade para determinar em que filas (física e virtual) é que é colocado determinado pacote, dependendo da classe de serviço em que este se encontra "marcado".

A classe **dsREDQueue** está definida nos ficheiros dsred.h e dsred.cc. As múltiplas filas físicas são implementadas com base na classe **redQueue**. A classe **redQueue** está definida nos ficheiros dsredq.h e dsredq.cc.

Por sua vez as classes **edgeQueue** e **coreQueue** derivam da classe **dsREDQueue** e implementam as funcionalidades que deverão estar presentes nos *edge nodes* e nos *core nodes* respectivamente.

6.4.1.1 Alterações efectuadas ao nível das filas

Foi acrescentada uma classe, a classe **ourdsREDQueue** que deriva da classe **dsREDQueue** e da qual as classes **edgeQueue** e **coreQueue** passaram a derivar.

À classe **ourdsREDQueue** foram acrescentados dois novos *arrays* de estruturas. Um deles mantém informação relativa ao número de pacotes que vão chegando a cada uma das filas físicas, enquanto o outro vai contabilizando as perdas que ocorrem nas mesmas filas.

- O *array* **packets** tem os seguintes campos: um campo **updated** que é actualizado sempre que chega um pacote novo à fila, quer seja ou não "descartado", um campo **old** que contém o número de pacotes que chegaram até ao intervalo de tempo imediatamente anterior, e finalmente um campo **diff** onde é contabilizado o número de pacotes que chegaram no intervalo de tempo actual (basicamente é obtido calculando a diferença entre o campo **updated** e o campo **old**);

- O *array* **drops** tem os mesmos campos que o *array* **packets** para contabilizar as perdas que ocorrem em cada umas das filas físicas num intervalo de tempo. Além destes campos contém um campo **rate** que contabiliza a taxa de perdas ocorridas no intervalo de tempo actual (basicamente é igual ao quociente entre o número de perdas ocorridas e o total de pacotes chegados no intervalo de tempo actual).

Além dos métodos *reset* e *enqueue* que tiveram que ser alterados por forma a actualizar estas novas variáveis foram criados os seguintes métodos novos:

- O método **dropsRate_update** é invocado com uma determinada periodicidade e actualiza os vários campos dos *arrays* **packets** e **drops**. Basicamente os campos **diff** são actualizados com a diferença entre o campo **updated** e o campo **old**. O campo **rate** do *array* **drops** é actualizado com o quociente entre o campo **diff** do *array* **drops** e o campo **diff** do *array* **packets**. Se o campo **diff** do *arrays* **packets** for inferior a 10 conclui-se que não chegaram pacotes de forma significativa e o campo **rate** do *array* **drops** é posto a zero, o que significa que não houve perdas. Os campos **old** dos dois *arrays* são actualizados com os valores contidos nos campos **updated** respectivos. Os campos **updated** não são actualizados neste método, mas sim no método *enqueue*. O campo **updated** do *array* **packets** é incrementado sempre que um pacote chega à fila, quer seja ou não "descartado" (*dropped*) e o campo **updated** do *array* **drops** é incrementado sempre que um pacote é "descartado";

- O método **dropsRate_changes** recebe como argumento um *double* que representa a variação máxima permitida. Este método basicamente faz o seguinte: para cada umas das filas físicas vai determinar o número de pacotes chegados e perdas ocorridas num intervalo de tempo (ou seja desde a ultima vez em foi feita uma actualização pelo método **dropsRate_update**). Para isso actualiza os campos **diff** da variável **packets** e da variável **drops**. Se o campo **diff** da variável **packets** variou de forma significativa, ou seja se chegaram mais de 10 pacotes, então é contabilizada uma nova taxa de perdas, ou seja é calculado o quociente entre o campo **diff** da variável **drops** e o campo **diff** da variável **packets**. Se não, ou seja se se concluiu que no instante de tempo actual não chegaram pacotes de forma significativa, a nova taxa de perdas toma o valor de zero. Essa taxa de perdas é comparada com a taxa de perdas contida no campo **rate** da variável **drops**. Se a diferença entre elas ultrapassar a variação máxima (valor introduzido como parâmetro no método) então é incrementado um inteiro **changes** que basicamente contabiliza o número de filas físicas em que houve alterações significativas na taxa de perdas. Esse inteiro **changes** é retornado quando foram determinadas as alterações em todas as filas físicas. Basicamente é retornado zero senão houve alterações significativas em nenhuma das filas ou um número diferente de zero, igual ao número de filas em que houve alterações significativas na taxa de perdas ocorridas. É de salientar que neste método os campos **old** das variáveis **packets** e **drops** não são alterados assim como o campo **rate** do *array* **drops**. Assim sendo, da próxima vez que este método for executado a taxa de perdas não

será medida em relação a este instante de tempo, mas sim em relação à ultima vez que actualização é feita. Isto permite-nos ter tempos diferentes de actualização e detecção de variações significativas. Um intervalo de tempo mais pequeno em que se tenta detectar se houve ou não alterações significativas e um intervalo de tempo maior em que são feitas as actualizações, quer haja ou não variações significativas. No entanto sempre que forem detectadas variações significativas todos os campos deverão ser actualizados;

- O método **getDropsRate** recebe como argumento uma *string* que preenche com o estado de cada umas das filas físicas. O primeiro valor colocado na *string* é o número de filas físicas logo seguido de pares com o número da fila e respectiva taxa de perdas (este último valor é retirado do campo **rate** do *array* **drops**).

6.4.2 A implementação do protocolo LS no NS

O **rtProtoLS** é uma implementação para o NS de um protocolo de estado de ligação que calcula dinamicamente as rotas de forma distribuída. É implementado no NS através da classe **LS**. A classe **Agent/rtProto/LS** é uma subclasse da classe **Agent/rtProto**, e é esta classe que implementa o protocolo LS. Tem uma classe correspondente em C++, a classe **rtProtoLS** que implementa o algoritmo Dijkstra para calcular as melhores rotas. A classe **Agent/rtProto/LS** implementa em OTcl a componente responsável pela troca de mensagens entre os vários agentes e manutenção das bases de dados topológicas em cada um dos nós.

O método **init** da classe **Agent/rtProto/LS**, além de inicializar todas as variáveis de instância dos agentes de encaminhamento invoca o método **send-periodic-update**. Este método é invocado periodicamente e é ele que é o responsável por difundir LSAs com estado das ligações pela rede. Basicamente, o método **send-periodic-update** invoca o método **sendUpdates** na classe rtProtoLS. Esse método invoca o método **sendLinkStates** na variável de instância **routing_**. A variável **routing_** é uma instância da classe **LsRouting**. O método **sendLinkStates** da classe **LsRouting** procura na base de dados topológica o estado das suas próprias ligações e envia-o em forma de LSA para todos os seus vizinhos.

Na implementação do protocolo LS a troca de mensagens entre os vários agentes de encaminhamento é implementada através de um centro de mensagens. O que é transmitido pelos vários agentes de encaminhamento é apenas um identificador da mensagem. As mensagens propriamente ditas são mantidas numa estrutura central (classe **LsMessageCenter**). Assim, quando um nó decide enviar um LSA, limita-se a acrescentar o LSA ao **LsMessageCenter** e enviar um identificador a todos os seus vizinhos, identificador esse que lhes permitirá ir buscar o LSA respectivo ao **LsMessageCenter**. Um nó quando consegue ir buscar o LSA ao **LsMessageCenter** envia um "aknowledge" ao nó que originou o LSA. Esse nó depois de ter recebido "aknowledge" de todos os nós que constituem a topologia apaga a respectiva mensagem do **LsMessageCenter**.

Os métodos principais da classe **LsMessageCenter** são três:

- o método **newMessage**
- o método **retrieveMessage**
- o método **deleteMessage**

Quando um nó recebe uma mensagem de um agente de encaminhamento LS é invocado o método **receiveMessage** da classe **LsRouting** através da variável de instância **routing_**. Este método envia um "aknowledge" ao nó que originou o LSA e depois vai verificar se esse LSA altera a base de dados topológica. Se sim, invoca o método **computeRoutes** para que as rotas seja re-calculadas.

6.4.2.1 A classe LsTopoMap

A classe **LsTopoMap** mantém a base de dados topológica, que como já foi referido é basicamente o resultado da concatenação dos vários LSAs recebidos.

Um LSA é por sua vez uma lista de "estados de ligação". Cada estado de ligação corresponde basicamente ao estado da ligação entre dois nós, ou melhor, o estado da ligação entre um determinado nó e um nó vizinho. Um "estado da ligação" é representado pela estrutura **LsLinkState** que tem basicamente três variáveis, a identificação do vizinho em causa, uma variável **status** que indica se a ligação está activa ou não, e um inteiro que representa o custo da ligação. Um LSA é uma lista constituída por todos os "estados de ligação" referentes a todas as ligações entre um determinado nó e todos os seus vizinhos. Um LSA é representado por um tipo designado por **LsLinkStateList**.

A classe **LsTopoMap** é uma lista de **LsLinkStateList** indexada por nó origem. Tem os seguintes métodos principais:

- o método **insertLinkState**
- o método **update**

6.4.2.2 A classe LsRouting

A classe **LsRouting** é a classe que implementa o algoritmo do cálculo dos caminhos (que neste caso é o algoritmo de Dijkstra), e preenche a tabela de encaminhamento.

6.4.3 Implementação do CoSLSP no NS

A ideia subjacente consiste em usar a implementação do DiffServ/AF da Nortel incorporada na versão 2.1b8 do NS para dotar os nós da capacidade de diferenciar o tráfego, extendendo essa funcionalidade até ao processo de encaminhamento, ou seja cada nó além de dar um tratamento diferenciado aos pacotes dependendo da classe a que pertencem,

escolhe também a melhor rota para encaminhar determinado pacote em função da sua classe.

Ao usar a implementação do DiffServ/AF da Nortel, cada ligação é composta por diferentes filas, tantas quantas as classes de tráfego presentes na simulação. Utiliza-se as extensões feitas à implementação da Nortel (descritas na secção 6.4.1) para contabilizar de tempos a tempos a taxa de perdas verificadas em cada uma das filas que constitui uma ligação.

São essas taxas de perdas que vão ser difundidas nos LSAs juntamente com o custo e com o estado das ligações, e que vão constituir a base de dados topológica que existe em cada um dos nós. E é sobre essa informação que a nova lógica de encaminhamento irá actuar para calcular as "melhores" rotas que satisfazem os requisitos das várias classes.

Para implementar o CoSLSP no NS foi necessário desenvolver:

- Novos agentes de encaminhamento capazes de divulgar mensagens com as novas métricas (neste caso, as perdas registadas em cada uma das filas de espera);
- Uma nova lógica de encaminhamento de acordo com a descrição feita na secção 6.3.1;
- Novos classificadores que olham para a classe de serviço além do endereço destino e encaminham o pacote para a fila de espera associada.

Como era necessário desenvolver estes três componentes, optou-se por criar um novo módulo de encaminhamento: o **COSRoutingModule**.

6.4.3.1 Um novo agente de encaminhamento e lógica de encaminhamento

O novo agente de encaminhamento desenvolvido foi implementado através de um nova classe: a classe **OURLS**. Esta classe tem uma correspondente em C++: a classe **rtProtoOURLS**.

Esta classe é uma subclasse de **Agent/rtproto** e é muito semelhante à classe original **Agent/rtproto/LS**. A diferença mais relevante tem a ver com o conteúdo dos LSAs trocados pelos encaminhadores entre si. Enquanto o agente de encaminhamento que implementa o protocolo LS divulga apenas o estado das ligações (ou seja se as ligações estão activas ou não) e o seu custo, os novos agentes de encaminhamento divulgam também outros parâmetros de QoS, especificamente, a taxa de perdas verificada nas filas de espera correspondentes a cada uma das classes.

Para implementar esta nova funcionalidade, a classe **Agent/rtproto/OURLS** inclui o método **update-queue-drops**. Este método examina periodicamente o estado das várias filas de espera. Para isso invoca o método **dropsRate_changes** em cada uma das filas. Se este método retornar um inteiro diferente de zero, quer dizer se registaram alterações significativas no estado das filas. Nesse caso é invocado o método **intf-changed**.

O método **intf-changed** é invocado quando são detectadas alterações significativas nas filas e periodicamente pelo método **send-periodic-update**, quer haja ou não alterações nas filas.

O método **intf-changed** invoca o método **intfChanged** da correspondente classe em C++ (a classe rtProtoOURLS). Este método começa por indagar qual o estado de cada uma das filas, actualizando os diferentes valores, recalcula as rotas e envia os LSAs em função do novo estado.

6.4.3.2 Um novo classificador e um novo rtmodule

Além do agente de encaminhamento foi necessário desenvolver um novo classificador que olhasse para a classe dos pacotes, além de olhar para o endereço destino, antes de encaminhar um pacote para uma determinada fila. Este classificador foi implementado através da classe **COSRtg**, subclasse da classe **Classifier**, e com uma correspondente classe em C++, a classe **COSRtgClassifier**, subclasse também da classe **Classifier**.

O que distingue este classificador do classificador usado no encaminhamento *unicast* que acompanha o NS é a tabela de encaminhamento mantida. No caso deste último, a tabela de encaminhamento tem uma única chave, o endereço destino. Todas as pesquisas são efectuadas apenas usando esta única chave. No novo classificador desenvolvido e chave da tabela de encaminhamento é um par constituído pelo identificador da classe e pelo identificador do nó destino.

6.4.3.3 Um novo rtObject: o COSrtObject

Em vez de um **rtObject** cada nó possui um **COSrtObject**. Esta classe é em tudo semelhante à classe **rtObject**. A única coisa que as distingue é o facto da classe **COSrtObject** escolher a melhor rota por par <identificador de classe, destino> e não apenas pelo destino. Todas as variáveis de instância foram alteradas de forma a suportar esta funcionalidade. Assim, os *arrays* em vez de serem indexados através do identificador do nó destino passam a ter um par como chave: identificador de classe de serviço e destino.

6.5 Teste da Implementação

Foi usado o Network Simulator, versão 2.18b, para simular o CoSLSP e os resultados obtidos com este novo protocolo foram comparados com os da implementação LS.

6.5.1 Cenários de Simulação

A primeira topologia de rede usada foi novamente uma rede típica de um grande ISP[65] e está ilustrada na figura 6.3.

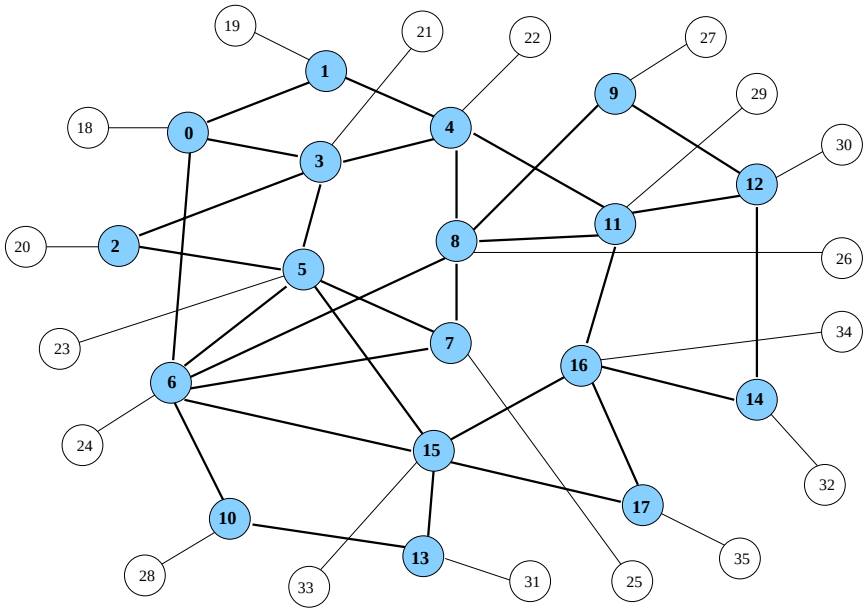


Figura 6.3: Topologia de rede típica de um grande ISP

Esta topologia inclui 36 nós: 18 são encaminhadores internos e os outros 18 são encaminhadores de fronteira. Cada um dos encaminhadores internos está ligado a um encaminhador de fronteira através de uma ligação simétrica de custo 1. Os nós internos estão ligados entre si através de 30 ligações assimétricas de forma a existirem diferentes caminhos alternativos com diferentes custos entre quaisquer pares de nós. Neste caso os custos das ligações são inteiros escolhidos aleatoriamente no intervalo [1, 10].

Foram considerados três classes de serviço distintas com diferentes requisitos de QoS expressos em termos da percentagem de perdas que conseguem tolerar. A classe 1, a mais exigente, consegue tolerar bem até 5% de perdas, a classe 2 tolera bem até 25% de perdas, enquanto a classe 3, a menos exigente, consegue tolerar até 50% de perdas. Cada ligação tem 3 filas físicas (uma por classe), tendo cada uma delas duas filas virtuais que correspondem a dois níveis diferentes de precedência em função das perdas ocorridas nas filas. Todas as filas foram configuradas exactamente da mesma forma, com o objectivo de evitar a diferenciação de tráfego no interior dos nós. Desta forma consegue-se garantir que a única diferenciação ocorrida é causada por acção do protocolo de encaminhamento.

Em cada simulação existem 18 fluxos activos entre dois pares de nós de fronteira escolhidos aleatoriamente entre todos os nós de fronteira da topologia. Os fluxos começam um de cada vez e são gerados de forma a criarem situações de congestão em todas as classes de serviço consideradas. Cada um dos fluxos é marcado numa classe entre as três classes consideradas. A classe de serviço a que pertence cada fluxo é escolhida aleatoriamente. Antes da simulação terminar os vários fluxos param, um de cada vez. Para concretizar esta experiência foi executado um conjunto de 100 simulações.

6.5.2 Análise dos resultados

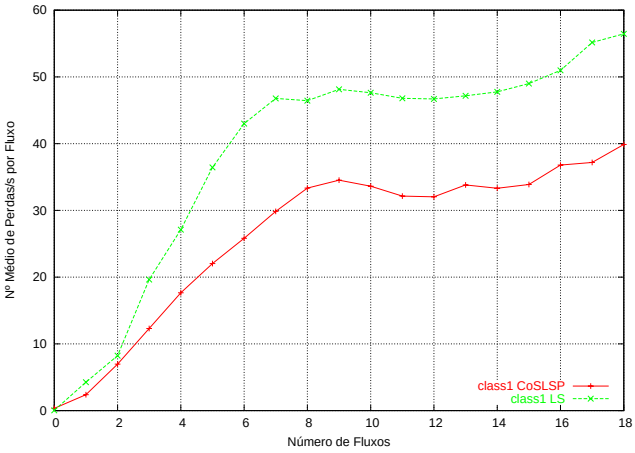
Os resultados das simulações são apresentados nas Figuras 6.4, 6.5, 6.6 e 6.7. Os resultados reflectem a média calculada depois de 100 simulações independentes.

As figuras 6.4(a), 6.4(b) e 6.4(c) mostram a média de perdas de pacotes ocorridas nos fluxos de cada classe de serviço quando são usados os protocolos CoSLSP e LS. Como era de esperar o números de perdas sofridas pelos fluxos de cada uma das diferentes classes de serviço é menor quando o protocolo CoSLSP é usado. Como se pode verificar pela análise da figura 6.4(d) esta diferença é maior na classe 1 do que na classe 2, e maior na classe 2 do que na classe 3. Isto deve-se ao facto da classe 1 ter requisitos de QoS mais exigentes em termos de perdas, do que a classe 2 e 3, e a classe 2 ter requisitos de QoS mais exigentes do que a classe 3.

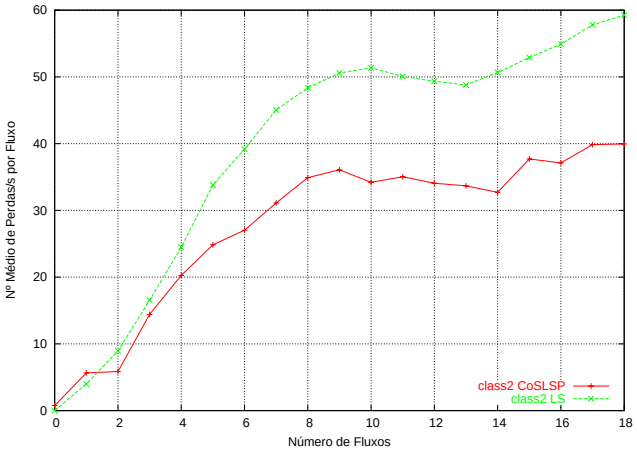
As figuras 6.5(a), 6.5(b), 6.5(c) mostram o custo médio dos caminhos por fluxo em cada classe de serviço quando são usados os dois protocolos (LS e CoSLSP). Quando o CoSLSP é usado, o custo dos caminhos escolhidos pelos diferentes fluxos é maior porque nem sempre é escolhido o caminho mais curto uma vez que este nem sempre consegue satisfazer os requisitos. No entanto esta diferença não é muito significativa uma vez que o algoritmo de descoberta de caminhos do protocolo CoSLSP escolhe sempre o caminho mais curto que consegue satisfazer os requisitos de QoS da respectiva classe de serviço. A figura 6.5(d) mostra o custo médio dos caminhos por fluxo em todas as classes de serviço quando é usado o protocolo CoSLSP. As diferenças observadas não são significativas, mas pode-se observar que o custo médio dos caminhos da classe 1 é ligeiramente maior do que o custo médio dos caminhos da classe 2, e o custo médio dos caminhos da classe 2 é também superior ao custo médio dos caminhos da classe 3. Isto explica-se devido ao facto dos requisitos da classe 1 serem mais exigentes do que os requisitos da classes 2 e 3 e por isso ser eventualmente necessário procurar caminhos de maior custo que os satisfaçam.

As figuras 6.6(a), 6.6(b), 6.6(c) mostram o número médio de ligações que constitui os caminhos por cada classe de serviço quando são usados os dois protocolos (LS e CoSLSP). A figura 6.6(d) mostra o número médio de ligações que constitui os caminhos em todas as classes de serviço quando é usado o protocolo CoSLSP. Neste caso observa-se que não há grande distinção entre estes protocolos no que diz respeito a esta métrica.

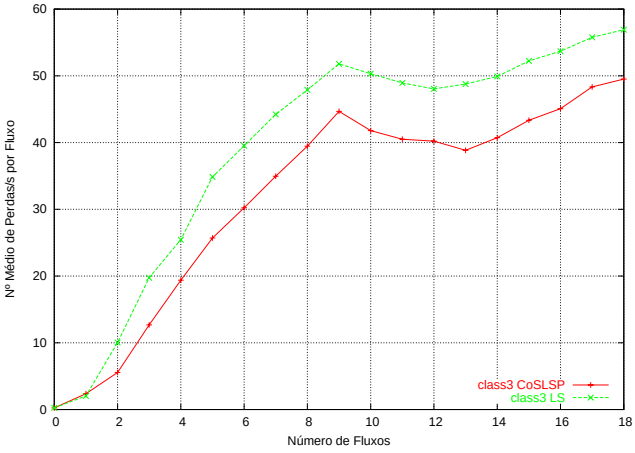
A figura 6.7 mostra o número médio de alterações nos caminhos verificadas em cada classe de serviço. Como seria de esperar este número é maior na classe 1 do que na classe 2 e maior na classe 2 do que na classe 3. Este facto ocorre porque a classe 1 tem requisitos de QoS maiores do que a classe 2 e a classe 2 requisitos mais exigentes do que a classe 3. Consequentemente, para manter estes requisitos, a classe 1 naturalmente tem que provocar mais mudanças nos caminhos, sendo de realçar que, apesar destas mudanças, a taxa de perdas se mantém controlada.



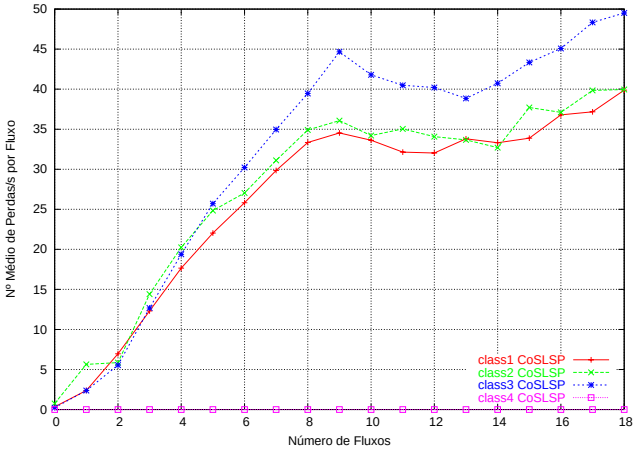
(a) *Perdas por fluxo - Classe 1*



(b) *Perdas por fluxo - Classe 2*



(c) *Perdas por fluxo - Classe 3*

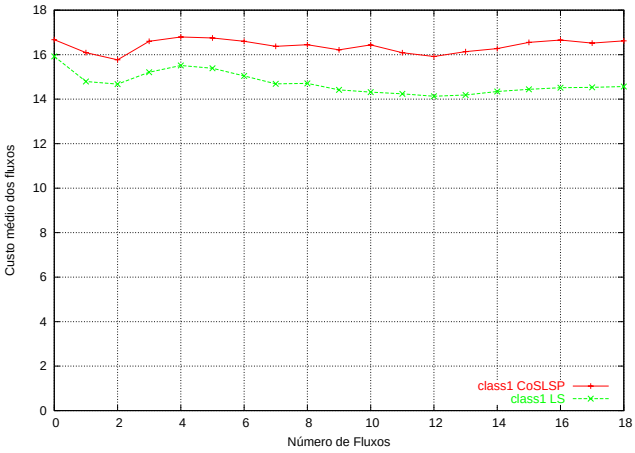


(d) *Perdas por fluxo nas diferentes classes - Protocolo CoSLSP*

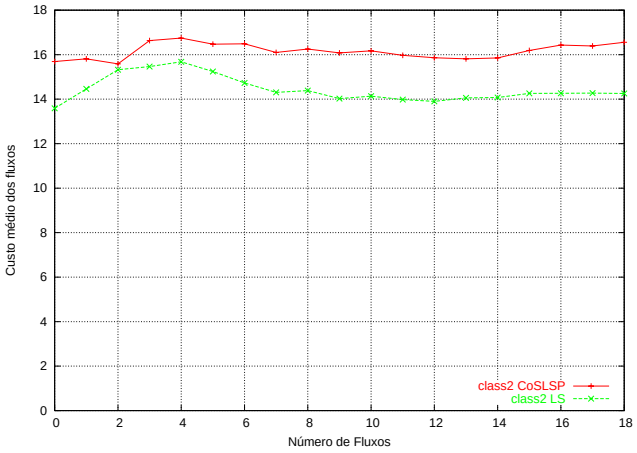
Figura 6.4: Resultados obtidos para a topologia típica de um ISP

6.6 Conclusões

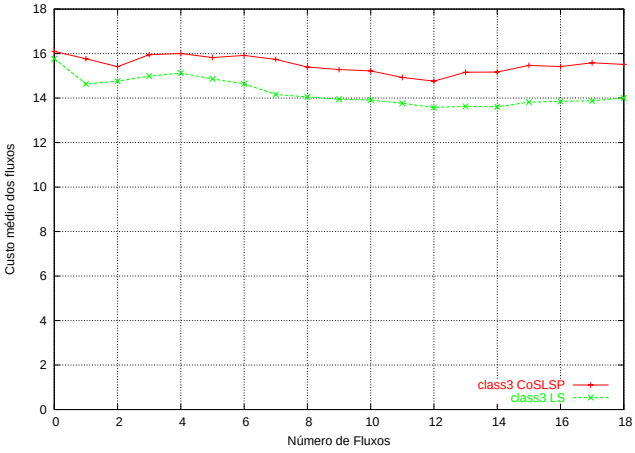
O CoSLSP é um protocolo de encaminhamento *unicast* por classes de serviço, adequado à utilização em ambientes do tipo DiffServ. É um protocolo de estado de ligação capaz de encontrar os caminhos mais curtos, se eles existirem, que satisfazem os requisitos de QoS das diferentes classes de serviço. A ideia básica consiste em determinar e instalar na tabela de encaminhamento um caminho distinto por cada classe de serviço considerada



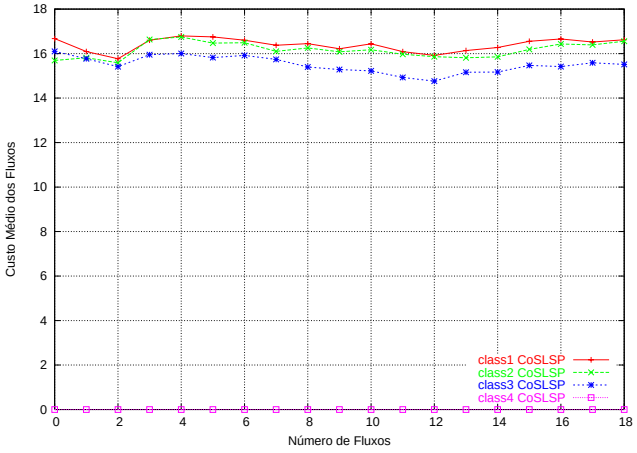
(a) *Custos Médios dos Caminhos - Classe 1*



(b) *Custos Médios dos Caminhos - Classe 2*



(c) *Custos Médios dos Caminhos - Classe 3*

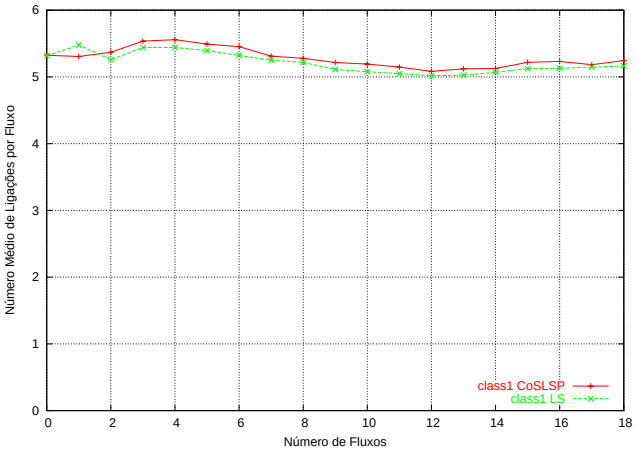


(d) *Custos Médios dos Caminhos nas diferentes classes - Protocolo CoSLSP*

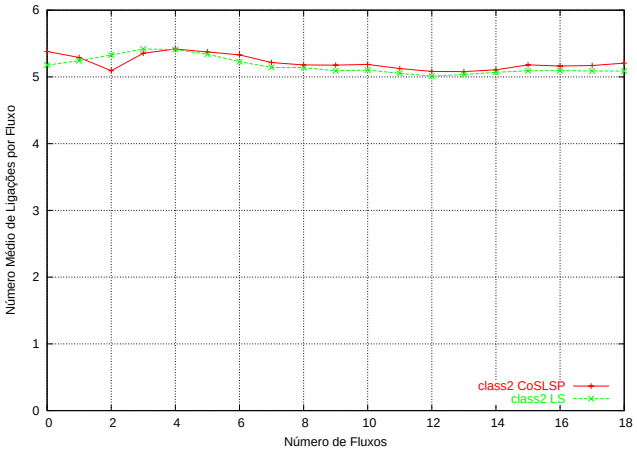
Figura 6.5: Resultados obtidos para a topologia típica de um ISP

num determinado domínio Diffserv. O processo de re-envio de pacotes também teve ser ser modificado no sentido de procurar a rota apropriada dependendo da classe de serviço em que cada pacote está "marcado".

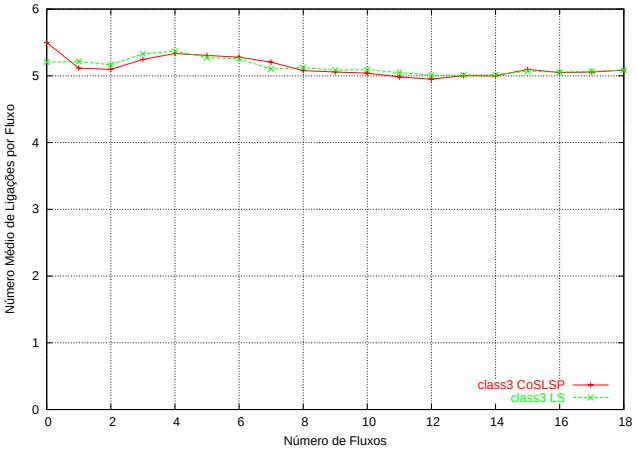
O CoSLSP foi implementado e avaliado com o Network Simulator. Os testes efectuadas mostraram que o CoSLSP nos casos em que se verifica congestão na rede, é capaz de encontrar "melhores" rotas em termos da métrica de QoS considerada. Além do objectivo principal de implementar o encaminhamento dependendo dos requisitos de QoS estabe-



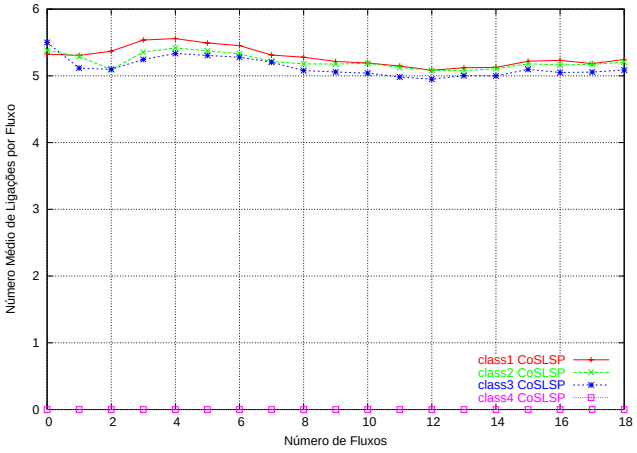
(a) *Número Médio de Ligações dos Caminhos - Classe 1*



(b) *Número Médio de Ligações dos Caminhos - Classe 2*



(c) *Número Médio de Ligações dos Caminhos - Classe 3*



(d) *Número Médio de Ligações dos Caminhos nas diferentes classes - Protocolo CoSLSP*

Figura 6.6: Resultados obtidos para a topologia típica de um ISP

lecidos, o protocolo CoSLSP procura também optimizar a utilização de recursos de rede. Este objectivo é conseguido através do algoritmo para o cálculo dos caminhos que procura sempre em primeiro lugar o caminho mais curto. Se este não satisfizer o requisitos de QoS especificados, o segundo caminho mais curto é calculado e assim sucessivamente até que um caminho exequível seja encontrado ou um limite imposto por configuração atingido.

Outro aspecto importante que o CoSLSP procura minimizar é a sobrecarga introduzida pelo protocolo devido às actualizações frequentes. As métricas de QoS alteram o

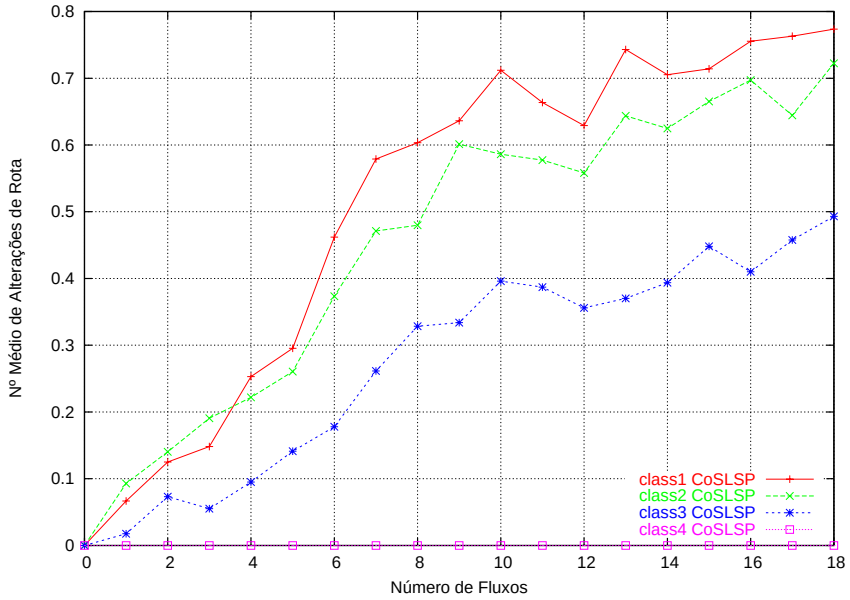


Figura 6.7: Alterações nos caminhos

seu valor com muito maior frequência do que por exemplo, o custo das ligações. Para responder a este potencial problema o CoSLSP desencadeia actualizações da informação de estado apenas quando é detectada uma alteração significativa no valor da métrica, em relação ao último valor anunciado.

Os protocolos de encaminhamento com QoS têm uma importante desvantagem que é o facto da métrica ou métricas que procuram optimizar (tipicamente o atraso fim-a-fim, ou percentagem de perdas) mudarem muito mais frequentemente do que o tradicional custo das ligações (número de saltos ou um custo imposto de forma administrativa). Em consequência disto os caminhos óptimos mudam também muito mais frequentemente, a menos que sejam usados mecanismos adicionais para prevenir as oscilações frequentes no encaminhamento. Com o CoSLSP este problema é minimizado, uma vez que este protocolo procura optimizar o habitual custo das ligações. Este custo só sofre alterações se uma ligação ou um nó falha, ou quando é efectuada alguma alteração nas configurações dos equipamentos. O CoSLSP só desencadeia o processo de alteração no encaminhamento se os caminhos previamente calculados deixam de satisfazer os requisitos de QoS das classes correspondentes, ou se por outro lado aparecem caminhos mais curtos que passam a satisfazê-los.

Capítulo 7

Encaminhamento Multicast por Classes de Serviço

Este capítulo descreve um protocolo de encaminhamento *multicast* por Classes de Serviço: o MCMRP (Multi-Class Multicast Routing Protocol). Basicamente é este protocolo que vem de encontro ao objectivo principal deste trabalho, o de estudar, conceber e testar estratégias de encaminhamento *multicast* por Classes de Serviço.

7.1 Enquadramento

Encaminhar tráfego *multicast* obriga à construção de uma ou mais árvores de distribuição. Os pacotes de dados são entregues aos diferentes destinatários usando essas árvores de distribuição. Assim sendo, o principal objectivo de um protocolo de encaminhamento *multicast* consiste em construir a árvore de custo mínimo. Este problema, o de encontrar uma árvore de custo mínimo é um problema NP_completo, e é designado por "Problema da árvore de Steiner"[67]. Foram propostas diferentes heurísticas para encontrar de forma eficiente árvores de distribuição *multicast*[76]. A heurística usada pela maioria dos protocolos de encaminhamento *multicast* consiste em construir uma árvore de distribuição adicionando um participante de cada vez através do melhor caminho encontrado desde o novo participante até ao nó mais perto dele que já pertence à árvore. Esta árvore é habitualmente designada por árvore do caminho inverso (*Reverse Path Tree*), precisamente porque é construída no sentido inverso ao sentido usado pelo tráfego *multicast*. Esta heurística assume que o custo das ligações é igual independentemente do sentido, ou seja que as ligações são simétricas. No entanto, as ligações podem ser assimétricas e nesse caso esta heurística pode conduzir a más árvores de distribuição de tráfego *multicast*.

Encontrar a árvore de custo mínimo em redes assimétricas é também um problema NP-Completo e designa-se por "Problema da árvore directa de Steiner". Existem alguns estudos teóricos baseados em grafos directos com o objectivo de constituírem apro-

ximações à resolução deste problema[68]. No entanto a maior parte dos protocolos de encaminhamento *multicast* desenvolvidos, como o CBT[36] e o PIM-SM[7] são baseados no encaminhamento do caminho inverso (*reverse path routing*).

Adicionalmente a maioria das aplicações *multicast* é por natureza sensível à qualidade de serviço disponível, e por essa razão poderão beneficiar do suporte de QoS ou CoS da rede subjacente, se algum deles estiver disponível. À semelhança do que se passa no encaminhamento *unicast* existem duas possíveis abordagens para fornecer Qualidade de Serviço ao nível do encaminhamento *multicast*: encaminhamento por fluxo e encaminhamento por classes de serviço. A primeira implementa o encaminhamento ao nível de cada fluxo individual. Foram propostas diferentes estratégias, baseando-se a maior parte delas na difusão de mensagens de controle que procuram caminhos satisfatórios em termos de requisitos de QoS para ligar um novo membro à árvore [61] [60]. A ideia subjacente é encontrar múltiplos caminhos possíveis. Entre os diferentes caminhos, o candidato o novo membro selecciona um que seja capaz de satisfazer os seus requisitos de QoS. Esta estratégia é adequada à arquitectura de serviços integrados (IntServ) que pretende disponibilizar garantias de QoS a cada fluxo individual que atravessa a rede através de reserva de recursos.

Como já foi referido no capítulo 6, o modelo de serviço integrados apresenta como grande desvantagem o facto de não ser uma arquitectura escalável. E uma das principais razões para isso é o facto de obrigar cada encaminhador a manter informação de estado por cada fluxo activo. Na arquitectura dos serviços diferenciados as garantias de QoS não são dadas a cada fluxo individual mas sim a um agregado de tráfego com o mesmo tipo de requisitos que se passa a designar por classe de serviço. Para se adequarem a este novo modelo, os protocolos de encaminhamento *multicast* terão que ser repensados, no sentido de não obrigarem a manutenção de informação de estado, por fluxo activo. Mais adequado é o cálculo de caminhos por classe de serviço de forma a satisfazer os requisitos, não de um fluxo individual mas sim os de um agregado, ou classe de serviço.

Neste capitulo descreve-se o protocolo MCMRP (Multi-Class Based Multicast Routing Protocol), um novo protocolo de encaminhamento *multicast* que constrói uma árvore por classe de serviço procurando satisfazer os diferentes requisitos das diferentes classes de serviço suportadas num determinado domínio.

7.2 Modelo

As ideias básicas subjacentes à implementação deste protocolo de encaminhamento *multicast* por Classes de Serviço são as seguintes:

- utilização de árvores directas, que consigam ter em consideração as assimetrias inerentes a estes contextos onde se pretende implementar a Qualidade de Serviço ao nível do encaminhamento;
- implementação de encaminhamento por classes de serviço, com rotas pré-estabelecidas,

em oposição ao encaminhamento por fluxo, a pedido;

- utilização quer de árvores partilhadas, quer de árvores centradas na fontes.

Procurou-se também que o protocolo desenvolvido estivesse de acordo com o modelo actual utilizado pelo *multicast* IP, ou seja, o protocolo de encaminhamento deverá suportar a junção de fontes e destinatários em qualquer altura e de forma assíncrona, e não exigir qualquer tipo de conhecimento prévio acerca da constituição do grupo.

7.2.1 Algoritmo para a construção das árvores

Numa primeira fase, é proposto um mecanismo para construir múltiplas árvores partilhadas, uma por cada classe de serviço presente, isto de forma a possibilitar que os diferentes receptores se juntem ao grupo sem saberem de antemão quem são as fontes e onde estão localizadas. Por outro lado sem um conhecimento completo acerca das fontes, também se torna complicado para um receptor definir os seus requisitos de QoS. Ele não sabe qual o volume tráfego que está em jogo, além de desconhecer a sua proveniência. Assim faz sentido que numa fase inicial sejam as fontes a definir os seus requisitos de QoS e a marcar os pacotes de acordo com esses requisitos ficando os receptores a receber o tráfego proveniente das diferentes fontes de acordo com os requisitos definidos por estas. As múltiplas árvores partilhadas são centradas num ponto pré-definido, também aqui designado por Rendez-Vous Point. O facto de se suportar múltiplas árvores não quer dizer que haja duplicação de tráfego, O Rendez-Vous Point deverá re-enviar os pacotes marcados numa determinada classe de serviço apenas através da árvore pertencente a essa classe de serviço, um vez que existirá uma árvore por cada classe de serviço suportada.

O mecanismo de construção destas árvores suporta as assimetrias já que constrói todas elas de forma directa e não invertida. Quando um novo membro decide juntar-se ao grupo, o encaminhador nomeado (que é quase sempre o que está mais próximo do candidato a novo membro do grupo), envia um pedido de junção explícito em direcção ao RP do grupo. Todos os encaminhadores ao longo do caminho desde o novo membro até ao RP do grupo não fazem mais nada a não ser re-enviar o pacote na direcção do RP, ou seja, nesta fase não é introduzida informação de estado nos encaminhadores. Quando o RP recebe um pedido de junção de um novo membro deverá enviar uma mensagem de resposta por cada classe suportada. Estas mensagens deverão, todas elas, atingir o novo membro, após terem percorrido o melhor caminho calculado para a respectiva classe de serviço. Todos os encaminhadores ao longo destes caminhos deverão, ao receber uma mensagem deste tipo (mensagem de *join-acknowledge*) actualizar as suas tabelas de encaminhamento *multicast* em conformidade. Basicamente trata-se de registar na respectiva entrada da tabela de encaminhamento a interface de entrada e de saída usadas pela mensagem de *join-acknowledge*.

O processo de junção ao grupo proposto pelo protocolo MCMRP está devidamente detalhado na figura 7.1. Na situação ilustrada existem 4 fontes e 4 receptores. A fonte $S1$ e $S4$ estão a transmitir pacotes marcados na classe j enquanto as fontes $S2$ e $S3$ estão a

transmitir pacotes marcados na classe i. Os 4 receptores juntaram-se às duas árvores de forma a poderem receber o tráfego proveniente da 4 fontes.

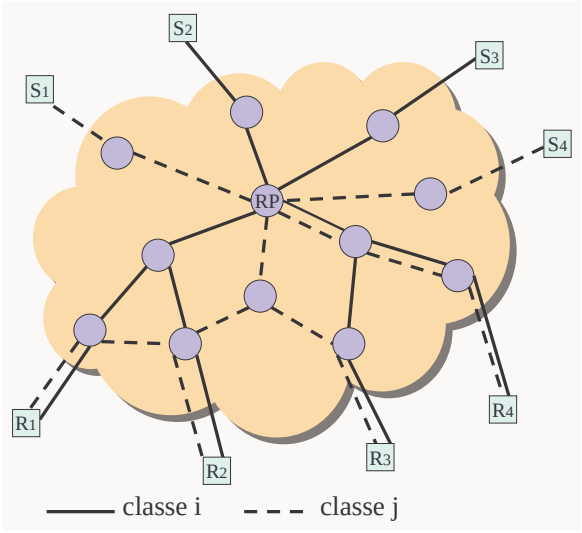


Figura 7.1: Múltiplas árvores partilhadas (uma por classe de serviço), segundo o protocolo MCMRP

O mecanismo de múltiplas árvores partilhadas descrito, não permite que os diferentes destinatários do tráfego *multicast* especifiquem os seus requisitos de QoS. O tráfego flui desde as fontes até aos diferentes receptores de acordo apenas com os requisitos de QoS definidos pelas fontes. Após o período inicial, um determinado receptor poderá requerer a reclassificação do tráfego proveniente de determinada fonte. Este objectivo não pode ser concretizado através das árvores partilhadas. Por isso, adicionalmente o MCMRP suporta também árvores centradas na fontes, árvores essas que irão permitir que os destinatários tenham também a possibilidade de especificar os seus requisitos de QoS.

Quando inicia o pedido de junção a uma fonte o destinatário deve incluir no pedido a classe de serviço desejada. Cabe à fonte decidir se está ou não em condições de aceitar tal pedido, sabendo que, se o aceitar, terá que gerar tráfego marcado na classe de serviço solicitada.

Cada umas das fontes pode receber diferentes pedidos de diferentes destinatários para diferentes classes de serviço dentro de um mesmo grupo *multicast*. No limite, para grandes grupos, pode haver pedidos em todas as classes de serviço suportadas pelo domínio. Mesmo na pior situação não deverão parecer problemas de escalabilidade porque o número de diferentes classes de serviço suportadas será sempre muito menor do que o número de destinatários. Na prática esta solução implica uma árvore centrada na fonte por cada classe de serviço a menos que seja estabelecida alguma ordem de precedência entre as classes.

Quando aceita um pedido de junção para uma nova classe de serviço a fonte deve

gerar uma mensagem de aceitação (*acknowledge message*) endereçada ao destinatário que requereu a junção à fonte numa determinada classe de serviço. A partir daqui o mecanismo de construção de um novo ramo da árvore de distribuição é semelhante ao descrito a cima para a construção de árvores partilhadas. Mas neste caso é gerada apenas uma mensagem de *join-acknowledge* por cada pedido de junção. Podem ocorrer duas situações distintas: o destinatário pode pedir para se juntar a uma árvore centrada na fonte na mesma classe de serviço em que a fonte está a emitir, ou pode requerer tráfego marcado numa outra classe de serviço diferente da classe de serviço por defeito.

No primeiro caso, quando um encaminhador no caminho entre a fonte e o novo destinatário recebe a mensagem de *join-acknowledge* se não pertence ainda à árvore centrada na fonte na classe de serviço requerida, deverá criar uma entrada `(i,S,G)` e copiar todas as interfaces de saída da entrada `(i,*,G)` para a nova entrada `(i,S,G)` criada. Isto é necessário porque no futuro os pacotes provenientes da fonte `S` vão ser re-enviados com base nesta entrada e já não na entrada `(i,*,G)`. Além disso quando um encaminhador no caminhos entre a fonte e o novo destinatário começa a receber tráfego da fonte através da interface de entrada que consta na entrada `(i,S,G)`, deverá enviar uma mensagem de *prune-request* através da árvore partilhada pertencente aquela classe de serviço. Esta mensagem de *prune-request* indica que os pacotes provenientes da fonte `S` na classe `i` não deverão ser re-enviados através da árvore partilhada para aqueles nós uma vez que estes já estão a receber esses pacotes via árvore centrada na fonte. Este mecanismo é implementado enviando uma mensagem de *prune-request* ao vizinho imediatamente a cima na árvore partilhada da classe `i`. Quando um encaminhador na árvore partilhada da classe `i` recebe este tipo de *prunes* cria uma entrada do tipo `(i,S,G)RPT`, muito à semelhança do que acontece no protocolo PIM-SM. A lista de interfaces de saída da entrada `(i,S,G)RPT` é copiada da lista de interfaces de saída da entrada `(i,*,G)` eliminando a interface usada para atingir o nó que gerou a mensagem de *prune-request*, que como é obvio pode não ser a interface por onde chegou a mensagem de *prune-request*. Estas entradas `(i,S,G)RPT` também deverão ser actualizadas quando chega uma nova mensagem de *join-acknowledge* para permitir a junção de um novo membro numa árvore com informação de estado criada por *prunes* de fontes nas árvores partilhadas.

Quando um destinatário decide juntar-se a uma fonte requerendo uma classe de serviço diferente da classe de serviço usada à partida pela fonte, este processo tem que ser diferente. Quando a entrada `(i,S,G)` é criada a lista de interfaces de saída não deverá ser copiada da entrada `(i,*,G)`, porque, neste caso, os destinatários que estão ligados à árvore partilhada deverão continuar a receber tráfego na classe de serviço original daquela fonte, e não na nova classe requerida. Pela mesma razão estas entradas não deverão ser actualizadas quando chega ao encaminhador uma nova mensagem de *join-acknowledge*. Adicionalmente o *prune* da fonte na árvore partilhada deve ser disparado pelo encaminhador nomeado quando este recebe as mensagens de *join-acknowledge*. Neste caso as mensagens de *prune-request* deverão ser enviadas ao longo de todas as árvores partilhadas com excepção da árvore partilhada pertencente à classe de serviço para a qual o destinatário comutou. Este mecanismo é necessário porque uma vez que o destinatário comutou para

outra classe não quer continuar a receber daquela fonte na classe por defeito.

O processo de comutação para uma árvore centrada na fonte está detalhado na figura 7.2. Na situação ilustrada o receptor $R1$ comutou para uma árvore centrada na fonte $S1$, requerendo a classe j. No entanto não abandona a árvore partilhada pertencente à classe j para poder continuar a receber o tráfego proveniente da outras fontes que estão a emitir nessa classe (neste caso, a fonte $S4$). São colocadas entradas do tipo `(j,S1,G)` nos nós, para evitar que o receptor $R1$ receba tráfego via árvore partilhada da fonte $S1$.

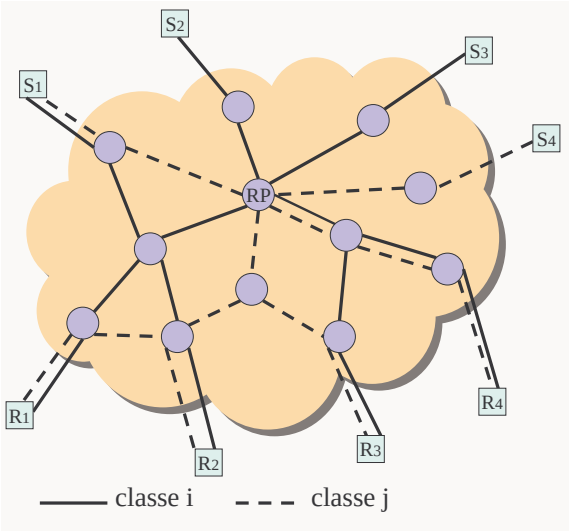


Figura 7.2: Árvore centrada numa fonte e árvores partilhadas

7.2.2 Informação de estado mantida nos nós

As entradas na tabela de encaminhamento têm os mesmos campos que os propostos no PIM-SM e alguns adicionais:

- À semelhança do que acontece no DTMP também neste caso foi introduzida a identificação do vizinho imediatamente acima na árvore. O motivo é o mesmo: facilitar a implementação do mecanismo de poda dos ramos da árvore (*prune*).
- Além das *flags* de caracterização das rotas, RPT-Bit e SPT-Bit, usadas no PIM-SM, foi introduzida a *flag* CST-Bit (*Class-of-Service Tree Bit*. Esta *flag* é usada para distinguir as entradas que foram criadas quando um receptor comutou para uma árvore centrada na fonte na classe por defeito, das que são criadas quando um receptor ao comutar para uma árvore centrada na fonte requer a reclassificação do tráfego. Se o receptor se junta a uma fonte na classe por defeito esta *flag* é colocada a 0, senão é colocada a 1. Nas entradas do tipo `(j,*,G)` esta *flag* também existe mas não tem qualquer significado, e é sempre colocada a 0.

7.3 Implementação

O MCMRP foi implementado utilizando o protocolo DTMP como ponto de partida. O DTMP (Directed Trees Multicast Protocol) é um protocolo de encaminhamento *multicast* que constrói árvores directas em vez das habituais árvores invertidas. A descrição completa deste protocolo, bem como os pormenores da implementação e os resultados obtidos através da simulação podem ser encontrados no capítulo 5.

O DTMP suporta árvores partilhadas e árvores centradas na fontes à semelhança do protocolo PIM-SM, de forma a colher as vantagens de ambas as estratégias. Foi concebido para ser utilizado em redes assimétricas onde os custos das ligações entre dois nós são distintos nos dois sentidos.

Outro dos componentes chave do protocolo MCMRP é o protocolo de encaminhamento *unicast* subjacente. Apesar do MCMRP ser independente do protocolo de encaminhamento *unicast* utilizado, este tem que ser capaz de calcular diferentes rotas mediante os requisitos das diferentes classes de serviço suportadas. Para conseguir construir a árvore de distribuição *multicast* mais adequada a cada classe de serviço, o MCMRP procura, sempre que precisa de construir um ramo de uma das árvores, a rota mais adequada para satisfazer os requisitos de QoS da respectiva classe. Essa pesquisa é concretizada consultando a tabela de encaminhamento *unicast* que deverá então ter a rotas por classe de serviço. Para ir de encontro a esta necessidade concebeu-se e concretizou-se o CoSLSP, um protocolo de encaminhamento *unicast* por classes de serviço, que foi descrito no capítulo 6.

Recorde-se que o CoSLSP tem como objectivo implementar um esquema de encaminhamento *unicast* por classes de serviço. A ideia subjacente consiste em encontrar uma rota para cada classe de serviço capaz de satisfazer os requisitos de QoS pré estabelecidos para essa classe. Adicionalmente este protocolo tenta também contribuir para o problema da optimização de utilização de recursos de rede. Assim, além de calcular apenas as rotas que podem satisfazer os requisitos de calda classe o CoSLSP tenta encontrar o caminho mais curto que os satisfaz. É um protocolo de estado de ligação que utiliza uma variante do algoritmo de Dijkstra capaz de encontrar as rotas mais curtas caso elas existam capazes de satisfazer determinados requisitos de QoS. De forma sucinta: o algoritmo de cálculo das rotas começa por procurar a rota mais curta; as métricas de QoS dessa rota são comparadas com os requisitos de QoS de determinada classe; se a rota for exequível é instalada na tabela de encaminhamento, senão a próxima rota mais curta é calculada; este processo continua até que seja encontrada uma rota exequível ou até que um limite imposto por configuração seja atingido. Desta forma é calculada e instalada na tabela de encaminhamento uma rota por cada classe de serviço, sendo o re-encaminhamento de tráfego modificado por forma a permitir as pesquisas por par <destino, classe de serviço>.

A implementação do MCMRP desenvolvida no NS é constituída por:

1. Um agente de encaminhamento que implementa a construção das árvores de distri-

buição *multicast* de acordo com a estratégia seguida pelo MCMRP;

2. Um novo classificador que implementa o re-envio do tráfego segundo o MCMRP.

Estas duas componentes são descritas com maior detalhe nas próximas secções.

7.3.1 O agente de encaminhamento MCMRP

A seguir são descritos de forma detalhada os métodos implementados. Estes métodos são basicamente os mesmo que foram construídos para implementar o agente de encaminhamento DTMP, embora a maior parte deles tenha sido re-escrito no sentido de suportar as múltiplas árvores, tanto as partilhadas como as árvores centradas na fontes.

- **Método join-group**

  Este método deverá ser invocado por um agente quando este decide juntar-se a um grupo ou quando decide comutar da árvore partilhada para uma árvore centrada na fonte.

  Se for invocado com apenas um argumento, o método depreende que o agente pretende juntar-se ao grupo (ou seja, à árvore partilhada) e invoca o método **join-rpt**.

  Se for invocado com três argumentos quer dizer que o agente já faz parte do grupo e pretende apenas comutar da árvore partilhada para uma árvore centrada na fonte numa determinada classe. Neste caso o método invoca o método **join-spt**.

- **Método join-rpt**

  O método **join-rpt** é invocado quando um determinado agente quer juntar-se ao grupo. Como já foi referido, quando um determinado nó se quer juntar a um grupo necessita, numa primeira fase tem de se juntar às diferentes árvores partilhadas.

  Assim sendo, para cada classe suportada é necessário antes de mais verificar se o nó ainda não pertence à árvore partilhada da respectiva classe, ou seja, é necessário verificar se na tabela de encaminhamento *multicast* existe alguma entrada $(*, G, i)$, sendo G o grupo em causa e i o identificador da classe. Se a entrada não existir é necessário criá-la e inicializá-la. Este tipo de entrada é inicializada com a seguinte informação:

  - A interface de entrada (iif) é inicializada com -1
  - O campo destinado ao endereço do vizinho na árvore é inicializado com o endereço do próprio nó.
  - As *flags* de caracterização da rota, RPT-Bit e SPT-Bit, são inicializadas respectivamente com os valores 1 e 0. A *flag* CST-Bit é inicializada com o valor 0.

Quer a entrada tenha acabado de ser criada, quer no caso de já existir mas com a lista de interfaces de saída vazia, é necessário construir uma mensagem de *join-request* e propagá-la em direcção ao RP. Se a entrada já existir e não tiver a lista de interfaces de saída vazia, só é necessário enviar a mensagem de *join-request* no caso de existir uma entrada do tipo (S, G, i) com o RPT-Bit a 1. Esse tipo de entradas têm a ver com o processo de comutação para árvores centradas nas fontes. São criadas pelas mensagens de *prune* de uma fonte numa árvore partilhada, mensagens essas que são enviadas sempre que é necessário retirar o tráfego proveniente de uma fonte da árvore partilhada. Isso acontece porque quando um nó comuta para uma árvore centrada numa fonte quer continuar a receber o tráfego de outras fontes pela árvore partilhada, mas, como é obvio, tem de deixar de receber o tráfego dessa fonte pela árvore partilhada uma vez que já o recebe pela árvore centrada na fonte. Quando há informação de estado deste tipo na árvore *multicast* é necessário ter alguns cuidados. Neste caso o *join-request* é propagado para garantir que os novos membros que se estão agora a juntar vão receber de todas as fontes através da árvore partilhada.

Em qualquer das situações é necessário acrescentar o agente que originou o pedido de junção à lista de interfaces de saída das entradas $(*, G, i)$, quer tenham sido criadas pelo procedimento, quer já lá existissem.

O processo de junção à árvore partilhada numa rede onde são consideradas duas classes de serviço, 1 e 2, está esquematizado na figura7.3.

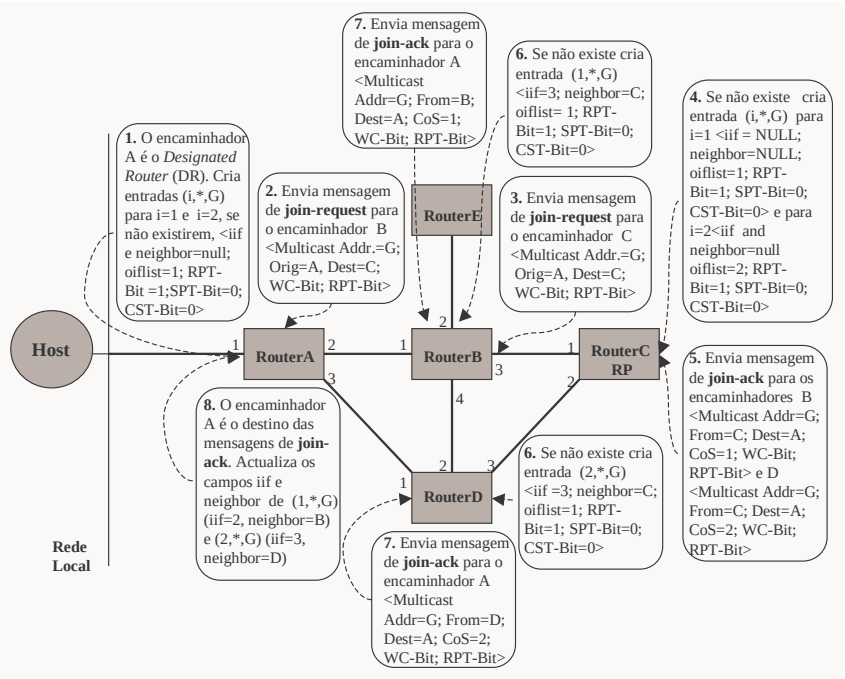


Figura 7.3: Implementação da construção da árvore partilhada segundo o protocolo MCMRP. As acções estão numeradas pela ordem em que ocorrem

- **Método join-spt**

  O método **join-spt** é invocado quando um determinado agente, que já faz parte do grupo, quer comutar para uma árvore centrada numa fonte numa determinada classe.

  Antes de mais é necessário verificar se o nó já faz parte de uma árvore centrada na fonte para aquela classe. Ou seja, é necessário verificar se na tabela de encaminhamento *multicast* existe alguma entrada (S, G, i), sendo G o grupo em causa e i o identificador da classe requerida pelo destinatário. Se não existir é necessário criá-la e inicializá-la. Este tipo de entrada é inicializada com a seguinte informação:

  - A interface de entrada (iif) é inicializada com -1
  - O campo destinado ao endereço do vizinho na árvore é inicializado com o endereço do próprio nó.
  - As *flags* de caracterização da rota, RPT-Bit e SPT-Bit, são inicializadas ambas com o valor 0.
  - A *flag* CST-Bit é preenchida a 0 ou 1, consoante se trate de uma junção a fonte na mesma classe que esta está a usar por defeito ou uma junção requerendo uma classe de serviço diferente.

  Quer a entrada tenha acabado de ser criada, quer no caso de já existir mas com a lista de interfaces de saída vazia, é necessário construir uma mensagem de *join-request* e propagá-la em direcção à fonte.

  Se a entrada já existir e não tiver a lista de interfaces de saída vazia, só é necessário enviar a mensagem de *join-request* no caso de ser uma entrada com o RPT-Bit a 1. O facto deste bit estar activo quer dizer que a entrada apesar de ser uma entrada (S, G, i), contém informação a respeito da árvore partilhada e não a respeito de uma árvore centrada na fonte S.

  Em qualquer das situações é necessário acrescentar o agente que originou o pedido de junção, à lista de interfaces de saída da entrada (S, G, i), quer tenha sido criada pelo procedimento, quer já lá existisse.

  O processo de comutação de uma árvore partilhada para uma árvore centrada na fonte está esquematizado na figura 7.4. Na situação ilustrada o destinatário decide comutar para uma árvore centrada na fonte na classe de serviço 1, supondo que a classe por defeito em que a fonte originalmente esta a emitir é a classe de serviço 2.

- **Método recv-join**

  Se se tratar de um encaminhador que está algures no caminho entre o nó que originou o pedido de junção e o RP ou a fonte, este método limita-se a propagar a mensagem de *join-request* em direcção ao RP ou em direcção à fonte.

  Se foi o RP que recebeu a mensagem de *join-request* e tratando-se de um pedido de junção ao grupo, então é necessário criar ou actualizar as entradas $(*, G, i)$ de todas

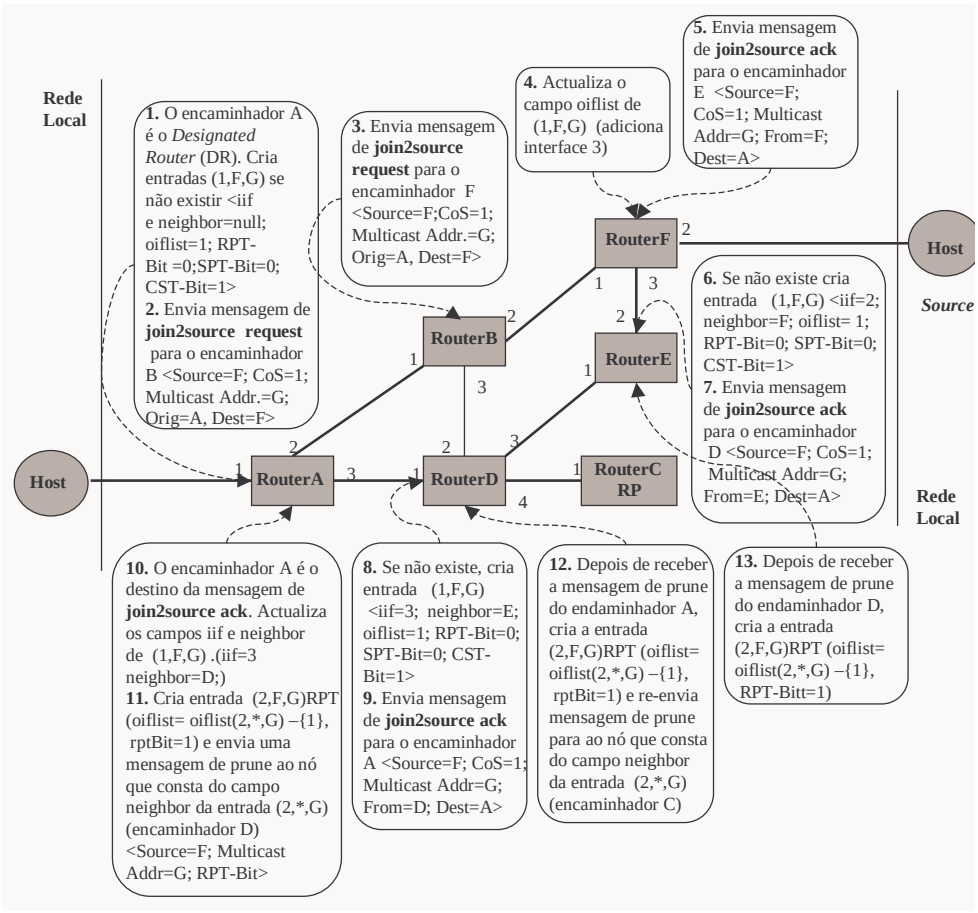


Figura 7.4: Processo de comutação da árvore partilhada para uma árvore centrada na fonte

as classes suportadas, e depois enviar uma mensagem de *join-acknowledge* por cada classe. Essas mensagens deverão ser "marcadas" nas classes de serviço respectivas de forma a seguirem os "melhores" caminhos para cada uma da classes de serviço.

Se foi a fonte que recebeu uma mensagem de *join-request* a ela destinada então é necessário criar ou actualizar a entrada (S, G, i) e enviar uma mensagem de *join-acknowledge* "marcada" na classe de serviço solicitada, endereçada ao nó que requereu a junção à fonte.

Ao criar uma nova entrada (S, G, i), é necessário ter dois aspectos em consideração:

- A *flag* de caracterização da rota, CST-Bit, fica com o valor 0 ou 1 dependendo de se tratar de uma junção à fonte na classe em que a fonte está a transmitir por defeito ou não, respectivamente.
- a lista de interfaces de saída é copiada da entrada (*,G,i) ou não dependendo também de se tratar de uma junção à fonte na classe em que a fonte está a

transmitir por defeito ou não, respectivamente.

- *Método recv-ack*

  Este método é invocado sempre que um nó recebe uma mensagem de *join-acknowledge*. Há que distinguir quatro situações diferentes:

  - A mensagem de *join-acknowledge* é proveniente do RP e por isso destina-se a criar mais um ramo numa das árvores partilhadas.
  - A mensagem de *join-acknowledge* é proveniente de uma fonte e por isso destina-se a criar mais um ramo numa árvore centrada numa fonte.
  - Dentro da primeira situação podemos distinguir outras duas situações:
    - Quem recebeu a mensagem de *join-acknowledge* foi um nó no caminho entre o RP e o candidato a novo membro do grupo
    - Quem recebeu a mensagem de *join-acknowledge* foi o candidato a novo membro do grupo
  - Dentro da segunda situação podemos distinguir outras duas situações:
    - Quem recebeu a mensagem de *join-acknowledge* foi um nó no caminho entre a fonte e o membro que iniciou a comutação para uma árvore centrada na fonte
    - Quem recebeu a mensagem de *join-acknowledge* foi o membro que iniciou a comutação para uma árvore centrada na fonte

  A primeira situação distingue-se da segunda testando as *flags* RPT-Bit e WC-Bit. As outras distinguem-se comparando a identificação do nó que recebeu a mensagem com a identificação do nó a quem se destina a mensagem (este segundo identificador está incluído num dos campos da própria mensagem).

  Se o nó que recebeu a mensagem de *join-acknowledge* está na primeira situação, ou seja constata que recebeu uma mensagem de *join-acknowledge* proveniente do RP, vai verificar se é ele o nó a que se destina a mensagem de *join-acknowledge* ou não. Se não for desencadeia a seguinte sequência de acções:

  - Verifica se já possui uma entrada (*,G,i). Se sim, significa que já faz parte da árvore partilhada. Se não, é porque ainda não faz, e nesse caso a primeira coisa a fazer é criar uma entrada (*,G,i), inicializada de forma conveniente:
    - A interface de entrada (iif) deverá ficar com o valor da interface por onde chegou a mensagem de *join-acknowledge*.
    - O campo destinado ao endereço do vizinho na árvore deverá ficar com o endereço do nó que enviou a mensagem de *join-acknowledge*.
    - A *flag* de caracterização da rota, RPT-Bit, fica com o valor 1.
    - A *flag* de caracterização da rota, SPT-Bit, fica com o valor 0.
    - A *flag* de caracterização da rota, CST-Bit, fica com o valor 0.

- Se a entrada já existe é necessário verificar se a interface de entrada que consta da entrada coincide com a interface por onde chegou a mensagem de *join-acknowledge*. Se não coincidir é porque estamos perante um processo de reconstrução da árvore. Nesse caso é necessário alterar a interface de entrada e o campo que contem a identificação do vizinho para os novos valores e enviar um *prune* no antigo ramo da árvore partilhada para acabar com esse estado.
- Em qualquer das situações é necessário descobrir qual é a melhor interface para atingir, de acordo com a classe de serviço em causa, o candidato a novo membro. não só para propagar nesse sentido a mensagem de *join-acknowledge*, mas também para acrescentar essa interface à lista de interfaces de saída da entrada (*,G,i).
- Depois é necessário verificar se há entradas (S,G,i) na tabela de encaminhamento *multicast* do nó. Se houver, é necessário verificar de que tipo de entrada se trata, e para cada uma das situações agir:
  * A primeira *flag* a testar é a *flag* designada por CST-Bit. Esta *flag* está activa quando a entrada diz respeito a uma árvore centrada na fonte numa classe diferente da classe que a fonte está a usar por defeito. Se se verificar que a entrada (S,G,i) é uma entrada deste tipo, não se deverá acrescentar a interface de saída à entrada. Se a *flag* estiver a 0 é necessário acrescentar a interface de saída à lista de interfaces de saída da entrada (S,G,i).
  * Depois é necessário verificar se a entrada tem o RPT-Bit a 1. Se tiver, compara-se a lista de interfaces de saída com a lista de interfaces de saída da entrada (*,G,i) e, caso sejam iguais deve-se apagar a entrada (S,G,i)RPT.

Se o nó for o destino da mensagem de *join-acknowledge*, ou seja se se tratar do nó que requereu a junção ao grupo (o candidato a novo membro), a única coisa a fazer é verificar se a interface de entrada que consta da entrada (*,G,i) coincide com a interface por onde chegou a mensagem de *join-acknowledge*. Se não coincidir é porque se está perante um processo de reconstrução da árvore. Nesse caso é necessário alterar a interface de entrada e o campo que contem a identificação do vizinho para os novos valores e enviar um *prune* no antigo ramo da árvore partilhada para acabar com esse estado.

Na segunda situação, ou seja, quando se verifica que a mensagem de *join-acknowledge* diz respeito a uma junção a uma árvore árvore centrada numa fonte, a primeira coisa a fazer também é verificar se o nó que recebeu a mensagem de *join-acknowledge* é o nó a que se destina a mensagem ou um nó a meio do caminho entre a fonte e o membro do grupo que requereu a junção à fonte.

Se for um nó a meio do caminho entre a fonte e o nó que requereu a junção é necessário desencadear a seguinte sequência de acções:

- Antes de mais é necessário verificar se já existe na tabela de encaminhamento *multicast*, uma entrada (S,G,i). Se existir, significa que o nó já faz parte da

árvore centrada na fonte naquela classe. Se não, é porque ainda não faz, e nesse caso a primeira coisa a fazer é criar uma entrada (S,G,i), inicializada de forma conveniente:

  * A interface de entrada (iif) deverá ficar com o valor da interface por onde chegou a mensagem de *join-acknowledge*.
  * O campo destinado ao endereço do vizinho na árvore deverá ficar com o endereço do nó que enviou a mensagem de *join-acknowledge*.
  * A *flag* de caracterização da rota, RPT-Bit, fica com o valor 0.
  * A *flag* de caracterização da rota, SPT-Bit, fica com o valor 0.
  * A *flag* de caracterização da rota, CST-Bit, fica com o valor 0 ou 1 dependendo de se tratar de uma árvore na classe em que a fonte está a transmitir por defeito ou não, respectivamente.
  * a lista de interfaces de saída é copiada da entrada (*,G,i) ou não dependendo também de se tratar de uma árvore na classe em que a fonte está a transmitir por defeito ou não, respectivamente.

- Se a entrada já existe é necessário verificar se é uma entrada com a *flag* RPT-Bit a 1 ou a 0. Se for uma entrada com a *flag* RPT-Bit a 1 é necessário mudar o bit para 0 bem como a interface de entrada e o nó vizinho.
- Se existe a entrada e tem o RPT-Bit a 0 é necessário verificar se a interface de entrada que consta da entrada coincide com a interface por onde chegou a mensagem de *join-acknowledge*. Se não coincidir é porque estamos perante um processo de reconstrução da árvore. Nesse caso é necessário alterar a interface de entrada e o campo que contem a identificação do vizinho para os novos valores e enviar um *prune* no antigo ramo da árvore para acabar com esse estado.
- Em qualquer das situações é necessário descobrir qual é a melhor interface para atingir, de acordo com a classe de serviço em causa, o nó que requereu a junção à fonte, não só para propagar nesse sentido a mensagem de *join-acknowledge*, mas também para acrescentar essa interface à lista de interfaces de saída da entrada (S,G,i).

Se o nó a que chegou a mensagem de *join-acknowledge* for o próprio nó que requereu a junção à fonte então a sequência de acções a desencadear é a seguinte:

- verificar se a interface de entrada que consta da entrada (S,G,i) coincide com a interface por onde chegou a mensagem de *join-acknowledge*. Se não coincidir é porque se está perante um processo de reconstrução da árvore. Nesse caso é necessário alterar a interface de entrada e o campo que contem a identificação do vizinho para os novos valores e enviar um *prune* no antigo ramo da árvore partilhada para acabar com esse estado.
- verificar se a entrada (S,G,i) tem o RPT-Bit a 1 e se tiver, mudá-lo para 0

- por fim verificar se se trata de uma junção na mesma classe em que a fonte está a transmitir por defeito ou não. Se se tratar de outra classe é necessário criar uma entrada (S,G,i)RPT na classe em que a fonte está a transmitir por defeito e podar essa fonte na árvore partilhada para evitar que o nó receba pacotes da mesma fonte em classes distintas.

- **Método handle-SetSPTBit**

  Este método é invocado pelo classificador do MCMRP quando pretende fazer o *prune* de uma fonte na árvore partilhada.

  Como já foi referido, quando um determinado agente quer comutar da árvore partilhada para uma árvore centrada numa fonte, é criada pelo método **recv-ack** uma entrada do tipo (S,G,i) com o SPT-Bit a 0. Este bit mantém-se a zero temporariamente até que o tráfego da fonte S começa a fluir através da árvore centrada na fonte. Quando isso acontece o bit é posto a 1 para tornar efectiva a entrada (S,G,i). Essa alteração do SPT-Bit de 0 para 1 é feita pelo classificador do MCMRP, assim que recebe o primeiro pacote marcado na classe i, vindo da fonte S pela interface esperada. Depois é necessário fazer um *prune* daquela fonte na árvore partilhada para evitar que o tráfego proveniente daquela fonte chegue ao nó também através da árvore partilhada. Esse *prune* é diferente dos outros porque tem o RPT-Bit a 1, apesar de ter o endereço de uma fonte na lista de *prune*, e o WC-Bit a 0. Este pedido deve ser re-enviado em direcção ao RP e não em direcção à fonte. Este *prune* só deverá ser enviado se a árvore pertencer à árvore partilhada daquela mesma classe.

- **Método leave-group**

  Este método deverá ser invocado quando um determinado agente associado a um nó decide abandonar um grupo, e invoca não só o método **leave-rpt** para cada uma das classes suportadas, como também o método **leave-spt** tantas vezes quantas as necessárias para ser removido de todas as árvores centradas nas fontes para as quais eventualmente tenha comutado.

- **Método leave-rpt**

  Para remover o agente duma árvore partilhada na classe de serviço i é necessário apagá-lo da lista de interfaces da saída da entrada (*,G,i). Isto é feito pelo método **leave-group** da classe **Node**.

  Se a lista de interfaces de saída da entrada (*,G,i) ficar vazia, a entrada poderá ser apagada e deverá ser gerada uma mensagem de *prune-request*. Um *prune* deste tipo tem o RPT-Bit e o WC-Bit ambos a 1. Alem disso o endereço que aparece na lista de *prunes* é o endereço do RP do grupo. No caso do PIM-SM esta mensagem de *prune-request* seria enviada para o próximo nó no caminho mais curto para o RP. No entanto, no MCMRP, à semelhança aliás do que acontece no protocolo DTMP, este mecanismo de *prune* teve que ser alterado pelo facto destes protocolos construírem árvores directas e não invertidas. Assim, a mensagem de *prune-request* é enviada em direcção ao vizinho imediatamente acima na árvore de distribuição *multicast*. A

identificação deste nó consta das próprias entradas da tabela de encaminhamento *multicast*, neste caso em concreto o campo *neighbor* da entrada (*,G,i).

- **Método leave-spt**

  Este método é invocado pelo método **leave-group** para retirar determinado agente de todas as árvores centradas na fontes para as quais eventualmente tenha comutado.

  Para remover um agente de uma árvore centrada numa determinada fonte S é necessário apagá-lo da lista de interfaces da saída da entrada (S,G,i). O agente deverá ser apagada quer a entrada tenha o RPT a 1 ou a 0. Isto é feito pelo método **leave-group** da classe **Node**.

  Se a entrada (S,G,i) tiver o RPT-Bit a 1, a entrada (S,G,i) não deverá ser apagada, mesmo que a lista de interfaces de saída fique vazia. Se a lista de interfaces de saída ficar vazia dever-se-á enviar um *prune* da fonte S na árvore partilhada. Este *prune* é semelhante ao enviado no método **handle-SetSPTBit**, ou seja, apesar de ter o endereço de uma fonte na lista de *prune* deve ser propagado ao longo da árvore partilhada da respectiva classe. Para que isso aconteça o RPT-Bit deverá ser posto a 1 e o WC-Bit a 0.

  Se a entrada (S,G,i) tiver o RPT-Bit a 0 e a lista de interfaces de saída ficar vazia, a entrada poderá ser apagada e o *prune* deverá ser re-enviado ao longo da árvore centrada na fonte ou seja, a mensagem de *prune-request* é enviada para o vizinho imediatamente acima na árvore de centrada na fonte (o campo *neighbor* da entrada (S,G,i)). Um *prune* deste tipo tem o RPT-Bit e o WC-Bit ambos a 0 e o endereço que aparece na lista de *prunes* é o endereço da fonte.

- **Método recv-prune**

  Este método é invocado sempre que um nó recebe uma mensagem de *prune-request*. Existem basicamente três *prunes* possíveis: o *prune* de um nó numa árvore partilhada, o *prune* de um nó numa árvore centrada na fonte e o *prune* de uma fonte na árvore partilhada. Para os distinguir é necessário testar o WC-Bit e o RPT-Bit.

  - Recepção de um prune de um nó na árvore partilhada

    Um *prune* deste tipo tem o RPT-Bit e o WC-Bit ambos a 1. Além disso o endereço que aparece na lista de *prunes* é o endereço do RP do grupo. Ao receber um *prune* deste tipo é necessário verificar antes de mais se existe alguma entrada (*,G,i). Se existir é necessário apagar actualizar a lista de interfaces de saída da entrada (*,G,i), apagando uma interface de forma a "podar" o ramo respectivo. No caso do PIM-SM a interface a apagar é a interface por onde chegou o *prune*. No caso do MCMRP, como as árvores de distribuição de tráfego são construídas de forma directa, a interface a apagar da lista de interfaces de saída deverá ser a interface usada pelo encaminhador para atingir o nó que enviou o *prune*. No caso de estarmos perante uma rede assimétrica, essa interface não tem de ser a mesma por onde chegou a mensagem de *prune-request*.

Se, depois de apagar esta interface, a lista de interfaces de saída ficar vazia, a entrada deverá ser apagada e a mensagem de *prune-request* deverá ser re-enviado ao longo da árvore partilhada, ou seja, deverá ser re-enviada para o nó que está imediatamente acima dele na árvore partilhada (campo *neighbor* da entrada (*,G,i)).

Depois é necessário verificar se existem entradas (S,G,i) onde consta a interface apagada da entrada (*,G,i). Se existirem, esta interface também deve ser apagada da lista de interfaces de saída das entradas (S,G,i). Se ao apagar a interface, a lista de interfaces de saída ficar vazia temos duas situações possíveis:

* Se a entrada (S,G,i) tiver o RPT-Bit a 0 então é necessário apagar a entrada e propagar o *prune* ao longo da árvore centrada na fonte S (ou seja, re-enviar a mensagem de *prune-request* para o vizinho identificado através do campo *neighbor* da entrada (S,G)), tendo o cuidado de colocar o WC-Bit e o RPT-Bit, ambos a zero.
* Se a entrada (S,G,i) tiver o RPT-Bit a 1 então não se pode apagar a entrada (S,G,i), mas é necessário propagar o *prune*, desta vez ao longo da árvore partilhada (ou seja, enviar a mensagem de *prune-request* para o vizinho identificado através do campo *neighbor* da entrada (*,G,i)), tendo o cuidado de colocar o RPT-Bit a 1 e o WC-Bit a 0.

– Recepção de um prune de um nó na árvore centrada na fonte

Um *prune* deste tipo tem o RPT-Bit e o WC-Bit ambos a 0. O endereço que aparece na lista de *prunes* é o endereço da fonte. Ao receber um *prune* deste tipo deve-se verificar se existe alguma entrada (S,G,i). Se existir a interface usada para atingir o nó que enviou o *prune* deve ser apagada da lista de interfaces de saída da entrada (S,G,i). Se existir uma entrada (*,G,i) e a entrada (S,G,i) tiver o RPT-Bit a 1, a entrada (S,G,i) não deverá ser apagada, mesmo que a lista de interfaces de saída ficar vazia, Caso contrário, se a lista de interfaces de saída ficar vazia, a entrada poderá ser apagada e o *prune* deverá ser re-enviado ao longo da árvore centrada na fonte S (ou seja, enviar para o vizinho identificado através do campo *neighbor* da entrada (S,G,i)), tendo o cuidado de manter o WC-Bit e o RPT-Bit, ambos a zero.

– Recepção de um prune de uma fonte na árvore partilhada

Um *prune* deste tipo tem o RPT-Bit a 1 e o WC-Bit a 0. O endereço que aparece na lista de *prunes* é o endereço da fonte da qual não se quer receber mais pacotes através da árvore partilhada. Ao receber um *prune* deste tipo deve-se verificar se existe alguma entrada (S,G,i) Se existir, a interface usada para atingir o nó que enviou o *prune* deve ser apagada da lista de interfaces de saída da entrada (S,G,i). Se existir uma entrada (*,G,i) e a entrada (S,G,i) tiver o RPT-Bit a 1, a entrada (S,G,i) não deverá ser apagada, mesmo que a lista de interfaces de saída ficar vazia, Caso contrário, se a lista de interfaces de saída ficar vazia, a entrada poderá ser apagada e o *prune* deverá ser re-enviado para o vizinho identificado através do campo *neighbor* da entrada (S,G,i).

Se existir uma entrada (*,G,i), mas não existir uma entrada (S,G,i), tem que ser criada uma entrada (S,G,i) com o RPT-Bit a 1. A lista de interfaces de saída é copiada da lista de interfaces de saída da entrada (*,G,i), apagando-se desta lista a interface usada para atingir o nó que enviou o *prune*.

7.3.2 Implementação de um classificador MCMRP

O classificador MCMRP é basicamente igual ao classificador DTMP, com apenas uma pequena diferença: a integração da *flag* CST-Bit que permite caracterizar o tipo de *join* requerido.

7.4 Teste da Implementação

Para implementar e avaliar o MCMRP[12] usou-se também o Network Simulator, versão 2.18b. Os resultados obtidos foram comparados com resultados obtidos através da utilização do PIM-SM. Como já foi referido o MCMRP recorre ao CoSLSP para estabelecer os ramos das árvores de distribuição *multicast*. No caso do PIM-SM utilizou-se o LS como protocolo *unicast* de suporte.

Começou-se por medir a qualidade das árvores construídas pelo MCMRP, usando diferentes métricas:

- O número de ligações que constituem as árvores de distribuição *multicast*;
- O número de réplicas de pacotes que são originadas pelos diferentes nós quando re-enviam os pacotes *multicast* ao longo da árvore de distribuição de tráfego;
- O tamanho médio dos caminhos fim-a-fim, ou seja dos caminhos que separam cada uma das fontes dos diferentes destinatários;

Nenhuma destas métricas tem em consideração as características das ligações, nomeadamente o seu custo, e não podem ser usadas para determinar como é que o processo de construção das árvores lida com as assimetrias das ligações. Para colmatar essa lacuna usou-se também uma métrica que combina o número de réplicas com o custo associado a cada ligação que cada réplica atravessa.

Além da qualidade das árvores interessa medir neste caso as perdas ocorridas em cada uma das classes de serviço consideradas, para assim se medir o ganho da utilização do encaminhamento por classes de serviço. As métricas consideradas neste caso foram a média de perdas ocorrida por fluxo de tráfego nas diferentes classes de serviço, e a média de perdas sofridas por cada um dos destinatários.

7.4.1 Cenários de Simulação

Para testar o MCMRP com o CoSLSP e compará-lo com o PIM-SM quando usado com o protocolo LS usou-se a mesma topologia que se usou para testar o CoSLSP embora com um cenário diferente, ou seja a topologia típica de um grande ISP com 36 nós.

7.4.1.1 Topologia de rede típica de um ISP

A primeira topologia de rede usada é uma rede típica de um grande ISP[65] e está ilustrada na figura 7.5.

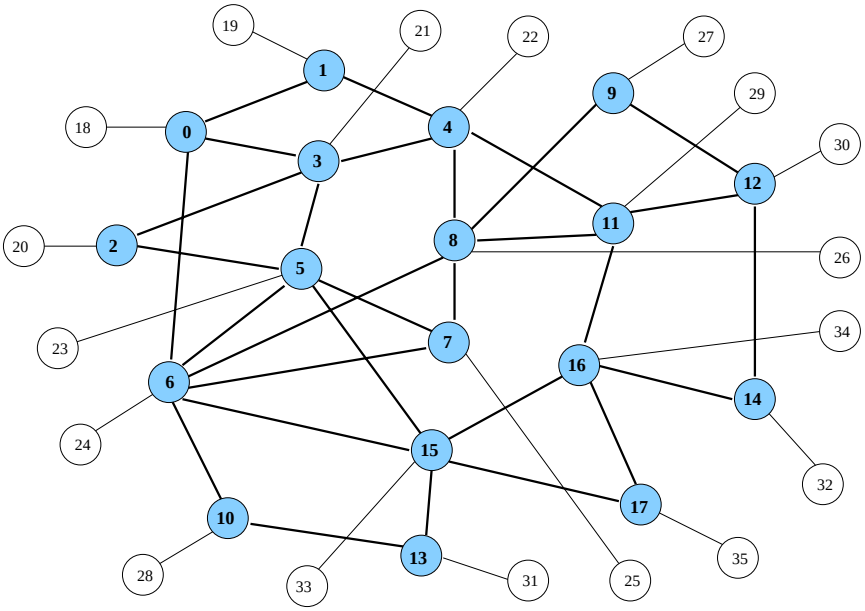


Figura 7.5: Topologia de rede típica de um ISP

Esta topologia inclui 36 nós: 18 são encaminhadores internos e os outros 18 são encaminhadores de fronteira. Cada um dos encaminhadores internos está ligado a um encaminhador de fronteira através de uma ligação simétrica de custo 1. Os nós internos estão ligados entre si através de 30 ligações assimétricas de forma a existirem diferentes caminhos alternativos com diferentes custos entre quaisquer pares de nós. Neste caso os custos das ligações são inteiros escolhidos aleatoriamente no intervalo [1, 10].

Foram considerados três classes de serviço distintas com diferentes requisitos de QoS expressos em termos da percentagem de perdas que conseguem suportar. A classe 1, a mais exigente, consegue suportar bem até 5% de perdas, a classe 2 suporta bem até 25% de perdas, enquanto a classe 3, a menos exigente, consegue suportar até 50% de perdas. Cada ligação tem 3 filas físicas (uma por classe), tendo cada uma delas duas filas virtuais que correspondem a dois níveis diferentes de precedência em função das perdas ocorridas

nas filas. Todas as filas foram configuradas exactamente da mesma forma de forma a evitar a diferenciação no interior dos nós.

Para cada simulação foi considerado um único grupo e o RP é escolhido aleatoriamente entre todos os nós da topologia. No início de cada simulação não existe nenhum membro no grupo, mas existem potenciais fontes e potenciais receptores associados a todos os encaminhadores de fronteira.

Neste primeiro cenário existem 9 fontes fixas e cada uma das 9 fontes gera tráfego marcado numa das três classes de serviço, gerada aleatoriamente. As fontes começam a emitir em direcção ao RP, uma de cada vez, a uma taxa constante de 250 kbs, o que acaba por depressa provocar a congestão de algumas ligações, uma vez que todas todas elas têm apenas a capacidade de 1.5 Mbs. Depois de todas as fontes estarem a emitir, 9 receptores escolhidos aleatoriamente começam a juntar-se ao grupo, também um de cada vez. Depois de todos os 9 nós se terem juntado ao grupo, 9 deles, escolhidos aleatoriamente, juntam-se às 9 fontes numa classe de serviço também aleatória. Este cenário (1 árvore partilhada e 9 árvores centradas nas fontes] é depois mantido até ao fim da simulação. Antes da simulação terminar os 9 membros do grupo abandonam-o e as fontes param de emitir.

Para cada experiência foram executadas 100 simulações independentes para cada protocolo e os resultados obtidos representam a média dessas 100 simulações.

7.4.2 Análise dos resultados

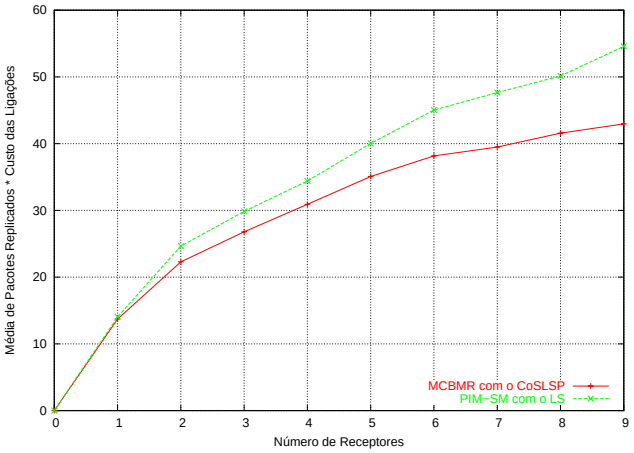
Os resultados obtidos através simulações efectuadas são apresentados figuras 7.6, 7.7 e 7.8.

As figuras 7.6, mostram as características das árvores construídas pelos dois protocolos (PIM-SM e MCMRP) quando é utilizado a primeiro topologia (topologia típica de um grande ISP). As curvas apresentadas na figura 7.6(a) mostram o custo das árvores em função do número de receptores. Este é um custo pesado já que é medido em função do número de réplicas *vezes* o custo das ligações atravessadas. As curvas apresentadas na figura 7.6(b) apresentam o número total de ligações da topologia que fazem parte das árvores de distribuição em função do número de membros do grupo.

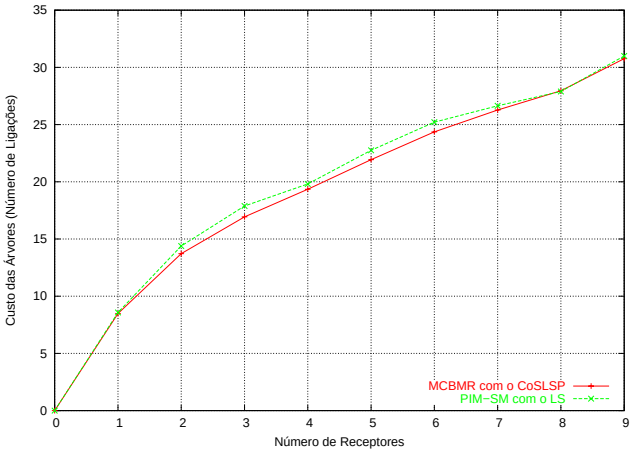
Estes resultados mostram claramente que o MCMRP constrói árvores de menor custo do que as construídas pelo protocolo PIM-SM. Isto deve-se ao facto do MCMRP construir árvores directas e era sem dúvida um resultado esperado, agora comprovado. É de realçar no entanto que o CoSLSP não escolhe as melhores rotas, mas sim as melhores rotas que adicionalmente respeitam os requisitos de QoS estabelecidos para cada classe de serviço. Obtém-se assim uma dupla vantagem: o MCMRP utilizando o CoSLSP constrói melhores árvores do que o PIM-SM com o LS, e estas árvores são adicionalmente condicionadas pelos requisitos de QoS.

Na figura 7.6(b) pode-se observar que não existem diferenças significativas no número de ligações envolvidas nas árvores de distribuição construídas pelos dois protocolos.

A figura 7.7, mostra as características dos caminhos que separam as raízes das dife-



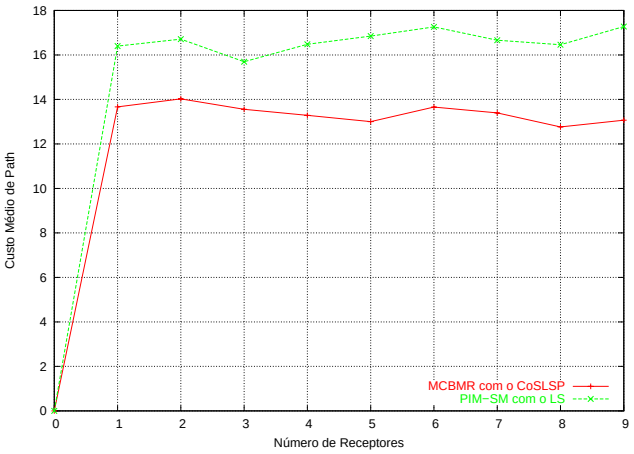
(a) *Custo das árvores (número de réplicas* `vezes` *o custo das ligações)*



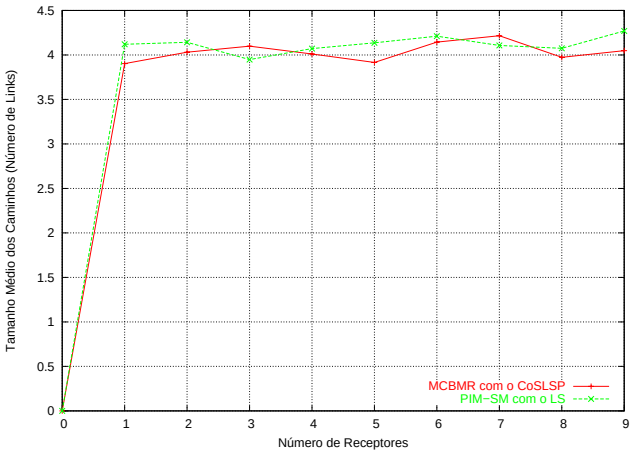
(b) *Número de ligações que fazem parte das árvores de distribuição*

Figura 7.6: Qualidade das árvores com o primeiro cenário de simulação

rentes árvores das respectivas folhas (membros do grupo), quer em termos de tamanho, quer em termos de custo.



(a) *Custo das médio dos caminhos fim a fim)*



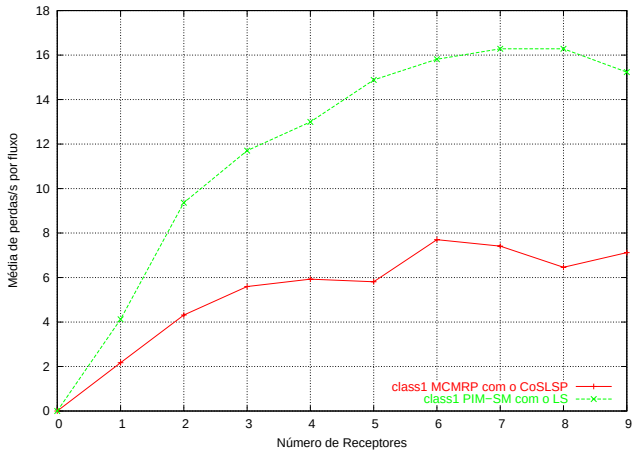
(b) *Número de ligações médio dos caminhos fim a fim*

Figura 7.7: Qualidade das árvores com o primeiro cenário de simulação

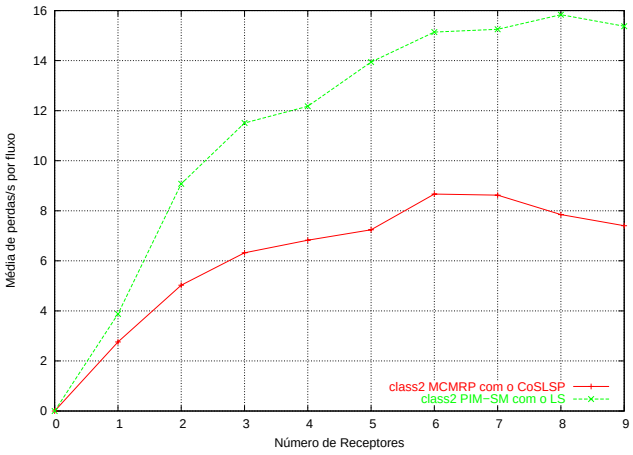
Observa-se exactamente a mesma tendência, ou seja, o custo médio dos caminhos fim a fim quando se utiliza o protocolo DTMP é significativamente menor do que o custo médio dos caminhos fim a fim quando se utiliza o PIM-SM. O mesmo não se verifica no

tamanho médio dos caminhos, mas cuja diferença não é significativa.

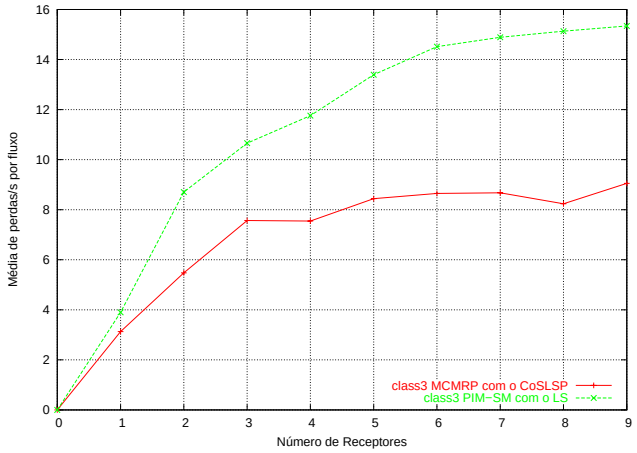
A figura 7.8 mostra a média de pacotes perdido por fluxo em função do número de receptores, quando se utiliza o MCMRP e o PIM-SM. As curvas que constam das figuras 7.8(a), 7.8(b) e 7.8(c) representam as perdas ocorridas nos fluxos de cada classe de serviço. A figura 7.8(d) mostra as perdas ocorridas nos fluxos de todas as classes quando se utiliza o protocolo MCMRP.



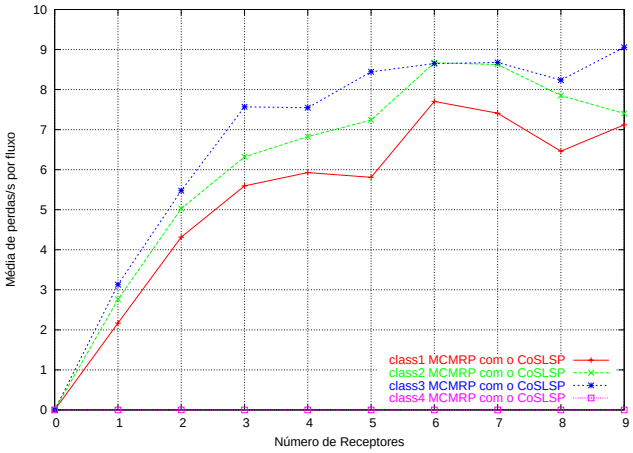
(a) *Perdas por fluxo - Classe 1*



(b) *Perdas por fluxo - Classe 2*



(c) *Perdas por fluxo - Classe 3*



(d) *Perdas por fluxo no MCMRP - Todas as classes*

Figura 7.8: Média de perdas por fluxo obtidas usando o primeiro cenário de simulação

Estes resultados demonstram que quando o protocolo MCMRP é utilizado são verificadas significativamente menos perdas. Isto deve-se ao facto do MCMRP procurar

encontrar rotas menos congestionadas quando as ligações começam a sofrer perdas. Por outro lado podemos observar através da figura 7.8(d) que as perdas sofridas quando se utiliza o protocolo MCMRP são menores para a classe 1 do que para as classes 2 e 3 e maiores para a classe 2 do que para a classe 3. Isto deve-se ao facto da classe 1 ter requisitos mais exigentes em termos de perdas do que a classe 2, e a classe 2 ter requisitos mais exigentes em termos de perdas do que a classe 3.

7.5 Conclusões

O MCMRP é um protocolo de encaminhamento *multicast* que implementa o encaminhamento *multicast* por classes de serviço, e por isso se adequa à utilização em ambientes DiffServ.

Sendo estes ambientes inerentemente unidireccionais o MCMRP propõe a utilização de árvores directas, partilhadas ou centradas nas fontes, em vez da típicas árvores invertidas utilizadas pela maior parte dos protocolos de encaminhamento *multicast* conhecidos. A heurística usada pelo protocolo MCMRP baseia-se nas respostas enviadas pelas fontes ou pelos encaminhadores RP aos pedidos explícitos de junção de candidatos a novos membros.

O MCMRP foi implementado e testado usando o Network Simulator. Os resultados obtidos mostram que na presença de ligações assimétricas o MCMRP é uma estratégia prometedora, permitindo o estabelecimento de árvores de distribuição de tráfego *multicast* directas com menores custos do que as árvores partilhadas e centradas na fontes criadas pelo protocolo PIM-SM. Adicionalmente o MCMRP é capaz de encontrar "melhores" árvores no respeita à métrica de QoS associada a cada classe de serviço.

Capítulo 8

Conclusões e Trabalho Futuro

Este capítulo apresenta as conclusões gerais do trabalho efectuado. Começa-se por fazer uma síntese das diferentes contribuições, realçando a motivação que levou à concepção e desenvolvimento dos diferentes protocolos aqui propostos, após o que se procede a uma análise crítica dos diferentes resultados obtidos. Por fim são apresentadas algumas considerações relativas ao trabalho futuro.

8.1 Síntese e Contribuições

A principal motivação do trabalho realizado prende-se com questões essencialmente relacionadas com a necessidade de adaptar os protocolos de encaminhamento (*unicast* e *multicast*) aos ambientes caracterizados por serviços de rede diferenciados, nomeadamente as redes DiffServ[3].

Como foi referido, não existe uma solução genérica adoptada universalmente para o problema do encaminhamento com QoS, embora seja comummente aceite que, perante tráfego com requisitos de QoS, os protocolos de encaminhamento deverão determinar as rotas, tomando de alguma forma, em consideração esses requisitos. Nesse caso, ou seja, quando um protocolo de encaminhamento determina rotas tendo tem em consideração os requisitos de QoS do tráfego, diz-se que se trata de um protocolo de encaminhamento com QoS.

As várias propostas que têm surgido ao longo dos últimos anos incluem diferentes estratégias para resolver o problema do encaminhamento com QoS, embora quase todas elas assentem nos mesmo princípios. A maior parte propõe que o cálculo dos caminhos seja feito por iniciativa da fonte antes de começar a transmitir, ou seja, a pedido. O encaminhador mais próximo da fonte reage, tipicamente, enviando para todos os seus vizinhos, mensagens de *probing*, endereçadas ao destinatário da conexão. Estas mensagens transportam não só os requisitos de QoS da nova conexão, mas também o QoS acumulado pela própria mensagem ao longo do seu percurso. Os encaminhadores, ao receberem

uma mensagem de *probing*, só a re-enviam através das ligações cujo QoS acumulado não exceda os requisitos de QoS da conexão. Desta forma, quando uma mensagem de *probing* chega ao destinatário consegue-se ter a certeza que o percurso por ela efectuado possui os recursos necessários à satisfação dos requisitos de QoS pretendidos. Basta então enviar uma mensagem de volta pelo mesmo caminho, mensagem essa que finalmente permite estabelecer o caminho definitivo.

Como é fácil de deduzir, este tipo de estratégia implica a instalação e manutenção de informação por fluxo nos diferentes encaminhadores envolvidos nesse caminho e genericamente, em toda a rede. Como principal vantagem aponta-se o facto de conseguir garantir que os requisitos de QoS são efectivamente satisfeitos desde que, depois de estabelecido o caminho definitivo, se efectue a respectiva reserva de recursos. Mas, por outro lado, esta solução apresenta alguns problemas graves, nomeadamente de escala, ao implicar que todos os encaminhadores mantenham informação de estado por cada fluxo activo. Além disso, e também por esta razão, o seu modo de funcionamento não é adequado à utilização em ambientes do tipo DiffServ.

O modelo de DiffServ resolve os problemas de escalabilidade existentes no modelo IntServ[2] eliminando precisamente a necessidade de manutenção de informação de estado por fluxo. No modelo DiffServ a qualidade de serviço deixa de ser garantida mediante os requisitos de um fluxo e apenas existe um tratamento diferenciado por classe de serviço, sendo obviamente o número de diferentes classes muito inferior ao número de fluxos individuais. Os pacotes dos diferentes fluxos são marcados distintamente de forma a criar diferentes classes de tráfego. Os pacotes recebem um tipo de tratamento consoante a classe de tráfego a que pertencem.

A utilização da estratégia de encaminhamento por fluxo, a pedido, não é escalável nem parece assim adequada ao contexto de redes DiffServ. Em vez disso deverão ser determinadas diferentes rotas adequadas a cada classe de serviço. Em vez do cálculo a pedido, deverá ter lugar um cálculo prévio dos caminhos, de forma a que seja possível instalá-los nas tabelas de encaminhamento antes de ser necessário começar a processar pacotes. Desta forma, quando um pacote chega ao encaminhador, este deve ser capaz de, consultando a tabela de encaminhamento, determinar qual o próximo salto no caminho mais adequado à classe de serviço a que o pacote pertence. Como é óbvio estes caminhos por classe de serviço terão que ser actualizados de acordo com alguma dinâmica do estado da rede.

À luz desta motivação surgem então, e como principais objectivos deste trabalho, estudar, propor, realizar e testar soluções que permitam implementar o encaminhamento por classes de serviço dentro de um domínio administrativo, isto é, soluções de encaminhamento interno. O ênfase é dado ao encaminhamento *multicast*, embora também tenha sido necessário conceber e desenvolver uma estratégia de encaminhamento *unicast* por classes de serviço.

Assim sendo o trabalho desenvolvido pode ser agrupado ou sintetizado nas seguintes fases:

- Estudo e implementação do protocolo PIM-SM[7] no Network Simulator[6]. De entre todos os protocolos de encaminhamento *multicast* existentes o PIM-SM é umas das propostas mais promissoras. Pareceu assim óbvio começar por aqui. A implementação do protocolo no Network Simulator teve não só como objectivo a familiarização com o ambiente experimental que seria utilizado ao longo de todo o trabalho, mas também disponibilizar a implementação de um protocolo de encaminhamento *multicast* que poderia servir de referencial para comparação com os protocolos de encaminhamento *multicast* propostos e desenvolvidos ao longo deste trabalho.

- Tanto no protocolo PIM-SM, como na maioria dos protocolos de encaminhamento *multicast* tradicionais é usado o conceito de *reverse path routing* na construção das árvores de distribuição de tráfego *multicast*. Segundo essa estratégia as árvores são construídas a partir do candidato a novo membro e até ao primeiro nó na árvore, ou seja, exactamente no sentido contrário ao sentido usado pelo tráfego. Este facto torna este tipo de protocolos desadequado à utilização em redes assimétricas, como é o caso das redes que suportam QoS. Neste caso, o que faz mais sentido é construir as árvores no sentido directo do tráfego, pois só nesse sentido é que é possível avaliar convenientemente as métricas de QoS nomeadamente a largura de banda disponível, atrasos fim a fim e percentagem de perdas. Por esse motivo o primeiro protocolo concebido, implementado e avaliado no contexto deste trabalho é um protocolo de encaminhamento *multicast* que constrói as árvores de acordo com o sentido que é usado pelo tráfego: o DTMP (Directed Trees Multicast Protocol).

- Os protocolos de encaminhamento *multicast* constroem as árvores de distribuição de tráfego com base nos "melhores" caminhos descobertos pelos protocolos de encaminhamento *unicast*. Para ser possível obter diferentes árvores de distribuição de tráfego *multicast*, dependendo dos requisitos das diferentes classes de serviço, é necessário ter subjacente uma estratégia de encaminhamento *unicast* por classes de serviço. Por esse motivo, depois de avaliado o protocolo DTMP, e antes ainda de se iniciar a concepção e desenvolvimento do protocolo de encaminhamento *multicast* por classes de serviço, partiu-se para o desenvolvimento de uma estratégia *unicast* por classes de serviço, estratégia essa que deu origem a um novo protocolo: o CoSLSP (Class-Of-Service Link State Protocol). Este novo protocolo é um protocolo de estado de ligação que determina uma rota por cada classe de serviço.

- Com base no protocolo DTMP desenvolvido e utilizando o protocolo CoSLSP como protocolo de encaminhamento *unicast* subjacente propôs-se e desenvolveu-se um protocolo de encaminhamento *multicast* por classes de serviço. A esse protocolo deu-se a designação de MCMRP (Multi-Class based Multicast Routing Protocol).

Como contribuições desta tese realçam-se assim a concepção e desenvolvimento dos seguintes protocolos:

- o protocolo DTMP[8] - Directed Trees Multicast Protocol. Trata-se de um protocolo de encaminhamento *multicast* que suporta árvores partilhadas e árvores centradas na

fontes. A estratégia de construção das árvores permite construí-las no sentido real usado pelo tráfego, e por isso designam-se por árvores directas. Com a construção de árvores directas pretende-se tornar o protocolo mais eficiente na presença de redes assimétricas.

- o protocolo CoSLSP[9] - Class-of-Service Link State Protocol. Trata-se de um protocolo de encaminhamento *unicast* com uma estratégia de descoberta de caminhos por classes de serviço. Mediante os requisitos das diferentes classes de serviço consideradas, este protocolo determina, usando um algoritmo de estado de ligação, para cada destino possível o caminho mais curto que consegue satisfazer os requisitos de cada uma das diferentes classes de serviço consideradas.

- o protocolo MCMRP[11][12]- Muti-Class based Multicast Routing Protocol. Trata-se de um protocolo de encaminhamento *multicast* por classes de serviço. O MCMRP baseia o seu funcionamento na construção de diferentes árvores directas, quer partilhadas, quer centradas nas fontes. Concretamente prevê que numa fase inicial os diferentes candidatos a novos membros de um grupo, se juntem ao grupo através das árvores partilhadas. Existem tantas árvores partilhadas quantas as classes de serviço consideradas, de forma a que qualquer fonte possa começar a transmitir em qualquer classe de serviço atingindo todos os membros activos do grupo. As diferentes árvores são construídas usando a informação disponibilizada pelo protocolo CoSLSP de forma a que cada uma delas satisfaça os requisitos de QoS da respectiva classe de serviço. Passado esse período inicial, um determinado membro do grupo pode, se assim o desejar, juntar-se a uma determinada fonte requerendo uma determinada classe de serviço. Se a fonte aceitar o pedido, é construída uma árvore, agora centrada na fonte, capaz de satisfazer os requisitos de QoS da classe de serviço requerida.

8.2 Principais conclusões do trabalho desenvolvido

Antes de mais convém aqui assinalar que este trabalho não é o resultado de uma iniciativa isolada. Está sim integrado num conjunto de outros trabalhos que a seguir se referem.

A implementação de uma rede com serviços diferenciados envolve a construção de algoritmos para agregar os diferentes níveis de serviço, efectuar a marcação de pacotes de acordo com os requisitos de qualidade de serviço dos fluxos de tráfego a que pertencem e tratar de forma diferenciada estes pacotes mediante a marcação efectuada. Este tratamento diferenciado é habitualmente implementado nó a nó através de diferentes filas de espera utilizando diferentes tipos de algoritmos de escalonamento[77][78]. Paralelamente é necessário efectuar a monitorização do estado da rede e implementar mecanismos de controle de admissão[79][80].

A principal hipótese aqui estabelecida, e que serviu de base ao desenvolvimento deste trabalho, é a de que o estabelecimento de rotas por classes de serviço pode complementar,

ao nível do encaminhamento, as técnicas de diferenciação estabelecidas ao nível dos nós. Complementarmente, para além das soluções intra-domínio, é também necessário algum tipo de tratamento diferenciado nas ligações entre diferentes domínio administrativos[81].

Para simplificar a exposição das conclusões gerais do trabalho efectuado optou-se por agrupá-las segundo os diferentes protocolos concebidos e desenvolvidos.

8.2.1 Protocolo DTMP

O protocolo DTMP[8] foi o primeiro protocolo desenvolvido. Tem como principal objectivo a construção de árvores de distribuição directas em vez das habituais árvores invertidas. O protocolo foi inspirado no protocolo PIM-SM suportando, à semelhança deste, tanto árvores partilhadas como árvores centradas nas fontes.

O protocolo proposto foi implementado e avaliado usando o Network Simulator. Foram efectuados diversos testes com diferentes topologias no sentido de demonstrar que na presença de redes assimétricas o protocolo DTMP consegue construir árvores de menor custo que o protocolo PIM-SM sem introduzir, por este facto, demasiada sobrecarga na rede.

A concepção e implementação deste protocolo constitui um primeiro passo no sentido da construção de um protocolo de encaminhamento *multicast* com QoS. Para implementar encaminhamento com QoS, a estratégia das árvores invertidas não é adequada, pois ao considerarmos métricas de QoS, como a largura de banda disponível, a probabilidade de ocorrência de perdas ou os atrasos fim a fim estamos obrigatoriamente a transformar a topologia, mesmo que tenha todas as ligações com um custo unitário, numa topologia assimétrica, uma vez que as métricas têm que ser avaliadas (medidas) no sentido real assumido pelo tráfego.

Outra característica importante do DTMP é o facto de ser independente do protocolo *unicast* subjacente, o que poderá facilitar a integração do DTMP com a base instalada, tornando-se fácil usá-lo como uma extensão do PIM-SM no sentido de suportar árvores directas.

8.2.2 Protocolo CoSLSP

O CoSLSP[9] é um protocolo de encaminhamento *unicast* por classes de serviço adequado à utilização em ambientes do tipo DiffServ. A ideia subjacente aos protocolos de encaminhamento por classes de serviço consiste em procurar uma rota por cada classe de serviço existente, rota essa que poderá conseguir satisfazer o requisitos da classe de serviço respectiva. No caso do CoSLSP, a estratégia adoptada foi a de usar um protocolo de estado de ligação como suporte. Este protocolo é responsável pela difusão de informação do estado das ligações por todo o domínio DiffServ, de forma a que cada encaminhador consiga determinar a "melhor" rota para cada uma das classes de serviço consideradas.

O CoSLSP foi implementado e avaliado com o Network Simulator. Os resultados obtidos foram comparados com o protocolo LS, uma implementação de um protocolo semelhante ao protocolo OSPF, disponível na versão do Network Simulator utilizada. Os testes efectuadas mostraram que o CoSLSP nos casos em que se verifica congestão na rede, é capaz de encontrar "melhores" rotas em termos da métrica de QoS considerada: taxa de perdas de pacotes.

Além disso o protocolo CoSLSP procura optimizar a utilização de recursos de rede. Este objectivo é conseguido através do algoritmo para o cálculo dos caminhos que procura sempre em primeiro lugar o caminho mais curto. Se este não satisfizer os requisitos de QoS especificados, o segundo caminho mais curto é calculado e assim sucessivamente até que um caminho exequível seja encontrado ou seja atingido um limite imposto por configuração.

Esta estratégia para calcular os caminhos, além de contribuir de forma significativa para optimizar a utilização dos recursos da rede, consegue também melhorar o comportamento do protocolo ao nível da sua estabilidade. Os protocolos de encaminhamento com QoS tem uma importante desvantagem em relação aos protocolo de encaminhamento tradicionais, que é facto da métrica ou métricas que procuram optimizar (tipicamente o atraso fim a fim, ou percentagem de perdas) mudarem muito mais frequentemente do que o tradicional custo das ligações (número de saltos ou um custo imposto de forma administrativa). Em consequência disto os caminhos óptimos mudam também muito mais frequentemente, a menos que sejam usados mecanismos adicionais para prevenir as oscilações frequentes no encaminhamento. Com o CoSLSP este problema é minimizado, uma vez que este protocolo procura optimizar o habitual custo das ligações. Este custo só sofre alterações se uma ligação ou um nó falha, ou quando é efectuada alguma alteração nas configurações dos equipamentos. O CoSLSP só desencadeia o processo de alteração no encaminhamento se os caminhos previamente calculados deixam de satisfazer os requisitos de QoS das classes de serviço correspondentes ou, se por outro lado, aparecem caminhos mais curtos que passam a satisfazê-los. Apesar disso o CoSLSP procura minimizar a sobrecarga introduzida pelo protocolo devido às actualizações frequentes desencadeando as actualizações da informação de estado apenas quando é detectada uma alteração significativa no valor da métrica, em relação ao último valor anunciado.

8.2.3 Protocolo MCMRP

O MCMRP[11][12] é um protocolo de encaminhamento *multicast* que implementa o encaminhamento *multicast* por classes de serviço, e por isso se adequa à utilização em ambientes DiffServ.

Sendo estes ambientes inerentemente unidireccionais o MCMRP propõe a utilização de árvores directas, partilhadas ou centradas nas fontes, em vez da típicas árvores invertidas utilizadas pela maior parte dos protocolos de encaminhamento *multicast* conhecidos. A heurística usada pelo protocolo MCMRP baseia-se nas respostas enviadas pelas fontes ou pelos encaminhadores RP aos pedidos explícitos de junção de candidatos a novos membros.

O MCMRP foi implementado e testado usando o Network Simulator. Os resultados obtidos mostram que na presença de redes assimétricas o MCMRP é uma estratégia prometedora, permitindo o estabelecimento de árvores de distribuição de tráfego *multicast* directas com menores custos do que as árvores partilhadas e centradas na fontes criadas pelo protocolo PIM-SM. Adicionalmente o MCMRP é capaz de encontrar "melhores" árvores no respeita à métrica de QoS associada a cada classe de serviço.

Além disso o MCMRP respeita o modelo hoje em uso na Internet para a implementação de comunicações *multicast* no sentido em que: não exige qualquer conhecimento prévio da acerca da constituição do grupo *multicast*, fontes e candidatos a novos membros de um grupo necessitam apenas de saber qual é o RP de determinado grupo; permite que qualquer fonte comece a transmitir a qualquer momento para um determinado grupo; e por fim, permite que os diferentes membros de um grupo *multicast* se juntem a ele ou o abandonem em qualquer altura.

Foi portanto proposto um novo ambiente de trabalho para encaminhamento *multicast* com QoS, tendo sido também analisadas e avaliadas as características dos novos protocolos desenvolvidos.

8.3 Trabalho Futuro

Neste trabalho, nomeadamente na implementação dos protocolos CoSLSP e MCMRP, optou-se por usar como métrica de QoS a percentagem de perdas ocorrida em cada ligação. Trata-se de um métrica multiplicava, ou seja, a regra que compõe a taxa de perdas acumulada é dada pela seguinte fórmula: $Tp_{(c)} = 1-((1-Tp_{(i,j)})*(1-Tp_{(j,k)})*...*(1-Tp_{(j,k)}))$, sendo c um determinado caminho, $Tp_{(c)}$ a taxa de perdas verificada no caminho c, e $Tp_{(i,j)}$ a taxa de perdas verificada na ligação que une os nós i e j.

Os requisitos das diferentes classes de serviço consideradas terão que ser expressos naturalmente em função desta métrica. Através do protocolo CoSLSP os custos das diferentes ligações são divulgados para os outros encaminhadores do domínio, juntamente com as percentagens de perdas ocorridas nas diferentes ligações. Estes constroem com esses dados a base de dados topológica sobre a qual aplicam o algoritmo de cálculo de rotas para determinar a "melhor" rota para cada classe de serviço.

Ficou por experimentar a utilização de outro tipo de métricas de QoS, de que são exemplo a largura de banda disponível nas ligações (métrica côncava) e o atraso fim a fim, métrica aditiva. Também seria interessante obter diferentes combinações destas métricas de QoS, de forma a que fosse possível que cada classe de serviço especificasse os seus requisitos de QoS utilizando mais do que uma métrica. Parece ser fácil de extender o protocolo nesse sentido, mas são coisas que ficaram por fazer, e assim aqui fica como sugestão para trabalho futuro.

Outro aspecto que ficou por contemplar foi uma comparação dos protocolos desenvolvidos com protocolos que implementam o encaminhamento por classes de serviço. Os

protocolos usados como referencial de comparação com os protocolos desenvolvidos foram uma implementação de um protocolo de estado de ligação semelhante ao protocolo OSPF, no caso do encaminhamento *unicast*, e no caso do encaminhamento *multicast* uma implementação, desenvolvida especificamente para esse efeito, do protocolo PIM-SM. Fazia falta comparar os protocolos aqui propostos com protocolos do mesmo tipo, ou seja protocolos que implementassem o encaminhamento por classes de serviço. Na altura em que o trabalho teve inicio não se conhecia nenhuma proposta deste tipo que pudesse ser comparável com os protocolos propostos. Agora já se conhecem algumas propostas, pelo menos ao nível do encaminhamento *unicast*, por isso aqui fica a sugestão, para trabalho futuro.

Um outro desenvolvimento passa certamente pelo estudo da integração destes novos mecanismos de encaminhamento interno com QoS com soluções de encaminhamento externo que também contemplem serviços diferenciados e encaminhamento *multicast*.

Uma outra área de desenvolvimento futuro poderá ser a utilização das métricas e parâmetros que condicionam o encaminhamento (*unicast* e *multicast*) por outros mecanismos de diferenciação no nível de rede, nomeadamente como mecanismos auxiliares para processos de controlo de admissão e de escalonamento.

O funcionamento em ambientes mistos, nomeadamente o estudo da coexistência e problemas de integração com redes PIM-SM (puras) é também um outro aspecto que deverá ser analisado, atendendo à base PIM-SM já instalada.

Bibliografia

[1] ISO. *Information technology—Open Systems Interconnection—Basic Reference Model: The Basic Model*. ISO/IEC, 7498-1 edition, 1994.

[2] A. Mankin, Ed., F. Baker, B. Braden, S. Bradner, M. O'Dell, A. Romanow, A. Weinrib, and L. Zhang. Resource ReSerVation protocol (RSVP) – version 1 applicability statement some guidelines on deployment. Request for Comments 2208, Internet Engineering Task Force, September 1997.

[3] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated service. Request for Comments 2475, Internet Engineering Task Force, December 1998.

[4] S. Brim, B. Carpenter, and F. Le Faucheur. Per hop behavior identification codes. Request for Comments 2836, Internet Engineering Task Force, May 2000.

[5] K. Fall and K. Varadhan. *The NS Manual*, Jan 2001. URL=http://www.isi.edu/nsnam/ns/ns-documentation.html.

[6] António Costa, Maria João Nicolau, Alexandre Santos and Vasco Freitas. Implementação e teste do PIM-SM no Network Simulator. In *Conferência sobre Redes de Computadores 2002*, September 2002. *In Portuguese.*

[7] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei. Protocol independent multicast-sparse mode (PIM-SM): protocol specification. Request for Comments 2362, Internet Engineering Task Force, June 1998.

[8] Maria João Nicolau, António Costa, Alexandre Santos and Vasco Freitas. Directed Trees in Multicast Routing. In *Quality of Service in Multiservice IP Networks QoSIP2003*, February 2003. LNCS 2601, pp 321-333, Ed. Marco A. Marsan et al, Springer-Verlag Berlin Heidelberg 2003.

[9] Maria João Nicolau, António Costa, Alexandre Santos and Vasco Freitas. A Class Based Unicast Routing Protocol. Technical report, Universidade do Minho, 2003.

[10] J. Moy. OSPF version 2. Request for Comments 2328, Internet Engineering Task Force, April 1998.

[11] Maria João Nicolau, António Costa and Alexandre Santos. A Framework for Multi-class-based Multicast Routing. In *Terena Networking Conference 2002*, June 2002.

[12] Maria João Nicolau, António Costa, Alexandre Santos and Vasco Freitas. Towards Multi-class Based Multicast Routing. In *6th IEEE Int Conf on High Speed Networks and Multimedia Communications HSNMC2003*, June 2003. LNCS 2720, pp. 52-61, Ed M. M. Freire et al, Spinger-Verlag Berlin Heidelberg 2003.

[13] C. L. Hedrick. Routing information protocol. Request for Comments 1058, Internet Engineering Task Force, June 1988.

[14] Mark A Sportack. *IP Routing Fundamentals*. Cisco Press, 1999.

[15] Y. Rekhter and T. Li. A border gateway protocol 4 (BGP-4). Request for Comments 1654, Internet Engineering Task Force, July 1994.

[16] J. Moy. OSPF version 2. Request for Comments 2178, Internet Engineering Task Force, July 1997.

[17] J. Postel. Internet control message protocol. Request for Comments 792, Internet Engineering Task Force, September 1981.

[18] K. Thulasiraman and M. N. S. Swamy. *Graphs : theory and algorithms*. John Wiley & Sons, Inc., 1992.

[19] V. K. Balakrishnan. *Network optimization*. CRC Press, Inc., 1995.

[20] Christian Huitema. *Routing in the Internet*. Prentice Hall, 2000.

[21] Uyless Black. *IP routing protocols : RIP, OSPF, BGP, PNNI, and cisco routing protocols*. Prentice Hall, 2000.

[22] Sam Halabi. *Internet routing architectures*. Cisco Press, 2001.

[23] J. Moy. OSPF specification. Request for Comments 1131, Internet Engineering Task Force, October 1989.

[24] K. Lougheed and Y. Rekhter. Border gateway protocol (BGP). Request for Comments 1163, Internet Engineering Task Force, June 1990.

[25] G. Malkin. RIP version 2. Request for Comments 2453, Internet Engineering Task Force, November 1998.

[26] G. Malkin and R. Minnear. RIPng for IPv6. Request for Comments 2080, Internet Engineering Task Force, January 1997.

[27] K. Lougheed and Y. Rekhter. Border gateway protocol 3 (BGP-3). Request for Comments 1267, Internet Engineering Task Force, October 1991.

[28] Y. Rekhter and T. Li. A border gateway protocol 4 (BGP-4). Request for Comments 1771, Internet Engineering Task Force, March 1995.

[29] J. Hawkinson and T. Bates. Guidelines for creation, selection, and registration of an autonomous system (AS). Request for Comments 1930, Internet Engineering Task Force, March 1996.

[30] V.K Balakrishnam. *Network Optimization*. Chapman and Hall, 1995.

[31] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for Steiner trees. *Acta Informatica*, 15:141–145, 1981.

[32] Yogen K. Dalal and Robert M. Metcalfe. Reverse path forwarding of broadcast packets. *Commun. ACM*, 21(12):1040–1048, 1978.

[33] D. Estrin, D. Farinacci, A. Helmy, V. Jacobson, and L. Wei. Protocol independent multicast (pim) dense mode protocol specification, 1996.

[34] D. Waitzman, C. Partridge, and S. E. Deering. Distance vector multicast routing protocol. Request for Comments 1075, Internet Engineering Task Force, November 1988.

[35] J. Moy. MOSPF: analysis and experience. Request for Comments 1585, Internet Engineering Task Force, March 1994.

[36] A. Ballardie. Core based trees (CBT version 2) multicast routing. Request for Comments 2189, Internet Engineering Task Force, September 1997.

[37] Hans Eriksson. Mbone: the multicast backbone. *Commun. ACM*, 37(8):54–60, 1994.

[38] J. Moy. Multicast extensions to OSPF. Request for Comments 1584, Internet Engineering Task Force, March 1994.

[39] J. Wroclawski. The use of RSVP with IETF integrated services. Request for Comments 2210, Internet Engineering Task Force, September 1997.

[40] J. Wroclawski. Specification of the controlled-load network element service. Request for Comments 2211, Internet Engineering Task Force, September 1997.

[41] S. Shenker, C. Partridge, and R. Guerin. Specification of guaranteed quality of service. Request for Comments 2212, Internet Engineering Task Force, September 1997.

[42] R. Braden, Ed., L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation protocol (RSVP) – version 1 functional specification. Request for Comments 2205, Internet Engineering Task Force, September 1997.

[43] J. Postel. Internet protocol. Request for Comments 791, Internet Engineering Task Force, September 1981.

[44] P. Almquist. Type of service in the internet protocol suite. Request for Comments 1349, Internet Engineering Task Force, July 1992.

[45] Z. Wang and J. Crowcroft. Bandwidth-delay based routing algorithms. In *Proceedings of the GLOBECOM'95*, pages 2129–2133, 1995.

[46] R. Guerin. Qos routing in networks with inaccurate information: Theory and algorithms, 1999. IEEE Transactions on Networking, Vol. 7, No. 3, June 1999.

[47] G. Apostolopoulos, S. Kama, D. Williams, R. Guerin, A. Orda, and T. Przygienda. QoS routing mechanisms and OSPF extensions. Request for Comments 2676, Internet Engineering Task Force, August 1999.

[48] Hussein F. Salama, Douglas S. Reeves, and Yannis Viniotis. A distributed algorithm for delay-constrained unicast routing. In *INFOCOM (1)*, pages 84–91, 1997.

[49] Quan Sun and Horst Langendorfer. A distributed delay-constrained dynamic multicast routing algorithm. In *European Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS'97)*, pages 97–106, 1997.

[50] L. Costa, S. Fdida, and O. Duarte. A scalable algorithm for linkstate QoS-based routing with three metrics. In *Proceedings of the IEEE International Conference on Communications — ICC'2001*, June 2001.

[51] K. Shin and C. Chou. A distributed route-selection scheme for establishing real-time channel. In *Proceedings of Sixth IFIP Int'l Conf. on High Performance Networking*, pages 319–329, 1995.

[52] Shigang Chen and Klara Nahrstedt. Distributed quality-of-service routing in high-speed networks based on selective probing. In *LCN - Local Computer Networks*, pages 80–89, 1998.

[53] B. Wang and J. Hou. Multicast routing and its QoS extension: problems, algorithms, and protocols. *IEEE Network*, 14, January 2000.

[54] H. Tyan, J. Hou, B. Wang, and Y. Chen. Qos extension to the core based tree protocol, 1999.

[55] R. Izmailov S. Biswas and B. Rajagopalan. A qos-aware routing framework for pim-sm based ip-multicast, June 1999. IETF Internet Draft.

[56] Vachaspathi P. Kompella, Joseph C. Pasquale, and George C. Polyzos. Multicast routing for multimedia communication. *IEEE/ACM Transactions on Networking*, 1(3):286–292, 1993.

[57] F.A. Kuipers and P. Van Mieghem. Mamcra: a constrained-based multicast routing algorithm. *Computer Communications, vol. 25/8*, pages 801–810, 2002.

[58] F. A. Kuipers and P. Van Mieghem. QoS routing: Average complexity and hopcount in m dimensions. *Lecture Notes in Computer Science*, 2156:110–??, 2001.

[59] K. Carlberg and J. Crowcroft. Building shared trees using a one-to-many joining mechanism. *ACM Computer Communication Review*, pages 5–11, 1997.

[60] Michalis Faloutsos, Anindo Banerjea, and Rajesh Pankaj. Qosmic: Quality of service sensitive multicast internet protocol. In *SIGCOMM*, pages 144–153, 1998.

[61] Shigang Chen, Klara Nahrstedt, and Yuval Shavitt. A qos-aware multicast routing protocol. In *INFOCOM (3)*, pages 1594–1603, 2000.

[62] Ion Stoica, T. S. Eugene Ng, and Hui Zhang. REUNITE: A recursive unicast approach to multicast. In *INFOCOM (3)*, pages 1644–1653, 2000.

[63] Luís Henrique M.K. Costa, Serge Fdida, and Otto Carlos M.B. Duarte. Hop-by-hop multicast routing protocol. In *ACM SIGCOMM'2001*, pages 249–259, August 2001.

[64] W. Fenner. Internet group management protocol, version 2. Request for Comments 2236, Internet Engineering Task Force, November 1997.

[65] George Apostolopoulos, Roch Guerin, Sanjay Kamat, and Satish K. Tripathi. Quality of service based routing: A performance perspective. In *SIGCOMM*, pages 17–28, 1998.

[66] K. Calvert and E.W. Zegura. *GT-ITM: Georgia Tech internetwork topology models* (software), 1996.
URL=http://www.cc.gatech.edu/fac/Ellen.Zegura/gt-itm/gt-itm.tar.gz.

[67] P. Winter. Steiner problem in networks: A survey. *Networks*, 17:129–167, 1987.

[68] Moses Charikar, Chandra Chekuri, To yat Cheung, Zuo Dai, Ashish Goel, Sudipto, and Ming Li. Approximation Algorithms for Directed Steiner Problems. *Journal of Algorithms*, 33(1):73–91, October 1999.

[69] S.Ramanathan. Multicast Tree Generation in Networks with Asymmetric Links. *IEEE/ACM Transations on Networking*, 4(4):558–568, 1996.

[70] J.Eric Klinker. Multicast Tree Construction in Direct Networks. In *IEEE MILCOM*, Whashington DC,USA, October 1996.

[71] Zheng Wang and Jon Crowcroft. Quality-of-service routing for supporting multimedia applications. *IEEE Journal of Selected Areas in Communications*, 14(7):1228–1234, 1996.

[72] B. Salkewicz Z. Zhang, C. Sanchez and E. Crawley. Quality of service extensions to ospf or quality of service path first routing (qospf), June 1996. draft-zhang-qos-qospf-00.ps.

[73] Gonçalo Quadros e Edmundo Monteiro Marília Oliveira, Bruno Melo. Quality of service routing in the differentiated services framework. In *Proceedings of SPIES's International Symposium on Voice, Video, and Data Communications (Internet III: Quality of Service and Future Directions)*, November 2000.

[74] Raimo Kantola Peng Zhang and Samuli Aalto. Qos routing for diffserv networks: Issues and solutions, 2002. Technical Report.

[75] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. Assured forwarding PHB group. Request for Comments 2597, Internet Engineering Task Force, June 1999.

[76] P. Berman and V. Ramaiyer. Improved approximations for the Steiner tree problem. In *Proceedings of the Third Symposium on Discrete Algorithms*, pages 325–334, 1992.

[77] Pedro Sousa, Paulo Carvalho and Vasco Freitas. End-to-End Delay Differentiation of IP Traffic Agregates using Priority Queueing Models. In *Proc of the 2002 Workshop on High Performance Switching and Routing (HPSR2002)*, pp 178-182, IEEE Comm Soc, ATM Forum, IEICE Comm Soc, Kobe, Japan, May 2002.

[78] Pedro Sousa, Paulo Carvalho and Vasco Freitas. Scheduling Time-Sensitive IP Traffic. In *6th IFIP/IEEE International Conf on Management of Multimedia Networks and Services, (MMNS 2003)*, Ed A. Marshall et al, Queen's University, Belfast, N Ireland, September 2003. LNCS 2839, pp. 368-380, 2003.

[79] Solange Lima, Paulo Carvalho, Alexandre Santos and Vasco Freita. A Distributed Admission Control Model for CoS Networks using QoS and SLS Monitoring. In *IEEE 2003 International Conference on Comunications (ICC 2003)*, CQ6-1: QoS Control in Networks, Anchorage, Alaska, USA, May 2003.

[80] Solange Lima, Paulo Carvalho, Alexandre Santos and Vasco Freitas. Managing Services Quality through Admission Control and Active Monitoring. In *6th IFIP/IEEE International Conf on Management of Multimedia Networks and Services, (MMNS 2003)*, Ed A. Marshall et al, Queen's University, Belfast, N Ireland, September 2003. LNCS 2839, pp. 142-154, 2003.

[81] António Costa, Maria João Nicolau, Alexandre Santos and Vasco Freitas. Policy Aware QoS Inter-domain Multicast Routing. In *Workshop on High Performance Switching and Routing HPSR 2003*, July 2003. pp 275-280, IEEE Comm Soc, Cat #03TH8660, ISBN 0-7803-7710-9, Libr of Congr #200211489.