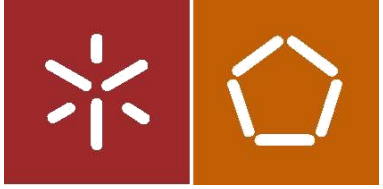
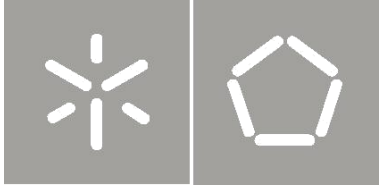


Universidade do Minho
Escola de Engenharia

Hugo Luís da Cunha Ferreira

Encaminhamento *Anycast* em redes *IPv6*.

Outubro de 2012



Universidade do Minho
Escola de Engenharia

Hugo Luís da Cunha Ferreira

Encaminhamento *Anycast* em redes *IPv6*.

Dissertação de Mestrado
Ciclo de Estudos Integrados Conducentes ao
Grau de Mestre em Engenharia de Comunicações

Trabalho efetuado sob a orientação de:
Professora Doutora Maria João Nicolau
Professor Doutor António Costa

Outubro de 2012

Não sou nada.

Nunca serei nada.

Não posso querer ser nada.

À parte isso, tenho em mim todos os sonhos do mundo.

Álvaro de Campos em "Poemas"

Agradecimentos

O presente trabalho encerra um ciclo de aprendizagem, onde é imprescindível apresentar os meus sinceros agradecimentos a algumas pessoas.

Começo por agradecer aos meus orientadores, Professora Doutora Maria João Nicolau e Professor Doutor António Costa, pelo seu esforço, dedicação e empenho, realçando a sua ajuda para conseguir levar o trabalho "a bom porto".

Gostaria de deixar uma palavra de apreço a todos os meus amigos, que tiveram um papel fundamental ao longo do meu trajeto, agradecendo a amizade e o apoio que sempre me disponibilizaram.

Aos meus pais, tia e primo, pela inestimável ajuda e compreensão. Agradeço acima de tudo, os diversos sacrifícios suportados para concluir mais uma etapa da minha vida.

Por último, à Margarida, expresso o meu profundo agradecimento pelo seu companheirismo, paciência e incentivo ao longo de todo este período e principalmente pela valorização sempre tão entusiasta do meu trabalho.

A todos, reitero o meu agradecimento!

Resumo

O aparecimento do protocolo de comunicação *Internet Protocol version 6 (IPv6)* introduziu um novo paradigma de comunicação, denominado *anycast* (um-para-um-de-muitos). Este novo paradigma, utiliza o conceito de grupo, à semelhança do que acontece com o *multicast*, mas em oposição a este, a informação é enviada apenas para um dos membros do grupo (tipicamente o mais próximo) e não para todos. Embora já se tenham passado alguns anos desde o seu aparecimento, o *anycast* tem sofrido uma lenta evolução, contribuindo para esta situação o facto de não existir ainda um protocolo normalizado, que permita às aplicações usar de forma generalizada este paradigma de comunicação.

Tradicionalmente as soluções para o problema de encaminhamento *anycast* são simplesmente baseadas no encaminhamento *unicast* sem alterações. No entanto, e tratando-se de um paradigma que usa o conceito de grupo, é de esperar que os protocolos de encaminhamento *multicast*, ou alguma variante destes, possam vir a constituir uma boa solução para a implementação do *anycast* ao nível da rede. A presente dissertação apresenta um levantamento de propostas relacionadas com o tema e propõe um novo protocolo de encaminhamento *anycast* baseado no protocolo *Protocol Independent Multicast - Sparse Mode (PIM-SM)*, denominado *Tree-based Anycast Protocol (TAP)*. As alterações propostas ao protocolo *PIM-SM* são apresentadas na especificação do sistema, tendo sido o seu correto funcionamento aferido recorrendo ao *Network Simulator 2 (ns-2.35)*.

Palavras-Chave: *Anycast*, Encaminhamento, Redes, *IPv6*, *PIM-SM*, *TAP*.

Abstract

The introduction of the new Internet Protocol version 6 (IPv6), came with a new communication paradigm, named anycast. This new paradigm, uses the group as a concept, similar to what happens with multicast, but in opposition to this, the information is sent only for one of the members on the group (usually the closest one) and not for all. Although some years have passed since its appearance, anycast had a slow development, being the main reason the fact that it doesn't have a standard protocol that allows applications to use widely this communication paradigm.

The solutions for the anycast routing problem, traditionally, are based on unicast routing without any changes. However, and being this a paradigm that uses the group concept, it's expected that multicast routing protocols, or some kind of variant, would be a good solution to implement an anycast network-based protocol. This dissertation presents a survey of proposed anycast protocols and suggests a new routing protocol based on Protocol Independent Multicast - Sparse Mode (PIM-SM), designated as Tree-based Anycast Protocol (TAP). The chapter of the system specification introduce the changes to the protocol PIM-SM, and the correct behaviour measured using the Network Simulator 2 (ns-2.35).

Keywords: Anycast, Routing, Networks, IPv6, PIM-SM, TAP.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Códigos

Lista de Acrónimos

AOSPF	**A**nycast **O**pen **S**hortest **P**ath **F**irst
ARD	**A**nycast **R**eceiver **D**iscovery
ASTAS	**A**rchitecture for **S**calable and **T**ransparent **A**nycast **S**ervices
BGP	**B**order **G**ateway **P**rotocol
BR	**B**order **R**outer
CDN	**C**ontent **D**elivery **N**etwork
DARPA	**D**efense **A**dvanced **R**esearch **P**rojects **A**gency
DDoS	**D**istributed **D**enial-**o**f-**S**ervice
DNS	**D**omain **N**ame **S**ystem
DVARP	**D**istance **V**ector **A**nycast **R**outing **P**rotocol
DVMRP	**D**istance **V**ector **M**ulticast **R**outing **P**rotocol
ECMP	**E**qual-**C**ost **M**ulti-**P**ath
GIA	**G**lobal **I**nternet Protocol **A**nycast
IANA	**I**nternet **A**ssigned **N**umbers **A**uthority
IGMP	**I**nternet **G**roup **M**anagement **P**rotocol
IPSec	**I**nternet **P**rotocol **Sec**urity
IPv4	**I**nternet **P**rotocol **v**ersion **4**
IPv6	**I**nternet **P**rotocol **v**ersion **6**
NAM	**N**etwork **A**ni**M**ator
NAT	**N**etwork **A**ddress **T**ranslation
NDP	**N**eighbor **D**iscorey **P**rotocol
NS-2	**N**etwork **S**imulator **2**
NS-3	**N**etwork **S**imulator **3**
MLD	**M**ulticast **L**istener **D**iscovery
MOSPF	**M**ulticast **O**pen **S**hortest **P**ath **F**irst
OSPF	**O**pen **S**hortest **P**ath **F**irst
OBR	**O**riginator **B**order **R**outer

PIAS	**P**roxy **I**nternet Protocol **A**nycast **S**ervice
PIA-SM	**P**rotocol **I**ndependent **A**nycast - **S**parse - **M**ode
PIM - SM	**P**rotocol **I**ndependent **M**ulticast - **S**parse - **M**ode
RIP	**R**outing **I**nformation **P**rotocol
TAP	**T**ree-based **A**nycast **P**rotocol
TCP	**T**ransmission **C**ontrol **P**rotocol
TLD	**T**op-**L**evel **D**omain
TTL	**T**ime **T**o **L**ive
VINT	**V**irtual **I**nter**N**etwork - **T**estbed
VoIP	**V**oice **o**ver **I**nternet **P**rotocol

1

Introdução

O primeiro capítulo da dissertação tem como propósito contextualizar o aparecimento do *anycast*, expor as suas vantagens e desvantagens e apresentar um exemplo do seu funcionamento. O seu conteúdo começa por enquadrar a temática abordada, sendo de seguida enumerados os principais objetivos a alcançar e, por fim, é descrita a estrutura do documento.

1.1 Enquadramento

Desde a sua criação até à atualidade, a principal característica da *Internet* é a sua constante evolução. O número de utilizadores e as suas necessidades têm variado ao longo do tempo. Atualmente, existem dois protocolos para comunicações na *Internet* a funcionar simultaneamente, o *Internet Protocol version 4 (IPv4)* e o *Internet Protocol version 6 (IPv6)*[2]. Este último, também conhecido como o protocolo da nova geração, foi introduzido para suprimir as dificuldades do primeiro.

Devido à data da sua criação, por volta de 1970, o *IPv4* foi projetado para um diferente tipo de utilizadores e requisitos. O exemplo mais elucidativo é o do endereçamento de 32 *bits*. Nos dias de hoje, este já não permite identificar todos os dispositivos que se conectam à *Internet*. Isto aconteceu devido ao mundo ubíquo em que vivemos, em que atualmente tudo se encontra na *Internet* (desde computadores pessoais até aos televisores). Com o aparecimento das chamadas de voz através da *Internet* (*Voice over Internet Protocol (VoIP)*) e um conjunto de novos serviços, também uma nova necessidade de qualidade apareceu. Tornou-se necessário classificar o tráfego, fazendo a sua distinção. Outros dois aspetos que na altura não eram muito importantes como a segurança e a mobilidade, hoje em dia, são de extrema importância.

O *IPv6* surgiu então como o digno sucessor de um protocolo que tão bem serviu os seus propósitos ao longo dos anos, mas que começa a ficar ultrapassado. Começa por disponibilizar um

número maior de endereços, 128 *bits* para endereçamento, que nos permite atribuir endereços a aproximadamente 4,3 biliões de dispositivos. Foi também mudado o cabeçalho do pacote de dados, que passou a ter tamanho fixo (40 *bytes*) e ficou mais claro. Foram removidos alguns campos e outros modificados, o que permitiu aumentar a velocidade de processamento da informação. Aliado ao facto de passarem a ser os sistemas terminais (*end-devices*) a fazerem a fragmentação dos dados, proporcionou uma ajuda preciosa ao encaminhamento. A evolução da *Internet* tem sido esmagadora, ao passo que a evolução da capacidade dos encaminhadores de tratar o tráfego é substancialmente menor. Estudos realizados na época[3], acreditavam que, nos dias de hoje, a capacidade dos encaminhadores de processar informação já teria sido superada pelo tráfego atual a circular na rede. Além destas melhorias, existe uma série de funcionalidades adicionadas que com a implementação global deste protocolo de comunicação serão muito úteis.

Com este protocolo de comunicação existem três paradigmas de comunicação - o *unicast*, o *multicast* e o *anycast*. O *unicast* é o paradigma predominante de transmissão na *Internet*, sendo a sua comunicação realizada entre dois sistemas terminais (ponto-a-ponto). O tráfego *multicast* (um-para-muitos) surgiu como uma evolução natural do antigo *broadcast* (um-para-todos). Deixa de ser para todos, e cria a ideia de grupos, onde a comunicação deixa de ser "para todos" e aparece o conceito de grupo. Passa a ser possível comunicar com um grupo definido de utilizadores. O *anycast* é originalmente proposto[4] a Novembro de 1993, sendo particularmente útil para implementar serviços que podem ser disponibilizados por múltiplos servidores. É atribuído um único endereço *anycast* a todos os sistemas terminais que disponibilizem exatamente o mesmo serviço. Para os clientes desse serviço, basta-lhes dirigir o pedido ao endereço *anycast* e esperar que um dos sistemas (e apenas um) lhe responda. Do ponto de vista do serviço prestado é indiferente qual deles é, mas a escolha do sistema em melhores condições é um problema de encaminhamento específico da rede.

Os endereços *anycast* são sintaticamente indistinguíveis dos endereços *unicast*[5]. A lógica inerente a este tipo de endereçamento é de um endereço *unicast* em diferentes locais. A ideia visa permitir a um provedor de serviços aumentar a capacidade de balanceamento de carga, acrescentando um novo servidor numa outra rede. Quando um utilizador requer um determinado serviço, este é encaminhado para o servidor mais próximo da sua localização (com menor custo), utilizando o método de encaminhamento do *unicast*.

Na figura 1.1 é possível observar um exemplo de uma comunicação *anycast*. Um endereço

anycast único é atribuído aos três provedores de serviços - `Servidor Anycast 1`, `Servidor Anycast 2`, `Servidor Anycast 3`. Quando um `Cliente Anycast` deseja efetuar um pedido, cria um pacote *anycast* (pacote que contém um endereço *anycast* como destino) e envia esse pacote para a rede. Após o envio do pacote até à receção por parte do servidor mais próximo, é levado a cabo um processo de seleção (normalmente o caminho de menor custo). Aquando da receção do pacote *anycast* por parte do servidor mais próximo do cliente, é enviada a resposta ao pedido do cliente, idealmente utilizando o endereço *anycast* no campo de origem de modo a evitar problemas na altura da receção do pacote pelo cliente.

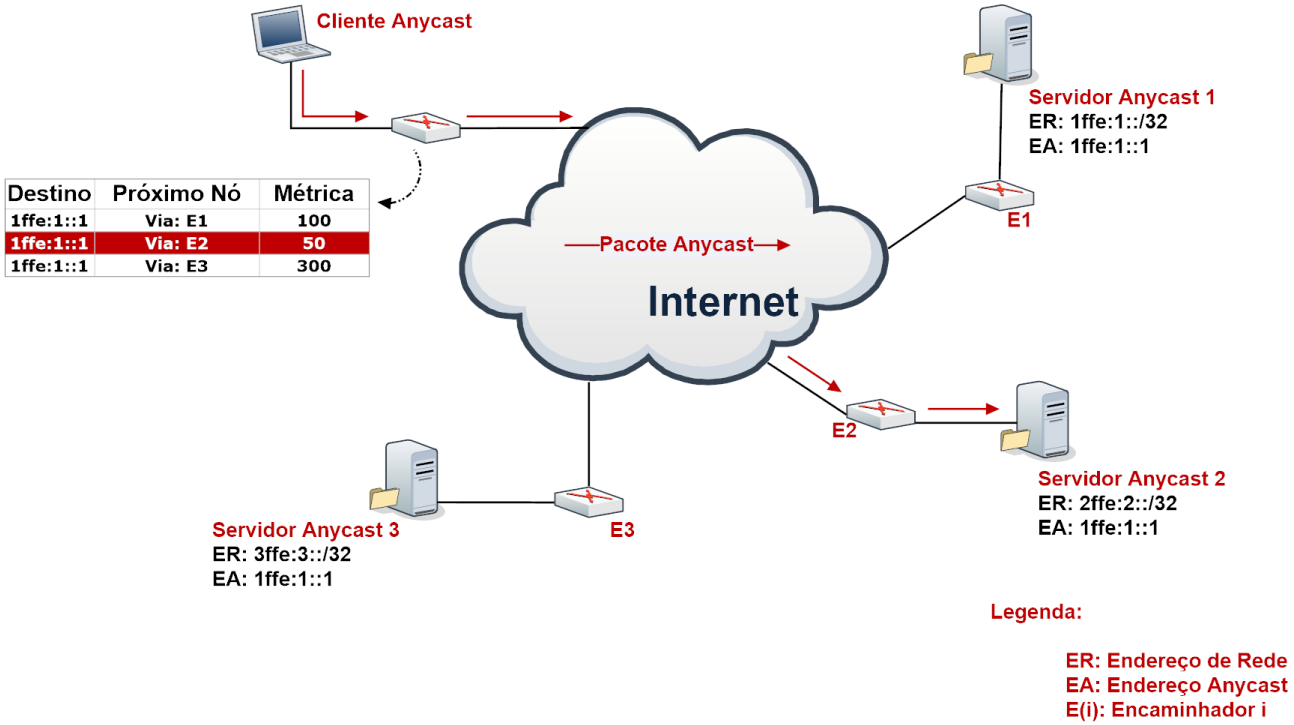


Figura 1.1: Esquema do encaminhamento *anycast*.

No encaminhamento global, o *anycast* prejudica a agregação de rotas pois permite que um mesmo endereço apareça associado a várias redes distintas. Como podemos observar na figura 1.1, é possível ao `Servidor Anycast 2` e ao `Servidor Anycast 3` estarem em redes diferentes da rede a que pertence o endereço *anycast*. Um encaminhador que receba rotas relativas a estas redes, é facilmente iludido, pensando que os servidores se encontram todos na mesma rede. Para conseguir distingui-las, é necessário adicionar entradas separadas a todos os encaminhadores ao longo da rede. Utilizando o mesmo exemplo, caso os servidores se encontrem em redes com endereços díspares pode ser muito complicado (ou mesmo impossível) efetuar a agregação de rotas. É necessário um novo sistema para encaminhamento, para retirar partido

do potencial do *anycast*. Existem diversas propostas no sentido de atacar esta problemática, mas existem ainda muitos problemas por resolver[6].

Não obstante ao problema da agregação das rotas, o encaminhamento *anycast* possuí diversas características interessantes. Uma das características mais importantes do *anycast* quando realizado na camada de rede é não ser necessário ao cliente conhecer a topologia total da rede, o que lhe confere uma enorme escalabilidade. Quando o `Cliente Anycast` na figura 1.1 faz um pedido, este é reencaminhado para o nó mais próximo dele (neste caso o `Servidor Anycast 1`). A grande vantagem acontece quando por qualquer motivo (por exemplo falha de *hardware*) o nó onde habita o `Servidor Anycast 1` deixa de estar acessível. Neste caso o seu pedido será na mesma atendido, mas por um servidor a uma maior distância.

O *anycast* tem enumeras aplicações, das quais se destacam as seguintes:

- **Serviços dependentes da localização**

  Esta é atualmente uma das mais populares aplicações do *anycast*. A escolha do servidor mais próximo do requerente do serviço é uma questão cada vez mais critica. Normalmente passa por apresentar uma lista de servidores distribuídos por múltiplos locais e oferecer o poder de seleção ao requerente do serviço. Com a utilização do *anycast* neste tipo de aplicações, garante-se que é feita uma seleção mais precisa para o encaminhamento do pedido. Em vez de confiar no juízo do requerente, realiza-se um encaminhamento mais transparente a partir de qualquer parte do mundo. Com isto o *anycast* assume um papel essencial pois permite escolher o nó mais próximo, fazendo assim com que exista uma resposta mais célere e eficaz.

- **Descoberta de serviços**

  Para ser possível a descoberta de serviços, primeiramente é atribuído um endereço *anycast* a um serviço, e de seguida é atribuído aos nós que suportam os servidores desse mesmo serviço. Isto permite uma grande tolerância a falhas na rede e a avarias de *hardware* na rede, pois o pacote *anycast* será reencaminhado para um outro local apropriado. Este tipo de serviço assume uma grande importância em redes com grande escalabilidade, como redes móveis *ad hoc* ou de sensores, onde a sua topologia muda constantemente. Nestas redes, mais importante que descobrir os serviços, é saber que existe a disponibilidade para o obter.

- **Balanceamento de carga**

  Com a quantidade de tráfego a aumentar diariamente na *Internet*, a necessidade de providenciar a resposta aos pedidos torna-se cada vez mais difícil. Existem diversas opções de como melhorar o desempenho do sistema (desde aumento da capacidade de processamento ao aumento dos sistemas). Atualmente, a melhor opção é a distribuição de vários servidores por diferentes zonas (onde se pretende providenciar o serviço). Como é possível observar na figura 1.2, os diferentes pedidos vão sendo distribuídos pelos diferentes servidores fazendo com que os pedidos não sobrecarreguem somente uma rede. Diminui-se ainda o tempo de resposta, pois o processamento de pedidos é dividido pelos vários elementos do grupo.

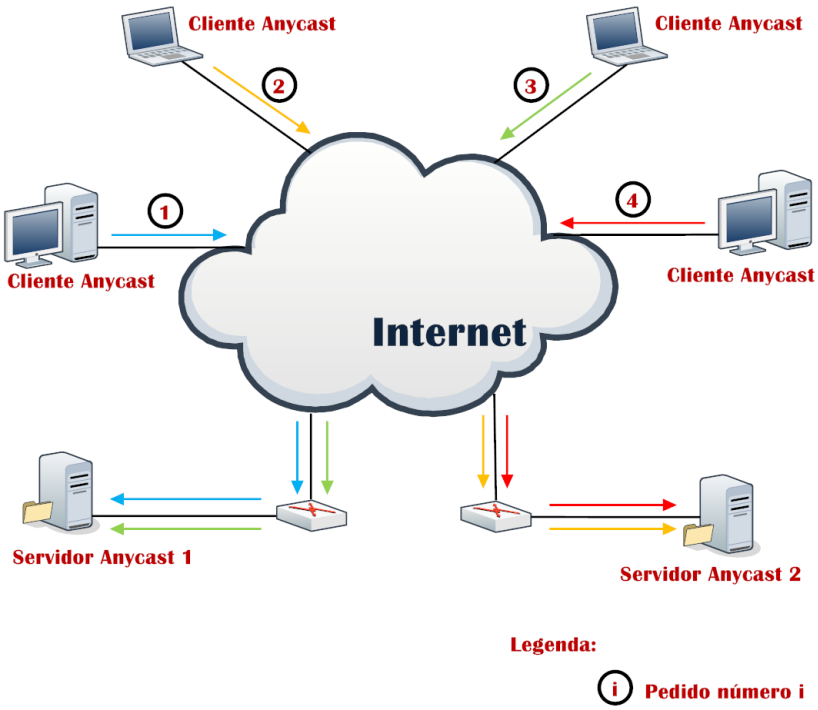


Figura 1.2: Funcionamento do balanceamento da rede

As aplicações referidas anteriormente seriam especialmente úteis as *Content Delivery Network (CDN)*[7]. Uma *CDN* pode ser descrita como um grupo de servidores, distribuídos pelo mundo, dedicados exclusivamente à entrega de diferentes conteúdos. A disponibilidade e o desempenho são os seus aspetos mais relevantes das *CDN's* .

Hoje em dia, praticamente todas as grandes empresas (em caso de exemplo a *Google*[1]) da

[1] `https://developers.google.com/speed/public-dns/faq#whatis`

área estão cientes dos benefícios do uso do encaminhamento *anycast*. Os servidores raiz do *DNS* são talvez o melhor exemplo da necessidade (e sucesso) da utilização deste tipo de encaminhamento. Através das especificações, o número de servidores raiz é limitado em todo o mundo. Estes são considerados o centro da *Internet* - o ponto vital.

No dia 21 de Outubro de 2002 estes foram atacados, através de um ataque distribuído de negação de serviço (*DDoS*) com o objetivo de paralisar a *Internet*[8]. Em caso de sucesso, as consequências económicas seriam muito graves. Felizmente devido ao bom trabalho dos operadores que controlam o *DNS*, o ataque foi ineficaz, no entanto permitiu abrir os olhos para um nova realidade (um melhor encaminhamento de tráfego) e uma nova solução começou a ser equacionada - o *anycast*.

Cinco anos após este ataque, a 6 de Fevereiro de 2007, um novo ataque foi realizado. Este ataque foi dez vezes maior e durou dez vezes mais tempo que o ataque anterior. Apesar do sistema como um todo, ter provado mais uma vez a sua resistência, os atacantes conseguiram afetar a performance de dois dos treze servidores raiz do *DNS*.

Dos treze servidores raiz do *DNS*, pelo menos seis foram afetados mas somente dois sentiram um forte abalo - o *G-root* (controlado pelo Departamento de defesa dos Estados Unidos da América) e o *I-root* (dirigido pelo *ICANN*) ambos situados nos Estados Unidos da América (Ohio e Califórnia respetivamente).

O motivo pelo qual estes dois foram tão severamente afetados foi pelo facto de não terem encaminhamento *anycast* (os outros três sem o *anycast* não foram atacados desta vez). Apesar dos ataques distribuídos de negação de serviços serem, como o próprio nome indica distribuídos, as *botnets* (grupos de computadores comprometidos) usadas para lançar os ataques tendem a não ser distribuídos uniformemente. Normalmente as *botnets* são projetadas para regiões especificas, possuindo uma região de funcionamento. No caso do ataque de 2007, a origem da maioria do tráfego provinha da zona da Ásia-Pacífico. Neste caso, os servidores implementados na zona absorviam o impacto do ataque, permitindo o normal funcionamento em outras regiões. Esta é uma das vantagens da implementação de serviços dependentes da localização (previamente explicado) recorrendo ao encaminhamento *anycast*. A não utilização do encaminhamento *anycast* nos servidores foi deliberada, pois na altura, alguns quadrantes consideravam o *anycast* como uma tecnologia imatura para ser utilizada num recurso tão importante.

A ilação retirada deste ataque foi importante para todos os que utilizam a *Internet* para qualquer tipo de negócio que depende da disponibilidade da rede. No início de 2009, pelo menos dez dos treze servidores raiz utilizam o *anycast*[9]. Atualmente, nem só os domínios de alto nível (*TLD*) implementam o *anycast*, mas muitas companhias descobriram que podem beneficiar com a sua correta utilização. Tendo como exemplo o serviço *DNS* é possível afirmar que o *anycast* tornou-o mais confiável, com maior performance e mais sólido contra ataques (*DDoS*).

1.2 Objetivos

O *anycast* é um paradigma de comunicação ainda relativamente desconhecido, sendo um dos objetivos deste trabalho realizar um estudo mais aprofundado sobre o *anycast*, como sugerido por alguns autores[6]. O principal objetivo e futuro contributo é a criação de um protocolo *anycast* ao nível inter-domínio. O trabalho desenvolvido ao longo desta dissertação incidiu na realização dos seguintes objetivos, previamente definidos:

- Compreender o funcionamento do paradigma de comunicação *anycast*.
- Estudar aprofundadamente um conjunto de protocolos inerentes ao tema e definir as suas principais limitações
- Especificar um protocolo *anycast* exequível.
- Implementar um protótipo que permita avaliar a solução proposta.
- Comparar os resultados obtidos com outras abordagens semelhantes.

1.3 Estrutura da tese

O presente documento está estruturado em seis capítulos, contendo ainda três anexos relativos ao trabalho desenvolvido.

O primeiro capítulo contém uma breve introdução ao tema desta dissertação, apresentando os principais desafios relativos ao *anycast*. É descrito ainda um exemplo real das vantagens da sua utilização e especificados os objetivos a serem alcançados. A descrição da estrutura,

adotada na elaboração da dissertação, termina este capítulo.

O capítulo 2 apresenta uma secção de contextualização ao *anycast*, onde é realizado um levantamento dos principais acontecimentos da sua história e as suas problemáticas. Apresenta ainda, através da utilização do emulador[10], o estado do encaminhamento *anycast* no panorama atual. O estado da arte relativamente ao encaminhamento *anycast* também é aqui dissecado, através do estudo das propostas existentes. Por fim, é realizado um resumo do capítulo, sendo realizada uma comparação entre os diferentes protocolos estudados e os seus resultados, apresentados segundo uma tabela conclusiva.

A especificação do sistema é apresentada no capítulo 3, explanando a arquitetura proposta para permitir o encaminhamento *anycast* ao nível do inter-domínio. Para facilitar a compreensão do sistema e seus benefícios, é descrito um exemplo prático da utilização do novo protocolo proposto, o *Tree-based Anycast Protocol (TAP)*.

No capítulo 4 é efetuada a descrição da implementação do sistema, começando por analisar qual o melhor simulador para o utilizar. De seguida, são descritas as funcionalidades do simulador escolhido[11] e os parâmetros necessários à implementação do novo protocolo. A parte final deste capítulo, apresenta os principais aspetos do protocolo implementado, sendo explicado os seus principais métodos.

O capítulo 5 apresenta a validação do *TAP*, começando por ser feita uma validação da implementação. O seu funcionamento é comprovado utilizando dois diferentes cenários. O primeiro cenário tem como objetivo retratar todos os casos abordados durante a especificação do sistema, utilizando a topologia apresentada no capítulo 3 como base. O segundo cenário é o mais imprevisível, para testar eventuais casos não considerados. É ainda realizado um teste comparativo com um protocolo estudado, relativamente à distância entre o cliente e o servidor escolhido.

No último capítulo, o sexto, são expostas as conclusões obtidas ao longo do trabalho desenvolvido. É realizada a aferição dos objetivos inicialmente propostos e delineadas algumas indicações para trabalho futuro.

Um documento auxiliar, denominado Tradução Adotada, foi incluído no documento para permitir perceber a tradução adotada ao longo da dissertação.

2

Comunicação de um-para-um-de-muitos

O segundo capítulo da dissertação funciona como um complemento ao anterior, debruçando-se sobre a história do *anycast* e as suas problemáticas. Contém ainda uma retrospetiva dos trabalhos realizados sobre o encaminhamento *anycast*, de modo a estabelecer o suporte para a implementação de uma nova abordagem. A primeira secção começa por fazer uma contextualização do *anycast*, relatando a sua evolução. São abordadas ainda outras temáticas interessantes para a tecnologia, mas que não são fundamentais para o trabalho realizado. A segunda secção apresenta o estado do encaminhamento *anycast* no panorama atual. Por fim, a última secção apresenta um conjunto de trabalhos que se propuseram a solucionar o problema do encaminhamento *anycast*.

2.1 Contextualização

Inicialmente, a utilização de endereços *anycast* em tráfego *IPv6* era muito restritiva. As principais restrições passavam pelo facto de não poderem ser utilizados como endereço de origem (o que trazia uma enorme dificuldade na resposta a um pedido, devido a erros na tradução de endereços de rede aquando a receção por parte do requerente) e pelo facto de a um simples sistema terminal (*host*) não poder ser atribuído um endereço *anycast* (deveria ser o encaminhador de saída a controlar um ou vários provedores). Só em Fevereiro de 2006 estas e outras restrições foram levantadas[12], sendo até recomendado que o servidor *anycast* responda a pedidos com o endereço *anycast* como endereço de origem de modo a prevenir a exposição da estrutura interna da sua rede.

Com este novo conjunto de diretivas, surgiu a definição do formato do endereço *anycast* (figura 2.1) . Para qualquer endereço *anycast* atribuído, existe um maior prefixo P que identifica a região topológica em que todos os servidores residem. Dentro da região identificada pelo prefixo P, é necessário manter o endereço *anycast* como um entrada separada na tabela de enca-

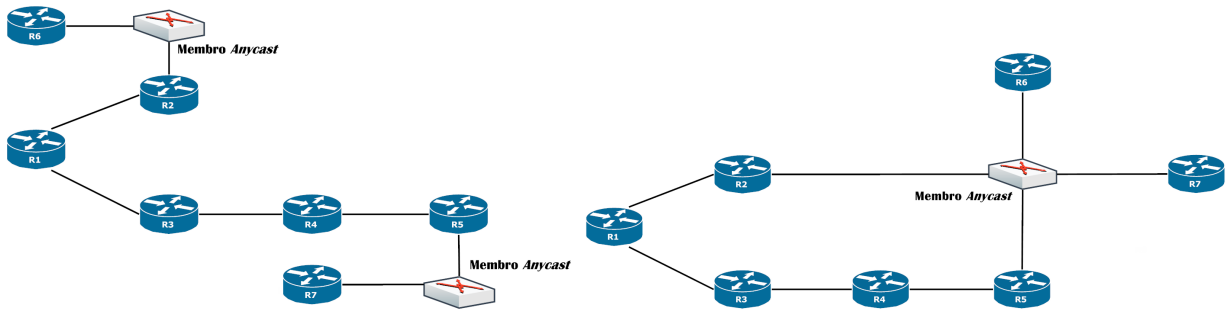
minhamento (*"host route"*). No pior caso possível o prefixo P pode ser nulo, quando não existe possibilidade de agregar as rotas. Neste caso, mantêm-se uma entrada separada na tabela de encaminhamento em toda a *Internet*, o que levanta um grande problema na sua utilização global.

Figura 2.1: Formato do endereço *Anycast*.

A Dezembro de 2006, é disponibilizado um documento acerca das mais corretas e recentes práticas nas operações com serviços *anycast*[13]. Este documento desenhado para servir de apoio a todos as pessoas que pretendem construir/gerir um serviço distribuído utilizando os benefícios do *anycast*, define uma terminologia, regras de encaminhamento intra-domínio, regras de encaminhamento inter-domínio e faz um análise dos seus riscos.

O encaminhamento ao nível da camada de rede divide-se em duas principais áreas, intra-domínio e inter-domínio. No encaminhamento *anycast*, a primeira destas áreas é a única que se encontra operacional (como podemos visualizar na secção 2.2.1). Caso o algoritmo implementado seja baseado em vetores de distância, como por exemplo *Routing Information Protocol (RIP)*, não existe nenhum desafio pois estes já se comporta como o desejável no tráfego *anycast* (se existirem dois ou mais endereços na rede interna, o pacote será entregue ao mais próximo).

O principal desafio ocorre quando se pretende utilizar um algoritmo de estado de ligação. É necessário garantir que os encaminhadores não cometam um erro e assumam que duas diferentes máquinas em locais distintos na rede, sejam consideradas uma só. As ilustrações seguintes (figuras 2.2(a) e 2.2(b)) permitem perceber o problema com maior clareza. Como podemos observar a distância entres os encaminhadores `R6` e `R7` (caso consideremos como métrica o número de saltos) na rede real é de oito saltos. Devido à forma como é implementado, o *Open Shortest Path First (OSPF)* tem uma visão distinta da rede. Este considera que os encaminhadores `R6` e `R7` são um só, ficando à distância de um salto. Apesar de não ser um erro que impossibilite a sua implementação, é um ponto importante a ter em conta para todos aqueles que a pretendam realizar.



(a) Rede real. (b) Visão por parte do *OSPF* da rede.

Figura 2.2: Compreensão do *anycast* por parte do *OSPF*.

A maior dificuldade do encaminhamento *anycast* acontece na área do inter-domínio. O principal problema é mesmo o da agregação das rotas, como referido anteriormente. Ao contrário do encaminhamento intra-domínio onde não agregar não trazia problemas, nesta área trata-se de uma questão vital. A propagação de cada rota para um serviço *anycast* apresenta um grande problema de escalabilidade, pois caso um provedor de serviços tenha por exemplo, nove servidores em diferentes locais, será necessário injetar nas tabelas de encaminhamento ao longo da *Internet* nove rotas. Isto acontece pois um encaminhador não consegue fazer a distinção entre um endereço *anycast* e um endereço *unicast*, como acontece no caso do endereço *multicast*. No caso do desejo de implementar o *anycast* globalmente este é um grande revés, pois obriga a que pesadas operações sejam realizadas no (*backbone*) da *Internet*. Atualmente, a prática passa por utilizar o prefixo mais curto (que cubra as diferentes redes onde os servidores estão) para anunciar as rotas, o que no caso de encaminhamento *anycast* a nível global pode ser um grande problema.

É necessário perceber que existem algumas considerações de segurança para a utilização do *anycast*, principalmente ao nível global. Como o tráfego passa por diferentes sistemas autónomos ao longo do seu percurso, é muito difícil controlar o seu correto funcionamento e eliminar potenciais ameaças. A maior ameaça neste tipo de tráfego trata-se do "roubo de serviço" (*service hijacking*). Esta consiste na possibilidade de uma pessoa não autorizada começar a anunciar rotas para um determinado serviço *anycast* na rede, capturando assim pedidos legítimos de tráfego ou de processo, comprometendo o serviço. É muito difícil para um cliente ou para um provedor de um serviço detetar uma pessoa com más intenções.

Quando alguém deseja implementar um serviço robusto utilizando esta tecnologia deve seguir um conjunto de simples passos que certificam um correto funcionamento. Somente servidores

legítimos *anycast* deverão ter a possibilidade de anunciar as suas rotas como provedores de um determinado serviço, o que pode levar a que os clientes desse serviço necessitem de ter o conhecimento que um servidor se encontra certificado. A comunicação poderá ser confidencial, de modo a evitar que alguém não autorizado consiga perceber o conteúdo das mensagens. Os clientes de um determinado serviço devem sempre verificar se a resposta a um pedido é atual, de modo a impedir ataques por repetição.

Como referido no início, o *IPv6* trouxe enumeras melhorias ao encaminhamento ao nível da rede. Uma das principais aconteceu na segurança com a introdução da segurança ao nível de rede (*Internet Protocol Security (IPSec)*). Com o auxílio do *IPSEC* é possível garantir autenticidade, confidencialidade, integridade e ainda proteger uma conexão contra ataques por repetição. A implementação do *IPv6* tem sido efetuada de um modo gradual, operando em simultâneo com o *IPv4*, sendo já suportado por inúmeras aplicações por todo o mundo.

Hoje em dia existem algumas propostas para a utilização do *IPSec* para assegurar a segurança nas comunicações *anycast*[14], tentando aliar a proteção oferecida pelo *IPsec* com as capacidades de distribuir a carga pelos servidores, isto é, na descoberta de serviços do modelo de comunicação Anycast.

Outra das limitações do *anycast* acontece na camada de transporte, devido à grande dificuldade de manter conexões contínuas (*TCP*) entre o requerente e o servidor provedor do serviço. Como a lógica é possuir uma grande escalabilidade, quando existe uma mudança na topologia pode implicar uma rutura da sessão *TCP*. A sincronização de estado pode muitas vezes ser complexa, implicar muitos recursos computacionais e nada amigável de implementar. Não obstante a isto, nos últimos anos têm sido feitos progressos nesta problemática, existindo vários estudos sobre a difusão de conteúdos através do *TCP* para serviços *anycast*[15].

Caso existam múltiplos caminhos de igual custo (*ECMP*) entre as extremidades da comunicação *anycast*, diversos problemas podem acontecer. Normalmente quando existem múltiplos caminhos de igual custo os resultados podem variar, isto é, não existe o compromisso de transmitir sempre para o mesmo servidor. Uma das hipóteses a utilizar, é mudar as opções de encaminhamento na topologia, mudando por exemplos alguns custos das ligações. A figura 2.3 permite observar que quando possuímos métricas diferentes para um mesmo endereço, o pedido realizado por uma máquina pode ser encaminhado por diferentes caminhos.

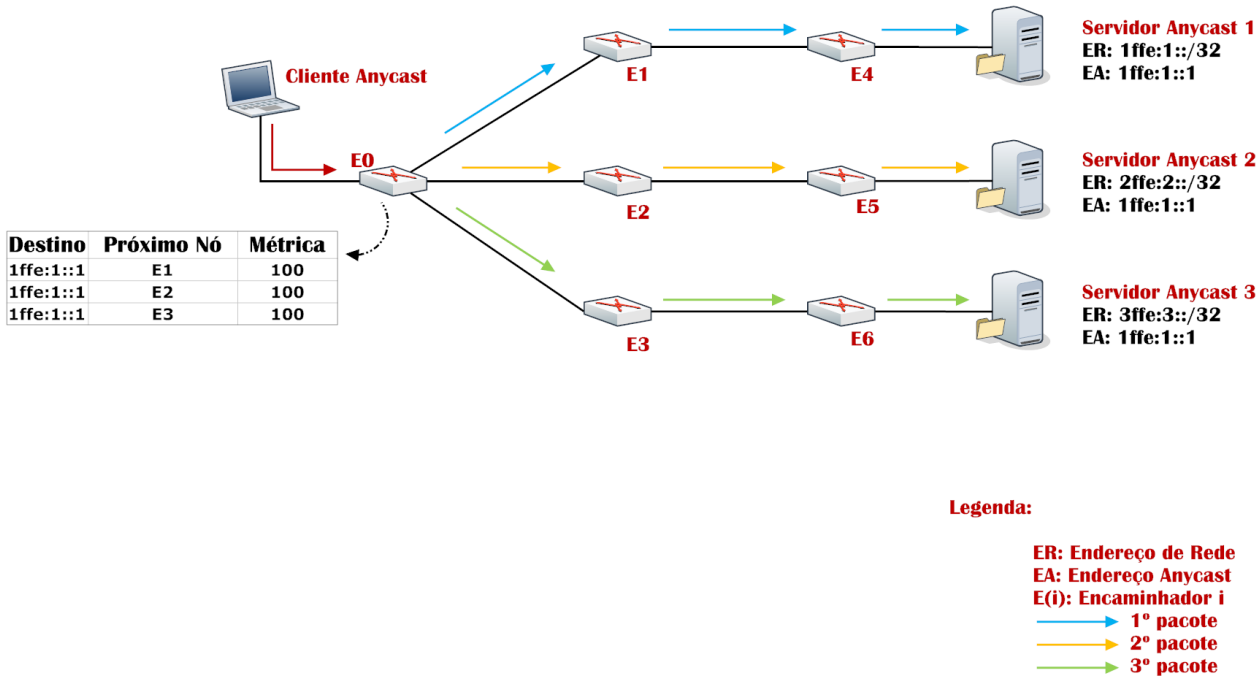


Figura 2.3: *ECMP*

O problema principal acontece quando existe uma mudança na topologia entre pedidos de clientes, podendo comunicar com o `Servidor 1` e no pedido seguinte com o `Servidor 2`. Isto pode a levar que os testemunhos de conexão (*cookies*) fiquem fora de sincronismo, que sítios protegidos por autenticação desconectem o cliente, entre outros. A alternativa para este tipo de problemas pode passar por algum tipo de sincronização entre todos os servidores provedores de um determinado serviço, utilizando por exemplo outro tipo de endereços que não *anycast*.

2.2 Enquadramento prático

Com o trabalho desenvolvido nesta secção pretende-se demonstrar o real comportamento do *anycast* nas redes reais. Para isso, utilizou-se o emulador gráfico da *Cisco Systems*, o *GNS3*[10]. O primeiro cenário pretende demonstrar o funcionamento intra-domínio e o seguinte funcionamento inter-domínio, com vários sistemas autónomos a influenciar na distribuição de carga.

2.2.1 Intra-domínio

O interior de um sistema autónomo é um ambiente controlado, recorrendo a políticas definidas pelo seu administrador, tornando-se assim o cenário ideal para testar qualquer tipo de implementação devido ao total conhecimento da rede. A primeira simulação concentrou-se no ambiente intra-domínio. Após uma cuidadosa inspeção à lista de comandos do *software* proprietário da *Cisco Systems* foi possível identificar um correspondente a um comando *anycast*. A empresa líder do mercado no seu segmento tem implementado nas suas configurações um comando que permite aos pacotes serem encaminhados de acordo com a filosofia *anycast*, para o nó mais próximo.

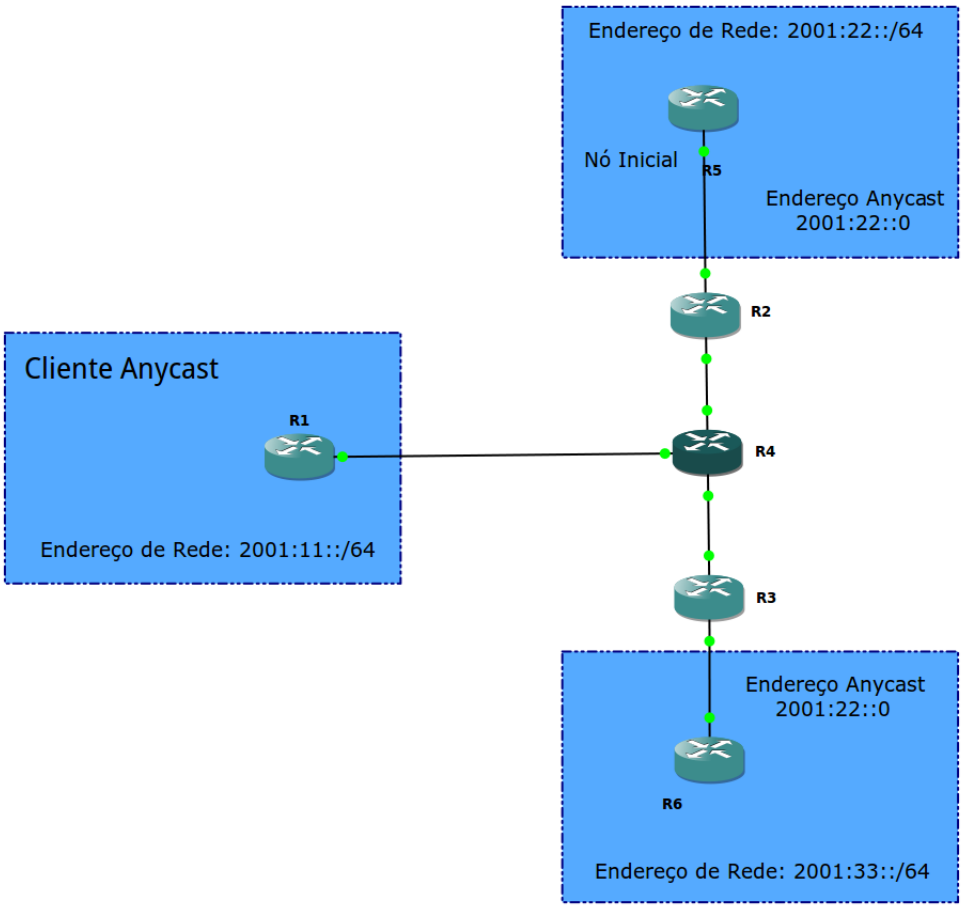


Figura 2.4: Topologia implementada.

O cenário desta demonstração (figura 2.4), contém quatros encaminhadores. O encaminhador central tem o prepositivo de dividir as diferentes áreas da demonstração. A razão de se ter optado por apenas um encaminhador (e não por diversos interligados) deveu-se ao facto de poupar recursos, não alterando em nada o comportamento da rede. Todos os encaminhadores foram configurados com o protocolo estado de ligação mais utilizado (*OSPF*), estando

divididos por três áreas. As ligações entre `R1-R4`, `R2-R4` e `R3-R4` encontram-se na área 0, convencionalmente chamada de *backbone*. A ligação entre `R2-R5` situa-se na área 2 e a ligação entre `R3-R6` na área 3. Após efetuar todas as ligações, é necessário proceder às configurações de cada componente. Além das configurações normais neste género de exercício (endereços *IP* e protocolo de encaminhamento), existem dois aspetos a realçar. O primeiro é o facto de ter sido necessário desligar o encaminhamento através de *multipath* (vários caminhos)[16], para quando acontecer um pedido este seja encaminhado só para um endereço mesmo que existam várias métricas iguais. O segundo aspeto e talvez o mais importante para o exercício, é a configuração do mesmo endereço *anycast* em duas máquinas diferentes. Foi considerado que o `R5` é o `Nó Inicial` (fornece o endereço *anycast* ao serviço), sendo o endereço definido de acordo com a norma[12] (o endereço atribuído é o `2001:22::0/128`). De seguida, o nó `R6` foi configurado com o endereço *anycast*, finalizando a configurações para averiguar o comportamento da rede (as configurações de todos os encaminhadores encontra-se no anexo B).

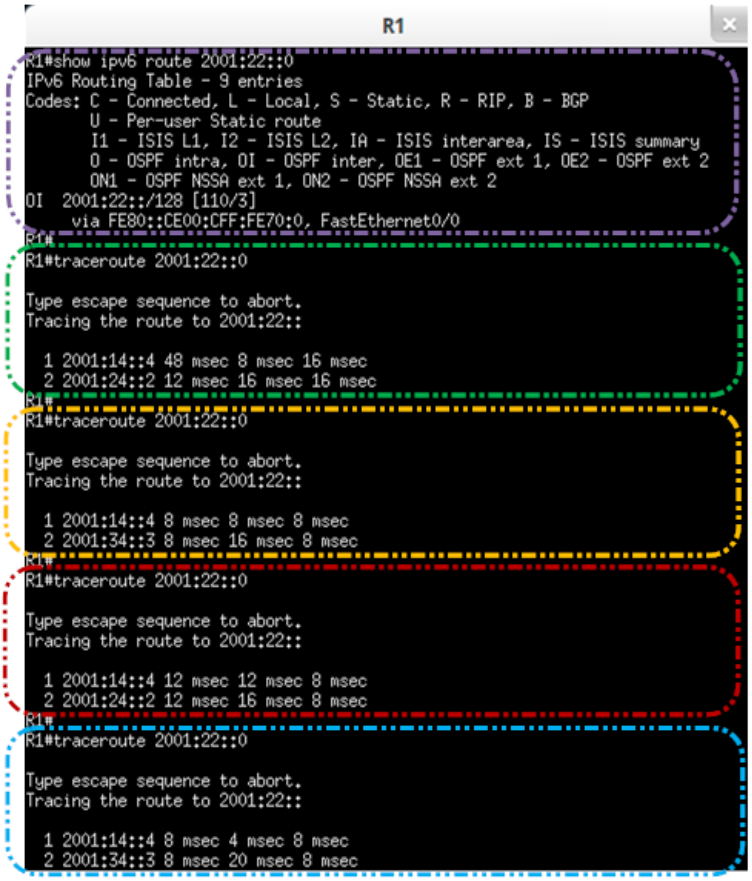


Figura 2.5: Linha de comandos R1.

As figuras 2.5 e 2.6 possuem os teste efetuados para verificar o comportamento da rede implementada, focando-se os testes no funcionamento do *anycast*. O encaminhador `R1` fará o

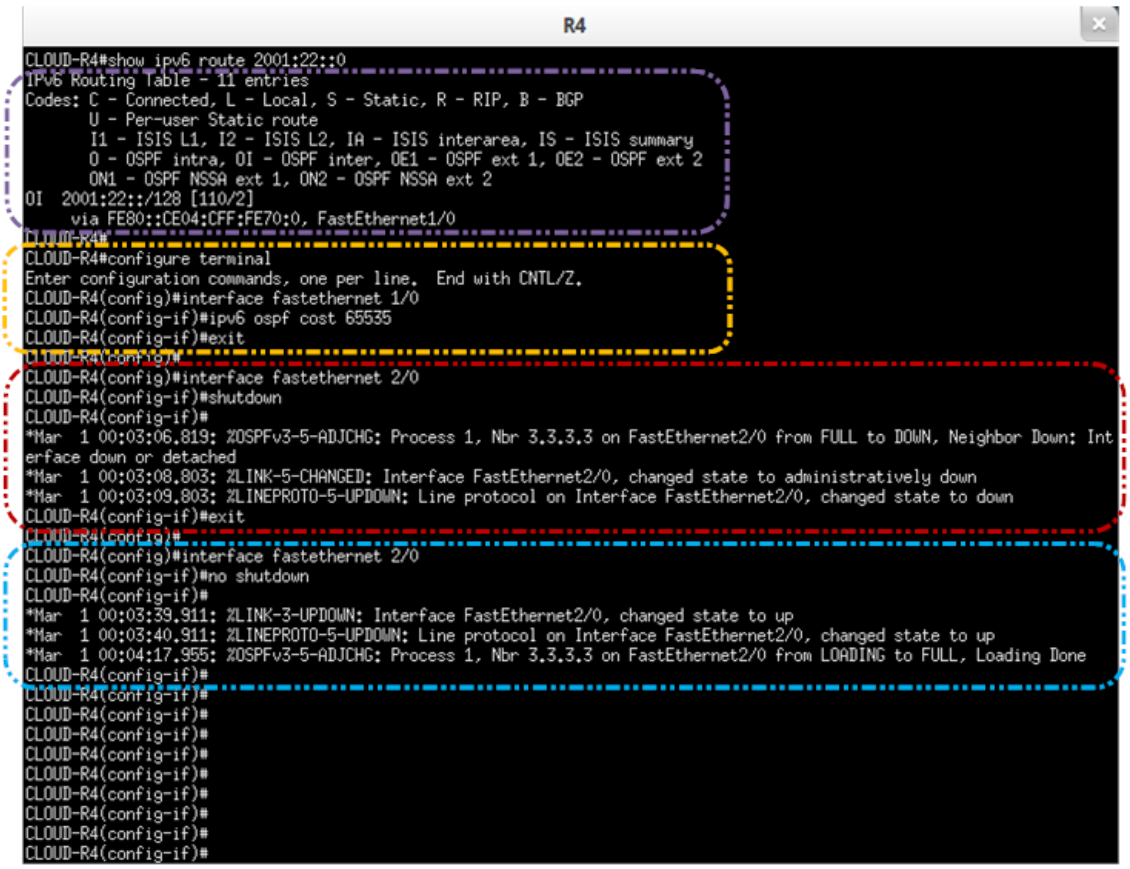


Figura 2.6: Linha de comandos R4.

papel de cliente, ao passo que os encaminhadores `R5` e `R6` farão de servidores. Passado o tempo necessário para que a rede convergisse, o primeiro teste realizado foi com o intuito de verificar o número de entradas na tabela de encaminhamento dos encaminhadores `R1` (na figura 2.5 a lilás) e `R4` (na figura 2.6 a lilás). É possível verificar que cada encaminhador possuí apenas uma entrada na tabela de encaminhamento, ignorando a outra rota. Com o auxílio do comando `traceroute`, foi verificado o caminho percorrido por um pacote desde a origem até ao destino (na figura 2.5 a verde). Como é possível verifica este pedido é enviado até ao `Nó Inicial`. De modo a verificar o comportamento da implementação aquando da degradação da métrica no caminho utilizado, o encaminhador `R4` aumentou o custo da ligação para o máximo permitido (na figura 2.6 a amarelo), na ligação para `R2` e como consequência para `R5`. Com atualização da métrica para aquela zona da rede, `R4` passa então a optar por encaminhar os pacotes para o outro servidor disponível. Quando é enviado um pedido para endereço `2001:22::0`, este passa a ser respondido por `R6` em vez de `R5` (devido ao agravamento da sua métrica), como podemos verificar na figura 2.5 a amarelo. O próximo teste que é necessário efetuar para averiguar as vantagens de optar pelo encaminhamento *anycast*, é na eventualidade da falha de um servidor. Como o encaminhador que está atualmente a responder aos pedidos é o `R6`, é configu-

rada administrativamente a falha para aquele segmento de rede (na figura 2.6 a vermelho). É necessário dar tempo à rede para esta convergir novamente, sendo então realizado novo pedido para o endereço *anycast*. Como é possível verificar na figura 2.5 a vermelho, o pedido volta novamente a ser encaminhado para `R5`. O último teste que é necessário efetuar, é da inclusão de um servidor com melhor métrica que a métrica atual. A simulação desta possibilidade foi efetuada com a ativação administrativamente do segmento de rede previamente desabilitado (na figura 2.6 a azul). O tempo de convergência detetado nesta situação, foi claramente superior ao detetado nas situações passadas. O comportamento da simulação foi o esperado, tendo o pedido sido encaminhado para encaminhador `R6` (na figura 2.5 a azul).

Para concluir esta sub-secção é possível afirmar que hoje em dia a implementação do encaminhamento *anycast* é uma realidade, oferecendo este enumeras vantagens aos administradores de rede. A sua implementação é relativamente simples, oferecendo enorme eficácia e robustez.

2.2.2 Inter-domínio

Como explicado anteriormente, o encaminhamento *anycast* ao nível do inter-domínio não é viável atualmente. Ao contrário da implementação anterior, onde existe encaminhamento *anycast*, quando a implementação pretendida quer agrupar vários SA, não existe nada implementado no *GNS3* que permita efetuar o encaminhamento *anycast*. Ao nível do inter-domínio trata o *anycast* como tráfego *unicast*, sendo necessário a todos os encaminhadores possuírem entradas individuais para todos os serviços do mesmo grupo. Apesar da implementação ter sido realizada, a opção recaiu sobre a sua não colocação no documento, pois nada de surpreendente acrescentou.

2.3 Estado de Arte

O estudo do estado da arte das tecnologias envolvidas nesta dissertação foi realizado com o intuito de fazer um levantamento das soluções atuais, de modo a tentar perceber o porquê de nenhuma destas ter sido ainda adaptadas. Serão explanados os trabalhos disponíveis relacionados com o propósito da dissertação, o encaminhamento *anycast* ao nível global, sendo no final feita uma comparação entre estes. A análise das diferentes abordagens serve para ajudar a ela-

borar uma arquitetura mais consistente, tentando dar um impulso ao encaminhamento *anycast*.

2.3.1 *Global IP Anycast*

Os autores do *Global IP Anycast (GIA)*[17] pretenderam criar uma arquitetura escalável para o encaminhamento *anycast* ao nível global. A proposta destes para o encaminhamento assenta em dois aspetos, é necessário cria o conceito de endereço *anycast* e os domínios só devem ser sobrecarregados com rotas que lhes interessam.

2.3.1.1 Endereçamento

Na opinião dos autores, tratar o encaminhamento *anycast* como *unicast*, não é o método mais eficiente acabando por diminuir as potencialidades da tecnologia. A atribuição de um espaço de endereçamento próprio tornou-se uma necessidade. A figura 2.7 retrata a sintaxe do novo tipo de endereço, sendo os *bits* iniciais o seu principal foco. Este define um `Indicador Anycast` que não é nada mais que um conjunto de *bits* que permite identificar este tipo de tráfego e assim distingui-lo dos outros tipos - *unicast* e *multicast*.

Figura 2.7: Pacote proposto pelos autores do *GIA*.

A sequência de bits '11110' é dada, como por exemplo pelos autores, para ser utilizada como `Indicador Anycast`. No caso de se tratar de um endereço *IPv6* isto não seria um problema devido ao grande número de endereços mas no *IPv4* os domínios onde o prefixo de *unicast* seja menor que 27 *bits* não teriam a possibilidade de alocar nenhum espaço para identificar a que grupo pertence (`Identificador de Grupo`). Apesar desta contrariedade os autores acreditam que endereços com tal prefixo são um caso especial e normalmente estão por detrás de um endereço de rede.

O campo seguinte identifica o domínio que suporta o endereço *anycast*. Este é chamado de `Prefixo Unicast do Domínio Principal` e tem um tamanho variável de acordo com o tamanho do domínio. O domínio que contém o primeiro nó de um grupo, fica estabelecido como o seu domínio principal.

Para finalizar a sintaxe deste novo endereço é definido um campo final. Este dá pelo nome de `Identificador de Grupo` e como o próprio nome indica permite diferenciar diferentes serviços *anycast* dentro do mesmo domínio principal. Assim um domínio pode criar diferentes grupos, prestando diferentes serviços. Este campo possuí também um tamanho variável dependendo diretamente do campo anterior para averiguar o número de grupos possíveis, quanto menor for o prefixo do domínio onde se encontra o serviço *anycast* maior é o número de grupos possíveis.

2.3.1.2 Encaminhamento

Parece ser um conceito óbvio que um encaminhador que utilize múltiplas vezes um serviço, gaste uma maior quantidade de recursos nesse serviço ao invés de outro em que não se encontra interessado. Tendo consciência deste conceito, o *GIA* propõe uma solução baseada na popularidade de um determinado serviço. Um encaminhador integrado num domínio de alto nível, classifica assim um grupo *anycast* num dos três seguintes grupos:

- `Grupo Interno` - é utilizado para quando o domínio possuí pelo menos um membro internamente. Todos os grupos são considerados internos para o seu próprio domínio mas isto não significa que não o possam ser para outros domínios.

- `Grupo Externo Popular` - é um grupo onde não existe localmente nenhum membro, mas onde os seus utilizadores frequentam regularmente esse serviço.

- `Grupo Externo Impopular` - é um grupo onde não existe localmente nenhum membro e não existe qualquer interesse no tráfego gerado por um serviço em questão.

A título de exemplo, não faz qualquer sentido um encaminhador de uma universidade portuguesa (por exemplo a Universidade do Minho) usar os mesmos recursos nas rotas relativas a um qualquer motor de busca (*Google*, *Bing*, *etc.*) e a um controlador de marés (por exemplo Instituto Hidrográfico). Enquanto o motor de busca é acedido minuto a minuto, a página do

Instituto Hidrográfico é raramente (se for alguma vez) consultada. A figura 2.8 permite perceber como se procede à classificação para este exemplo. Do ponto de vista da Universidade do Minho, o repositoriUM seria considerado interno (encontra-se dentro do mesmo domínio), o motor de busca *Google* seria considerado popular e o Instituto Hidrográfico seria considerado impopular.

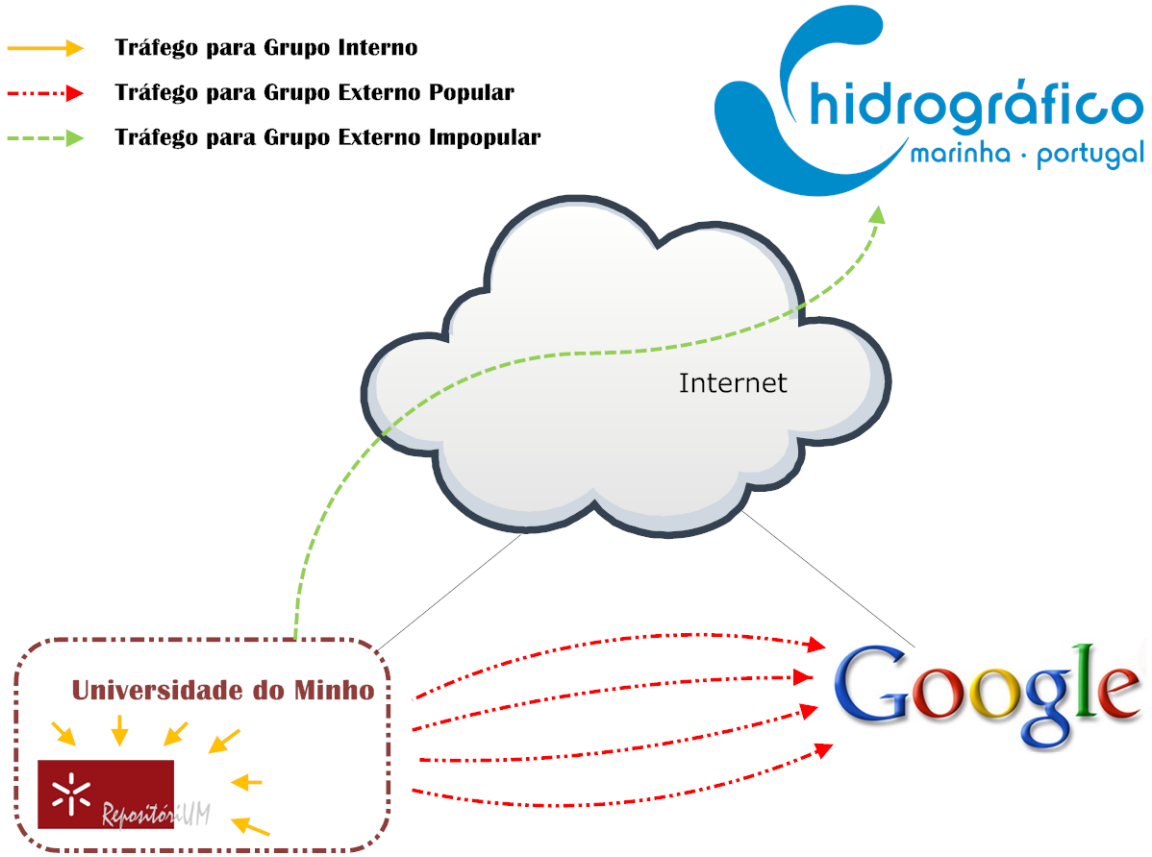


Figura 2.8: Classificação de grupos pelo *GIA*.

O encaminhamento para um grupo interno é executado dentro de um domínio, obrigando os encaminhadores a acrescentar uma entrada à tabela de encaminhamento para cada servidor naquele domínio. A solução permite um encaminhamento simples, mas pode afetar a escalabilidade do sistema. É convicção dos autores que isto não é um problema, pois estes defendem que um domínio é um sistema controlado e fortemente vigiado.

A lógica inerente a esta solução é retirar a necessidade de processamento dos grupos externos impopulares, poupando assim valiosos recursos. A probabilidade do aumento de grupos externos impopulares é bem maior que a probabilidade de aumento de grupos externos populares. Tendo como o objetivo não acrescentar rotas para grupos indesejados, o *GIA* tira partido do endereçamento proposto para proceder ao encaminhamento do tráfego. Como o pacote proposto

é uma concatenação do indicador *anycast* com o prefixo *unicast* do domínio principal mais o identificador do grupo, os encaminhadores podem proceder à desconcatenação do endereço (com o auxílio de uma variável de observação) para obter o prefixo *unicast* onde se encontra o serviço. Na posse do prefixo do domínio principal torna-se possível encaminhar o tráfego de acordo com a tabela de encaminhamento *unicast*. De ressalvar que durante este processo o endereço de destino continua o mesmo, de modo a permitir aos encaminhadores até ao destino efetuarem a mesma operação.

Durante o caminho até ao domínio principal podem acontecer três casos. O primeiro caso ocorre quando o pacote passa pelo domínio que contém um grupo, sendo neste caso o pacote entregue através o protocolo de encaminhamento intra-domínio. O segundo caso acontece quando o pacote atinge um domínio que considera como popular esse endereço, possuindo um melhor caminho para um dos servidores *anycast*. O pacote é então enviado pelo melhor caminho. O terceiro e final caso trata-se da possibilidade de este só encontrar grupos impopulares até ao destino, sendo encaminhado até ao domínio principal (domínio cujo o prefixo está especificado no endereço).

Com o potencial problema causado por grupos impopulares ultrapassado, surge o momento de definir como será otimizado o encaminhamento para grupos populares (com utilizadores interessados). O papel de classificar cada grupo é feito pelo encaminhador fronteira (*border router*) do domínio de alto nível. Periodicamente, estes observam se existem um aumento do fluxo de dados em direção a um grupo *anycast*. Quando detetam que um grupo se tornou popular, iniciam uma procura pelo servidor mais próximo.

O *GIA* propõe um protocolo inter-domínio baseado em perguntas (*queries*) distanciando-se assim do método tradicional do tráfego *unicast* que inundam a rede com os prefixos de todos os domínios na *Internet*. A adição de duas novas mensagens ao protocolo de encaminhamento inter-domínio dominante (*BGP*) foi a forma encontrada pelos autores para aprender rotas com os seus pares (*peers*). A nova mensagem de procura possuí campos semelhantes à já existente mensagem de atualização (*update*) do *BGP*, contendo um campo que guarda todos os sistemas autónomos por onde a mensagem passou, de modo a eliminar a possibilidade de ciclos. Contém ainda um campo para garantir a sua validade (*TTL*) de acordo com o número de saltos que acontecem ao longo da rede *Internet*. Cada mensagem de procura, pode ainda possuir múltiplos endereços *anycast*, diminuindo assim consideravelmente o nível de informação complementar de

controlo (*overhead*).

Com o intuito de descobrir quais são os grupos populares, o encaminhador fronteira (ou então uma segunda máquina adicionada para retirar o trabalho a este) observa o fluxo de pacotes em direção aos serviços *anycast*, procedendo assim à classificação. Existe também a possibilidade de certos domínios serem definidos como populares (por exemplo os servidores *DNS*) devido à sua importância no domínio. O processo de procura por um (ou mais) grupos(s) é desencadeado pelo encaminhador fronteira quando pretende atualizar as suas rotas, denominado encaminhador fronteira de origem (*OBR*). Este envia uma mensagem de pesquisa (valendo-se de mecanismos de difusão pelos seus pares) e aguarda até ao final de um tempo pré-definido para adicionar as novas rotas aprendidas (durante este processo encaminha normalmente, via uma rota padrão).

Dentro de um domínio o único encaminhador a processar uma mensagem de pesquisa é o encaminhador fronteira, não necessitando assim de obrigar outros encaminhadores com informações muito similares a efetuarem o mesmo processamento. A chegada da mensagem desencadeia uma série de passos (consoante a sua validade, $TTL > 0$), começando por iniciar uma revista às suas rotas conhecidas.

Como é possível verificar pela figura 2.9, o primeiro passo a realizar é verificar se a mensagem é originada por um par interno. No caso de ter sido um par interno a enviar a mensagem, não é necessário verificar as suas rotas (pois este já inspecionou as rotas antes de enviar a mensagem) pois estas são iguais. Se a mensagem não tiver sido originada por um par interno é necessário verificar se não se trata de um ciclo, de modo a evitar repetições e com isso desperdício de recursos. Para ser possível apurar os casos cíclicos, a nova mensagem *BGP* de pesquisa possuí um campo onde todos os sistemas autónomos visitados deixem a sua marca (*AS-PATH*). Eliminada a ameaça da repetição é necessário verificar os dois casos onde poderá existir uma resposta, grupos internos e grupos externos populares. No primeiro caso é construída uma mensagem para enviar ao *OBR*, informando-o que possui o grupo interno. No segundo caso são verificadas as rotas aprendidas e caso estas sejam válidas, é concatenado o caminho até ao domínio atual com o caminho necessário para aceder ao serviço *anycast*. Como acontece no primeira caso, a mensagem é diretamente enviada ao *OBR*. Na conjuntura de ainda existirem um ou mais endereços por responder (de notar que apesar da imagem poder iludir e parecer que se trata apenas de uma mensagem, isto só acontece para simplificar o esquema a apresentar) é necessário

verificar o último teste de validade. Começa-se por decrementar o valor do *TTL* (área a pesquisar) e verifica-se se já pesquisou até ao limite pretendido pelo originador da mensagem (no máximo até ao *backbone* da *Internet*). Se a mensagem já atingiu o seu limite é imediatamente descartada, senão é propagada aos seus pares para estes efetuarem os mesmo cálculos. Por fim, é importante referir que a mensagem só é propagada caso o caminho definido na mensagem de pesquisa seja menor que o caminho conhecido pelo encaminhador para o mesmo *OBR*. Isto é realizado com o auxílio de uma tabela para armazenar as pesquisas desencadeadas por outros.

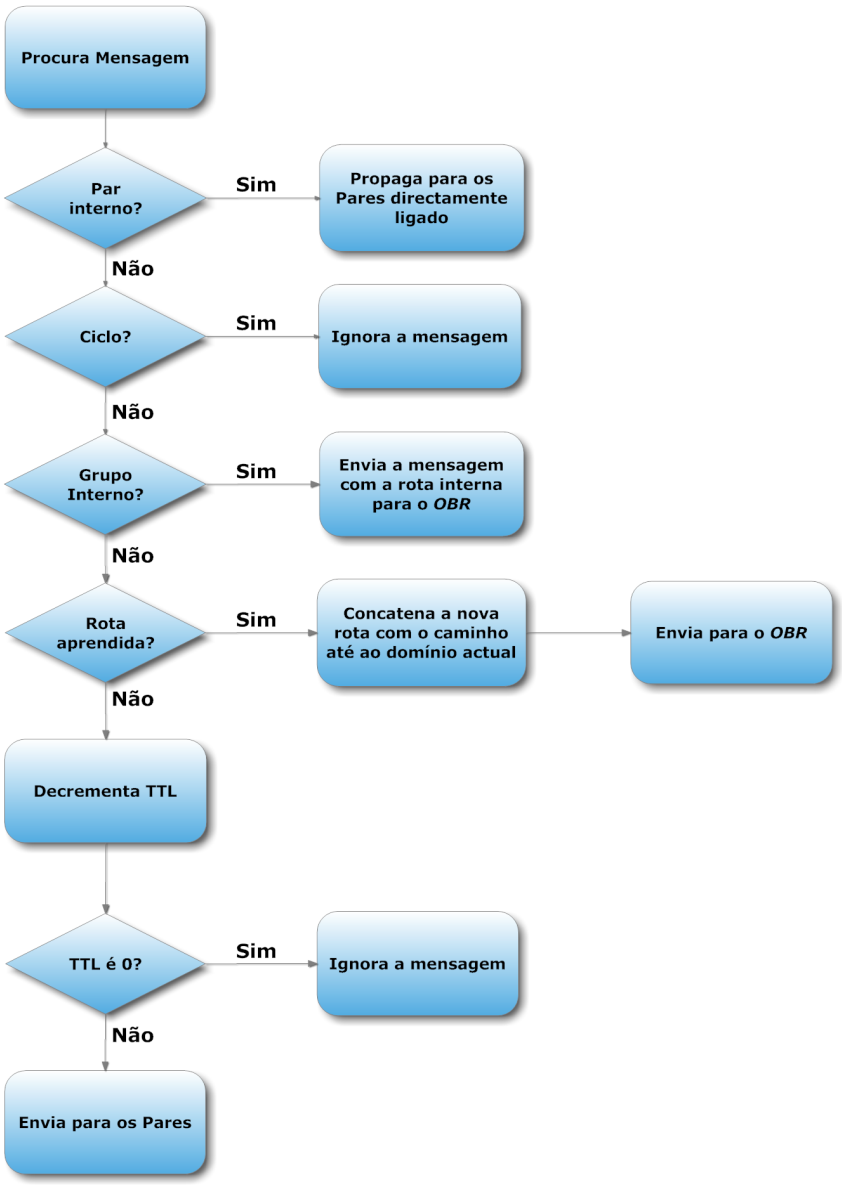


Figura 2.9: Receção e tratamento de uma mensagem *BGP* pelo *GIA*.

Como referido inicialmente quando um *OBR* gera uma mensagem, arranca um temporizador para controlar o tempo máximo para a receção de respostas. Quando o tempo expira, este verifica todas as respostas que recebeu e atualiza-se com as que tiverem melhor métrica. Os grupos

que foram pesquisados e dos quais não foi encontrado um caminho melhor, a sua popularidade é multiplicada por um fator depreciativo, para reduzir a possibilidade de serem incorporados numa futura pesquisa. De acordo com a política interna do domínio, estas novas rotas podem ser difundidas por todo o domínio ou simplesmente manterem-se no encaminhador fronteira. Como na resposta à pesquisa é incluído o endereço *unicast* do encaminhador em questão, este mesmo endereço é utilizado para criar uma ligação virtual (túnel), que consequentemente será utilizada para enviar o resto dos pacotes para aquele serviço *anycast*.

O último aspeto considerado pelos autores incide sobre quando uma rota aprendida deve ser retirada e substituída. Existem quatro casos a serem considerados para julgar a validade de uma rota apreendida. O primeiro aspeto a ter em conta acontece quando há uma falha de conectividade entre o domínio e o membro mais próximo. Neste caso o *OBR* é avisado via *BGP* desta falha, o que o leva a ter uma perceção da falha, o que desencadeia uma nova procura. A segunda possibilidade surge quando um membro *anycast* falha ou abandona o grupo. Como não é detetada automaticamente a situação, o envio de pacotes permanece contínuo até ao encaminhador fronteira onde ocorreu o problema detetar a falha. Este último passa a enviar os pacotes pela melhor rota que conhece e envia uma mensagem a informar da falha, o que levará ao *OBR* a considerar este grupo para uma futura pesquisa. A terceira situação acontece quando um encaminhador fronteira se apercebe que um domínio deixou de ser popular, retirando o suporte em especial para o grupo. Finalmente, a quarta e última possibilidade surge quando existe a necessidade de averiguar se a rota continua a ser a melhor para um determinado grupo, sendo lançada uma nova pesquisa com o valor limite (TTL) menor um salto que o anterior.

2.3.2 Soluções Baseadas em Rede Sobrepostas

Uma rede sobreposta (*overlay network*) é uma rede virtual que existe sobre uma rede subjacente, permitindo oferecer uma solução mais específica sem modificar a rede atual. A topologia da rede sobreposta não necessita de ter uma correspondência com a topologia da rede subjacente. Contrariamente ao encaminhamento *anycast* na sua forma mais simples, a implementação de uma solução baseada em redes sobrepostas permite ocultar de forma dinâmica os grupos *anycast* das tabelas de encaminhamento, reduzindo o peso das tabelas para uma única rota.

De seguida são apresentados dois exemplos baseados numa rede sobreposta, onde o primeiro (*Proxy IP Anycast Service (PIAS)*[18]) é desenvolvido de uma forma mais aprofundada visto

tratar-se de uma abordagem pioneira para o encaminhamento *anycast*. O segundo exemplo (*Architecture for Scalable and Transparent Anycast Services (ASTAS)*[19]) pode ser considerado um melhoramento ao *PIAS*, principalmente na forma como faz o dimensionamento e posicionamento dos serviços na rede virtual.

2.3.2.1 *Proxy IP Anycast Service*

A primeira proposta assenta na criação de uma rede que permita fazer a gestão do tráfego *anycast* de forma eficiente e robusta (importantes características do *anycast*), podendo ser utilizada (a título de exemplo) em comunicações entre pares (*P2P*). Especificamente, são introduzidos um grande número de *proxys anycast* por toda a rede *Internet*. Estes têm a missão de efetuarem a ligação entre o cliente e um servidor de um determinado serviço. Uma outra missão, talvez a mais fundamental, trata-se de efetuar o anúncio das rotas, quer no intra-domínio (*IGP*), quer no inter-domínio (*EGP*).

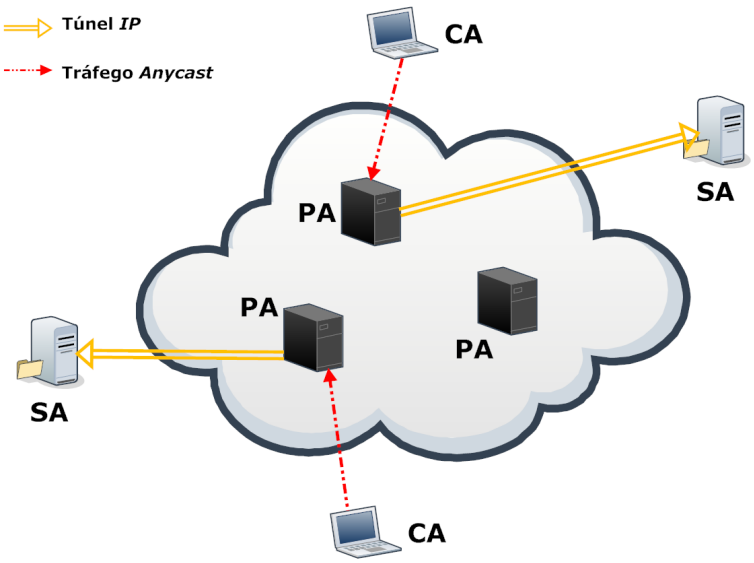
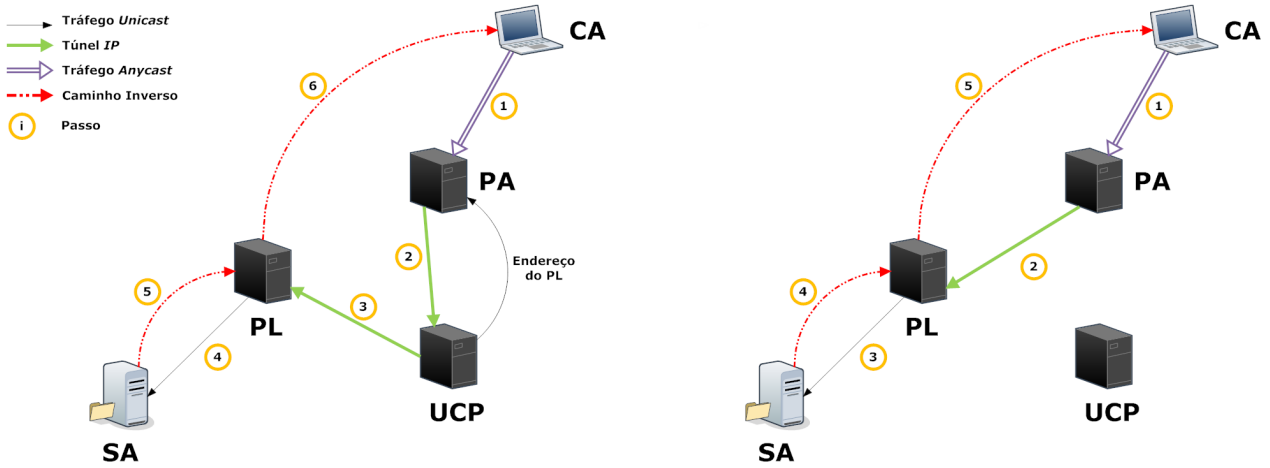


Figura 2.10: Arquitetura do *PIAS*.

A figura 2.10 ilustra sucintamente o funcionamento do *PIAS*. Inicialmente os `Servidores Anycast (SA)` utilizam o encaminhamento nativo *anycast* para descobrir o `Proxy Anycast (PA)` mais próximo do seu local. Quando o `SA` encontra o `PA`, efetua o registo do seu serviço passando a estar disponível para receber pedidos. Com este registo, é criada uma ligação virtual (túnel) entre os dois, sendo o tráfego posteriormente encaminhado por esta ligação. Apesar de poder parecer uma simples implementação, existe um ponto fulcral nesta abordagem relativo ao conhecimento que cada `PA` possuí. Se for pedido a um servidor deste género para ter conhecimento de todos os serviços *anycast*, em conjunto com a manutenção grupos, mais a receção

de pedidos teríamos que estar perante um servidor de portentosas capacidades.

A solução dos autores desta proposta passou pela divisão de tarefas, sendo o PA na realidade um conjunto de três servidores *proxys*. A unidade central do sistema, de nome Unidade Central Proxy (UCP) na figura 2.11, está encarregue de efetuar a supervisão do número de elementos do grupo (número de SA). Um grupo pode ser gerido por várias UCPs de modo a providenciar maior fiabilidade. Como um dos objetivos para esta implementação era que todos os grupos estivessem a poucos saltos de todos os clientes, é necessário que todas as UCPs sejam conhecidas entre elas.



(a) Encaminhamento inicial.

(b) Encaminhamento seguinte.

Figura 2.11: Encaminhamento segundo o *PIAS*

O *proxy* responsável pela ligação aos servidores, designado por Proxy de Ligação (PL), tem a tarefa fundamental de registar o novo serviço junto das UCPs aquando do contacto por parte do SA. Como em tudo o que é relacionado com o *anycast*, o PL escolhida é o mais próxima do SA. No entanto as suas funções não esgotam por aqui, ficando encarregue de efetuar a autenticação dos SA bem como a manutenção da sua disponibilidade.

O restante elemento de gestão, Proxy de admissão (PA), é responsável pela ligação aos clientes e receção dos seus pedidos. Recorrendo aos critérios tradicionais no encaminhamento *anycast* natural, é escolhido o PA mais próximo do Cliente Anycast (CA). Após a receção de um pedido, o PA poderá efetuar uma de duas opções. A primeira situação, representada na figura 2.11(a), retrata o caso de quando não conhece um PL próximo do serviço requisitado, tendo neste caso que perguntar ao UCP. A resposta por parte da UCP contém o PL mais

próximo do serviço (preferencialmente) permitindo assim estabelecer a comunicação. O caso mais simples decorre quando o PA conhece o PL em questão, representado na figura 2.11(b), permitindo assim comunicar diretamente com o LP e iniciar a comunicação imediatamente.

Como na altura da apresentação da proposta existiam severas restrições relativamente ao uso de endereços *anycast* no campo relativo ao endereço de origem, existia uma dificuldade natural a ultrapassar, a tradução de endereços de rede (*NAT*)[20]. Como os SA não poderiam responder com o endereço *anycast*, foi criada uma pequena artimanha para permitir a comunicação. O tráfego viaja então do SA até ao PL, numa primeira fase, contendo o pacote o endereço *unicast* do SA. Aqui é substituído pelo o endereço do grupo *anycast* e com auxílio de uma ligação virtual, é enviado o pacote. A utilização do PL e não do PA não é de todo inocente, pois assim permite-lhe monitorizar a vitalidade da rede. Não fazia sentido utilizar o PA para este tipo de controlo pois um cliente pode estar ligado a múltiplos PA (em intervalos de tempo diferentes) mas só se liga a um PL.

No caso de uma falha num SA, o PL tem o papel de informar o PA para invalidar a entrada que este contém, efetuando de seguida um novo pedido à UCP. Caso o PL falhe, será necessário ao servidor *anycast* proceder a novo registo junto de um novo PL.

2.3.2.2 *Architecture for Scalable and Transparent Anycast Services*

A abordagem anterior tinha como principal objetivo conseguir o encaminhamento global *anycast* baseado numa solução leve (não envolvendo demasiado processamento), ao passo que o foco do *ASTAS* é em atingir a implementação de um serviço escalável e para recursos intensamente acedidos (por exemplo jogos *online* ou *VoIP*).

A introdução do conceito de sessões, aumenta consideravelmente a eficiência na utilização de recursos. Segundo os autores, o conceito de sessões é o primeiro grande contributo do *ASTAS*. O segundo conceito é um conjunto de parâmetros para permitir uma otimização aquando do dimensionamento e posicionamento da rede. Foram considerados três tipos: o custo das infraestruturas, custo operacional da rede e a capacidade de acordo com o posicionamento. Como o contributo mais interessante para o encaminhamento *anycast* é o primeiro, este é o que é abordado de seguida, bem como caracterizadas as suas diferenças para a solução em que esta é

baseada (*PIAS*).

Quando é comparada a infraestrutura do *PIAS* com a do *ASTAS* salta à vista a primeira diferença, o número de nós de suporte à comunicação. No *PIAS* o suporte à comunicação *anycast* é assegurado por três nós (`UCP`, `LP` e `AP`) como explicado anteriormente. No entanto, o *ASTAS* confia num suporte somente com dois nós - o *proxy* de apoio aos clientes (`CP`) e o *proxy* de apoio aos servidores (`SP`).

Os objetivos comuns e as diferenças entre as abordagens, são dois dos aspetos a considerar quando se realiza uma comparação. As melhorias apresentadas por ambas as abordagens em relação ao encaminhamento *anycast* original são principalmente duas. A primeira melhoria concentra-se na facilidade que um servidor de um determinado serviço encontra quando tenta fazer a sua inclusão ou exclusão. Nestas novas abordagens, a rede sobreposta é quem tem em consideração a gestão dos grupos, não sobrecarregando assim as tabelas de encaminhamento. Como a lógica inerente ao encaminhamento *anycast* é da escolha do serviço mais próximo, isto nem sempre é o mais correto, pois um servidor mais próximo pode estar mais sobrecarregado. De modo a combater esta limitação, ambas as abordagens permitem escolher o provedor de serviço de acordo com outros critérios, tais como a carga na rede ou a velocidade de conexão.

Revistos os aspetos similares nas duas abordagens, chega agora a altura de destacar a principal diferença. Um dos objetivos do *PIAS* passa por uma solução leve (como referido anteriormente) com pouca troca de informação entres os servidores de suporte. Isto faz com que seja talhado para comunicações rápidas, devido à pouca informação complementar de controlo (*overhead*). Percebendo isto, os autores do *ASTAS* aproveitaram a dificuldade em manter sessões mais longas e propuseram uma nova solução. No *ASTAS* sempre que um cliente faz um pedido, é iniciada uma nova sessão para garantir o melhor serviço possível (no *PIAS* só ocorre uma troca de sessão quando ocorre uma falha no sistema). Com esta inovação foi criada uma plataforma que proporciona uma grande disponibilidade para serviços críticos, não obstante à clara desvantagem do elevado consumo de recursos por sessão, pois o `CP` tem que manter sempre a sessão aberta.

Para concluir surge uma última comparação sobre como é feita a resposta a um pedido, onde estes têm abordagens muito diferentes. Para responder a um pedido, o *ASTAS* utiliza uma ligação direta entre os servidores *anycast* e o `SP`, em vez de obrigar à tradução de endereços da

rede como acontece no *PIAS*. De modo a concretizar este objetivo é necessário a configuração do túnel por parte dos servidores que queiram prestar um serviço. Com esta funcionalidade, surge a possibilidade de proporcionar qualidade de serviço para encaminhamento *anycast*, cabendo ao (SP) ter em conta a rede e os recursos disponíveis.

2.3.3 Soluções baseadas em protocolos *multicast*

Devido a algumas semelhanças entre o tráfego *multicast* e o *anycast*, é relevante efetuar uma análise e adaptação dos principais protocolos de encaminhamento *multicast*. Nesse sentido, foi desenvolvido por um conjunto de autores um trabalho primeiramente teórico[21] explorando as suas semelhanças e diferenças, bem como possíveis alterações a três diferentes protocolos *multicast* - um baseado em vetores de distância (*DVMRP*[22]), um baseado em estado de ligação (*MOSPF*[23]) e um último mais direcionado a grupos dispersos (*PIM-SM*[24]). O segundo trabalho[25] foca-se na última ideia apresentada no anterior documento, adaptando o *PIM-SM* ao *anycast*.

Existem três importantes considerações a ter em conta:

1. **Endereçamento** - como referido anteriormente, na norma *IPv6* não existe diferenciação entre o endereçamento *anycast* e *unicast*, sendo este indistinguíveis quando comparados. Alguns autores[17] argumentam que para retirar partido do encaminhamento *anycast*, é necessário utilizar um novo espaço de endereçamento. Tendo os encaminhadores a capacidade de efetuar esta distinção, é-lhes possível encaminhá-los de acordo com o seu tipo. As seguintes propostas, optaram por manter o mesmo endereçamento que o aconselhado na norma.

2. **Implementação** - é irreal pensar que a implementação de um novo protocolo é instantânea. Para a implementação total de uma nova proposta, seria necessário substituir uma imensidão de encaminhadores, o que em termos de custos seria incomportável. Com isto em mente, os autores propõe numa implementação gradual da sua proposta, podendo esta funcionar com a alteração de somente um encaminhador. É crença destes, que apenas um novo encaminhador suportando a sua tecnologia, traz várias melhorias caso colocado entre o cliente e o servidor *anycast*.

3. **Protocolo** - os autores das próximas propostas, estão convictos que a adaptação dos

protocolos *multicast* à realidade traz enumeras vantagens, devido às suas semelhanças, reduzindo em muito a sua complexidade na altura de implementação.

2.3.3.1 Arquitetura de encaminhamento

Os autores pretendem uma implementação gradual, podendo esta ser considerada uma rede sobreposta sobre a rede tradicional. A rede sobreposta é composta pelo conjunto de encaminhadores capazes de identificar e encaminhar o tráfego *anycast* recorrendo ao protocolo desejado, denominados encaminhadores *anycast* (EA). Na rede *anycast*, os nós podem não estar fisicamente ligados, mas conectados por diferentes ligações lógicas ponto-a-ponto (caminhos virtuais, túneis ou encapsulação), como podemos verificar na imagem seguinte (figura 2.12).

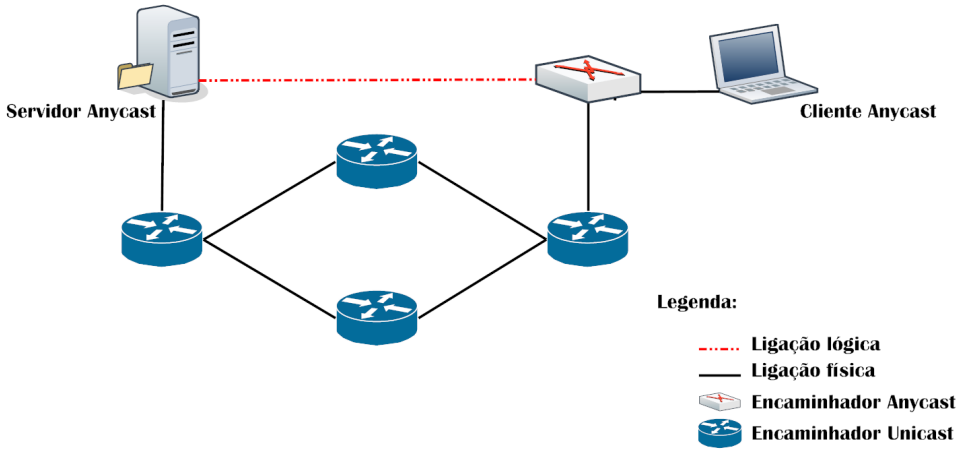


Figura 2.12: Esquema de ligações

Um EA possui uma tabela extra em relação a um encaminhador normal, uma tabela de encaminhamento *anycast*, que lhe permite direcionar adequadamente e eficazmente este tipo de tráfego. Quando um EA recebe um pacote, a primeira tabela que este inspeciona é a de encaminhamento *anycast*, e verifica se possui um caminho para o campo destino do pacote. Em caso afirmativo, trata-o como tráfego *anycast* e envia para o próximo EA, de acordo com a informação armazenada na tabela. No caso de não possuir uma entrada nesta nova tabela, envia o tráfego recorrendo ao encaminhamento tradicional *unicast*.

A figura 2.13 ilustra um exemplo do funcionamento do encaminhamento *anycast* valendo-se da introdução de EA. A atribuição de um endereço a um grupo *anycast*, tem por base o `Nó inicial` (*Seed Node*), ficando o grupo associado ao endereço *unicast* do sistema terminal em questão. A figura 2.13 fornece uma perspetiva geral de como funciona o encaminhamento e

representa a melhoria que este consegue introduzir na rede. Podemos observar que existem dois servidores *anycast* (A1 e A2) e que estes possuem o mesmo endereço *anycast*, mesmo que o servidor A2 pertença a uma rede diferente. O endereço do grupo *anycast* (3ffe:5::5) é o mesmo que o endereço *unicast* do servidor A1. Existem dois clientes (C1 e C2), de modo a demonstrar os cenários possíveis aquando da realização de um pedido. O primeiro cenário, onde o C1 se encontra, acontece quando entre ele e o Nó inicial existe um EA. Quando o pacote chega ao EA, é verificado se existe uma entrada na tabela de encaminhamento *anycast* e passa a ser tratado como tráfego *anycast*, sendo entregue ao servidor mais próximo (A2). O segundo cenário é o pior possível, pois o pedido viaja até ao Nó inicial (quando este não é o melhor caminho). Uma vez que o pedido não passa por nenhum EA, nunca é tratado como tráfego *anycast*, sendo então entregue via encaminhamento *unicast* à rede do Nó inicial. Considerando o melhor caminho, o caminho com menor número de saltos, podemos constatar que para o C2 o melhor servidor também é *A2*.

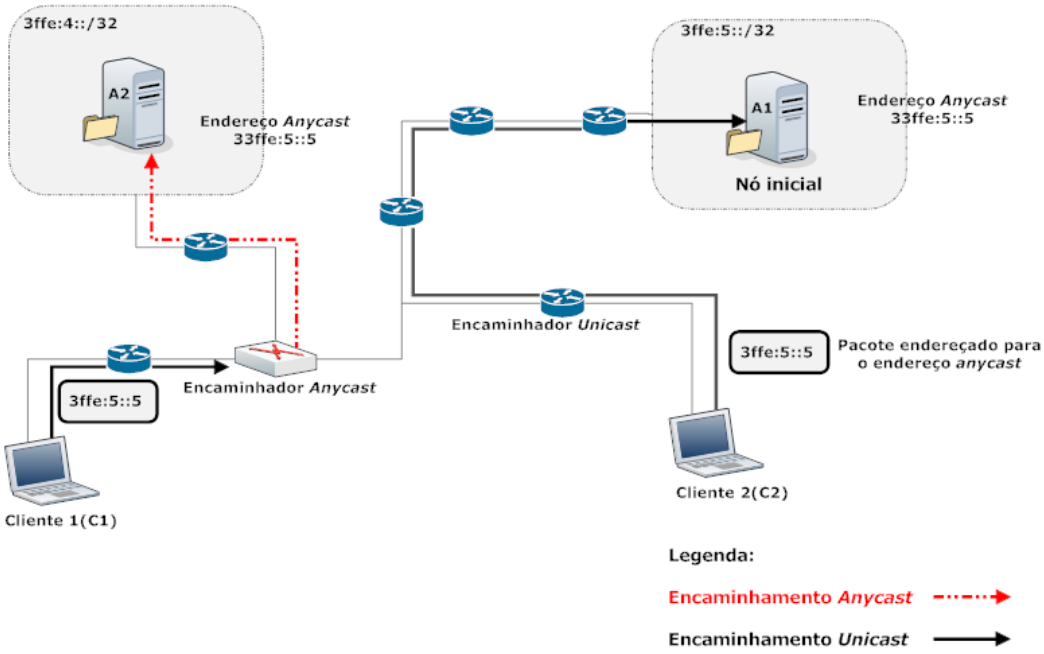


Figura 2.13: Exemplo de encaminhamento *anycast* sobre a rede *unicast*

As vantagens de uma implementação global, também são visíveis na figura 2.13. É facilmente percetível que todo o tráfego que passa pelo EA é entregue a um servidor mais próximo e que quanto maior for o número de encaminhadores implementados, maior será a eficácia no encaminhamento. Substituindo o encaminhador tradicional mais próximo do segundo cliente por um EA, verifica-se que os seus pedidos passariam a ser entregues ao servidor mais próximo deste na realidade (A2).

2.3.3.2 Adaptação de protocolos *multicast*

Definido o funcionamento geral da proposta, é agora necessário explicar como decorreria o encaminhamento *anycast*. Antes de começar a pensar nas alterações aos protocolos *multicast*, é necessário assimilar que uma comunicação *anycast* ocorre de um cliente para um servidor, ao passo que no *multicast* uma fonte (ou um conjunto de fontes), pode ter vários clientes interessados. Isto faz com que existam ligações de um (ou muitos) para muitos. Outro aspeto a considerar é que a variação de elementos do grupo *multicast* é maior que no *anycast*. Um endereço *anycast* é atribuído a um servidor, devendo este manter-se por várias comunicações, ao ponto que no *multicast* um endereço dura para uma comunicação.

A adaptação dos protocolos *multicast* à metodologia *anycast*, inicia-se pela especificação de como os nós efetuam a sua associação inicial. Como no tráfego *multicast*, é necessário permitir aos nós a capacidade para notificar o `EA` mais próximo, da sua junção ou abandono do grupo. Para conseguir este efeito, é redesenhada a função de descoberta de nós *multicast* implementada no *IPv6*, caracterizada pelo acrónimo *Multicast Listener Discovery (MLD)*[26]. É então proposto pelos autores o acrónimo *Anycast Receiver Discovery (ARD)*, que pode ser traduzido como descobridor de servidores *anycast*. Para esta proposta funcionar corretamente, seria necessário considerar que todos os encaminhadores eram do tipo `EA`, o que é irrealista. Umas das formas de contornar esta problemática seria implementar somente os servidores próximos deste novo tipo de encaminhadores, para garantir a sua estabilidade, pois quando é gerada uma mensagem de descoberta esta é enviada para o endereço local (`FF02::2`), tal como no *MLD*. Deste modo, sempre que um servidor se quisesse juntar/abandonar a um grupo, enviaria um mensagem `ARD` a informar o `EA`.

Sendo a associação inicial dos nós independente do protocolo, a construção e manutenção das tabelas de encaminhamento varia consoante o protocolo escolhido. Os autores propõem três diferentes protocolos:

1. *Distance Vector Anycast Routing Protocol (DVARP)* - baseado no protocolo *multicast DVMRP*[22].

2. *Anycast Open Shortest Path First (AOSPF)* - baseado no protocolo *multicast MOSPF*[23].

3. *Protocol Independent Anycast - Sparse Mode (PIA-SM)* - baseado no protocolo *multicast PIM-SM*[24].

Construção e manutenção das tabelas de encaminhamento *DVRMP*

Consoante o algoritmo a ser implementado, a estratégia para construção e manutenção das tabelas de encaminhamento varia. O primeiro a ser analisado trata-se do algoritmo baseado em vetores de distância, o *DVRMP*. No encaminhamento *multicast* é habitual que a constituição do grupo varie bastante ao longo do tempo, é bastante difícil prever quais as localizações onde é suposto um pacote ser entregue, o que torna uma abordagem por inundação apelativa. Apesar de aumentarem consideravelmente a carga na rede com informação complementar de controlo, conseguem obter uma imagem clara do grupo. Visto que a constituição de um grupo *anycast* é consideravelmente mais estável, o redesenho deste protocolo passa por uma troca de informação periódica, em substituição da inundação constante da rede aquando de uma alteração.

Na imagem seguinte (figura 2.14), é possível verificar como funcionaria este novo protocolo. Quando um `EA` recebe um mensagem `ARD` a informá-lo do novo endereço, atualiza a sua tabela de encaminhamento. Periodicamente, os encaminhadores adjacentes trocam as suas rotas, sendo atualizados com o novo endereço. Quando estes recebem esta atualização, acrescentam a informação às suas tabelas de encaminhamento.

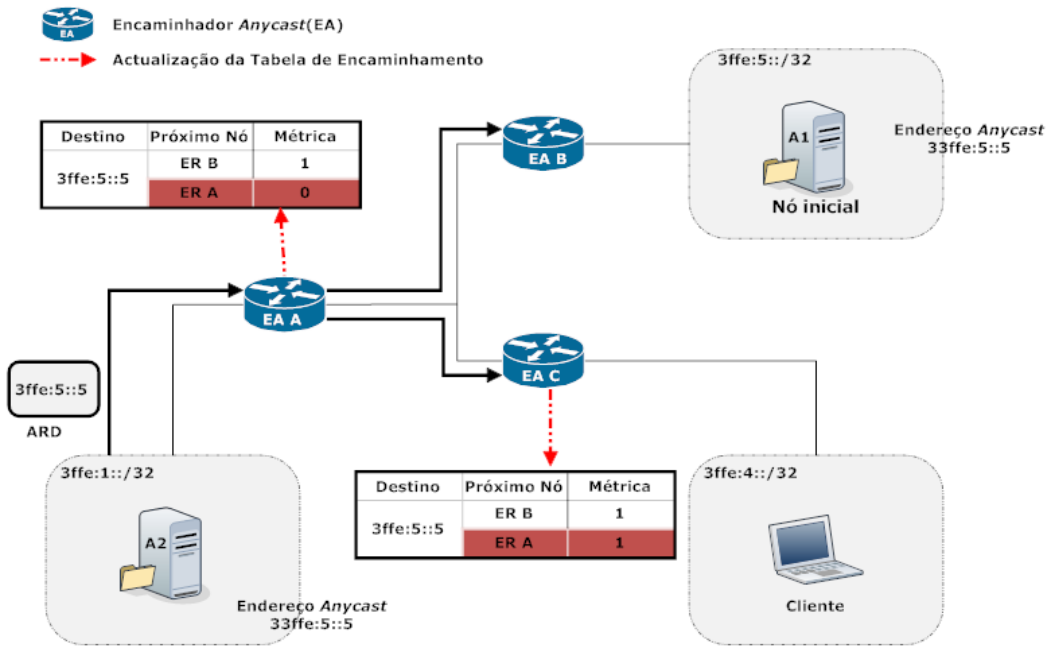


Figura 2.14: Exemplo do funcionamento do *DVARP*

Construção e manutenção das tabelas de encaminhamento *MOSPF*

O *MOSPF*, conhecido algoritmo *multicast*, é o seguinte algoritmo a adaptar ao universo *anycast*. Este é baseado no estados das ligações, isto é, a troca de informação acontece sempre que existe uma mudança na topologia, sendo uma abordagem interessante para o *anycast*. Com a normal estabilidade das ligações *anycast*, somente seria necessário trocar informação quando um servidor fosse alterado (quer fosse acrescentado ou retirado).

O seu funcionamento é facilmente descrito. Sempre que exista uma alteração de estado num servidor, o seu EA é informado, criando/atualizando uma entrada na base de dados que armazena as ligações. Com esta alteração, é desencadeado entre os encaminhadores uma troca de mensagens, enviando cada um destes os seus estados de ligação, ao longo de várias iterações. Com a receção destas mensagens, são atualizadas as ligações, correndo depois o algoritmo de *Djikstra* para criar uma árvore de caminhos mais curtos. Finalmente, todos os encaminhadores atualizam as suas tabelas de encaminhamento de acordo com a informação calculada.

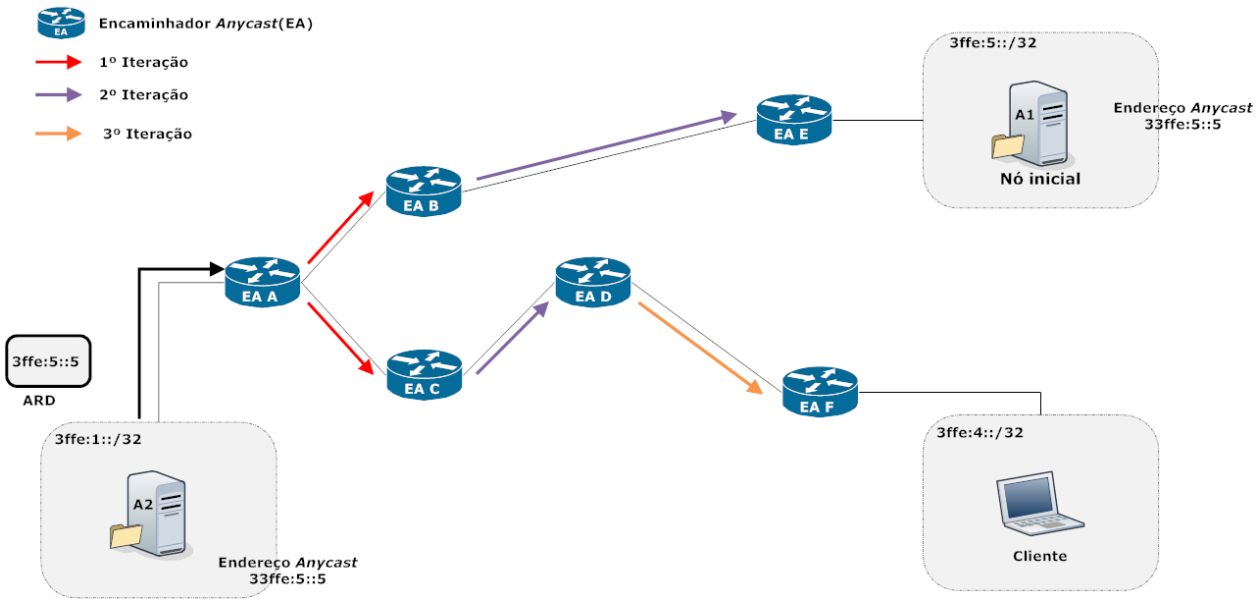


Figura 2.15: Exemplo do funcionamento do *MASPF*

Construção e manutenção das tabelas de encaminhamento *PIA-SM*

O *PIM-SM* é considerado independente do protocolo *unicast*, pois não possui nenhum protocolo específico para descobrir a topologia, recorrendo a qualquer um dos protocolos de encaminhamento *unicast* para essa descoberta. É ideal para grupos dispersos ao longo da *Internet*, enquadrando-se no panorama *anycast*. Constrói árvores unidirecionais, centradas num ponto

de encontro, normalmente designado por *Rendezvouz Point* (RP).

O *RP* é responsável pela gestão do grupo, servindo de intermediário entre clientes e o servidores. Existem duas possíveis mensagens a serem trocadas entre os EA e o RP, a PIA-Join para a junção ao grupo e a PIA-Prune para o abandono do grupo. Quando um EA recebe uma destas mensagens, cria/exclui a ligação para o membro, enviado mensagem para os seus vizinhos como podemos verificar na figura 2.16. O ponto final destas mensagens é sempre o RP, que quando recebe uma mensagem atualiza a sua tabela de encaminhamento. A atualização pode passar pela criação/exclusão de um nova entrada ou então pela adição a uma entrada anteriormente definida, permitindo assim múltiplos caminhos.

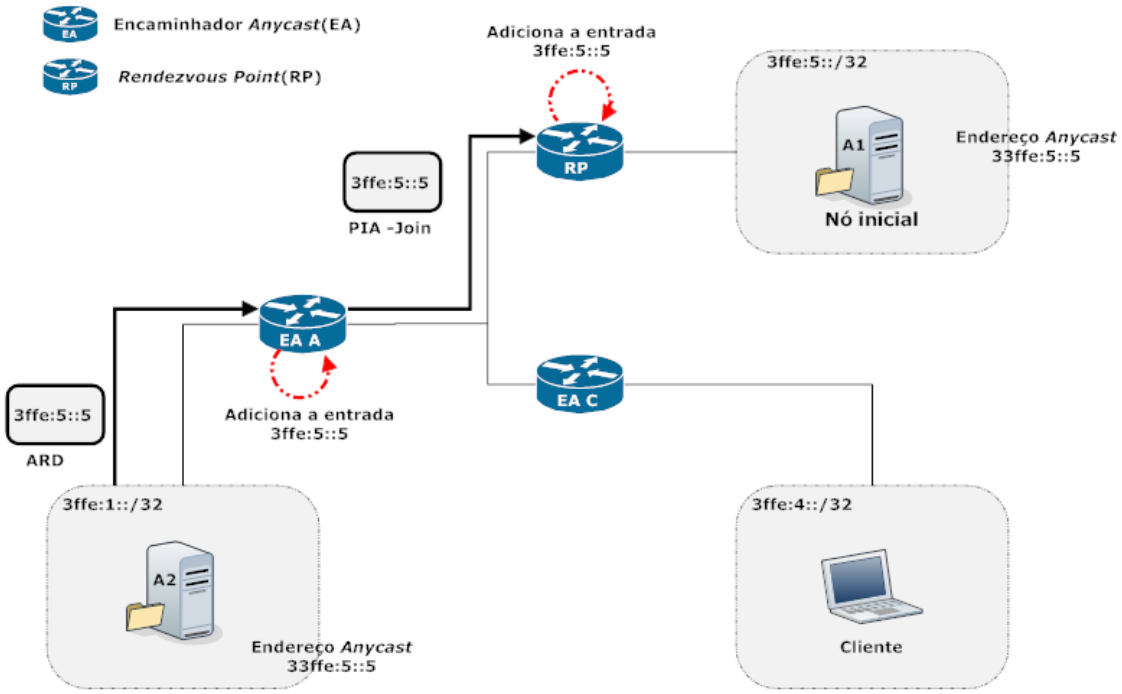


Figura 2.16: Exemplo do funcionamento do *PIA-SM*

2.3.3.3 Desenho do Protocolo de Encaminhamento *Anycast* para *IPv6*: *PIA-SM*

Devido à sua especificação, o *PIM-SM* foi otimizado para redes com nós dispersamente distribuídos ao longo da rede. Este aspeto é uma grande vantagem para a implementação de um protocolo *anycast*, principalmente quando este é habitualmente utilizado em tarefas com balanceamento de carga. Com a adaptação deste à realidade *anycast*, surge então o *PIA-SM*[25], que combina os mecanismos *multicast* para a gestão dos servidores e os mecanismos do *unicast* para encaminhar o tráfego (pacotes). Continuando o trabalho desenvolvido anteriormente[21],

os autores escolhem o *PIA-SM*, sendo este sujeito a uma análise mais profunda sobre o seu funcionamento. Doravante os encaminhadores denominados `PIA`, são os únicos capazes de suportar este o protocolo *PIA-SM*.

Descoberta de Encaminhadores *PIA* Vizinhos

A descoberta dos vizinhos, é talvez o passo fulcral desta implementação, pois permite aos encaminhadores `PIA` identificar os caminhos possíveis na sua topologia. Todos os encaminhadores `PIA` trocam mensagens entre si, denominadas por `PIA Hello`, com o objetivo de se identificarem (a laranja na figura 2.17). A figura 2.17 representa essa troca de informação, permitindo verificar que todos o encaminhadores `PIA` trocam entre si as mensagens `PIA Hello`. Com todos os vizinhos identificados, é eleito para cada segmento de rede um encaminhador responsável (*DR*), cuja a principal função é o envio de mensagens periódicas de aviso para o *RP*.

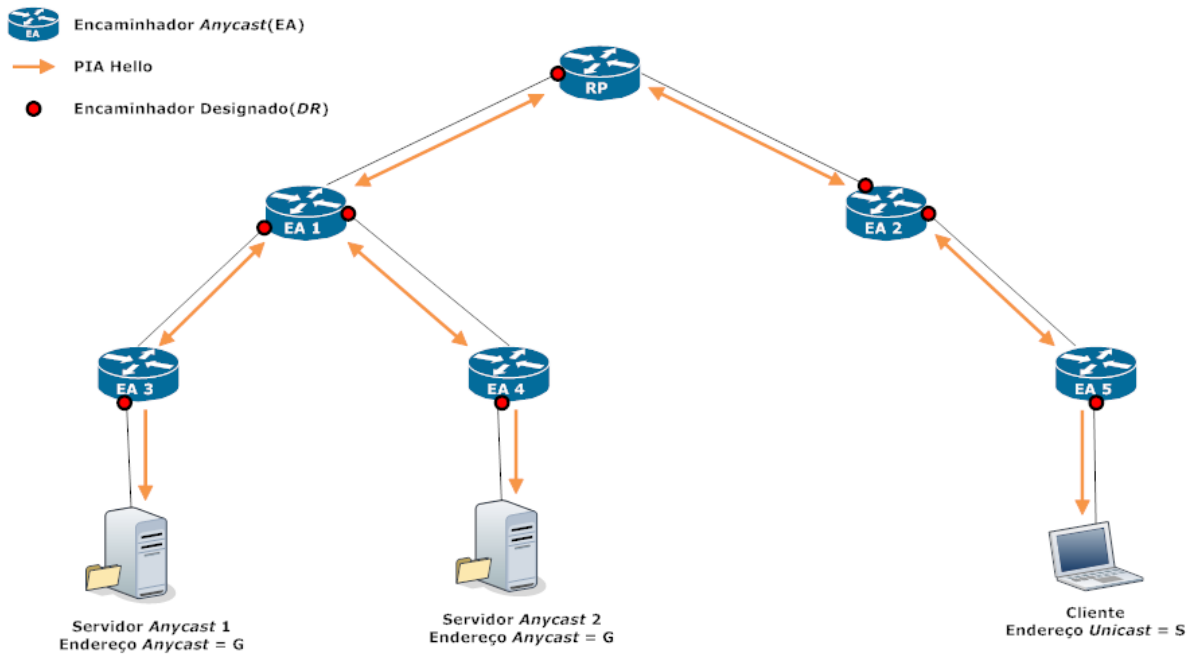


Figura 2.17: Descoberta de vizinhos `PIA`

Criação da Árvore de Caminhos Inversos

Como o caminho é calculado desde os servidores até ao *RP*, esta é conhecida por árvore de caminhos inversos. Os encaminhadores `PIA` intermédios com múltiplas possibilidades, escolhem a melhor e enviam a mensagem para o próximo encaminhador até ao *RP*. A criação desta

árvore faz-se em dois níveis: com mensagens ARD e com mensagens PIA Join.

Uma mensagem é gerada por um servidor *anycast*, aquando da tentativa de se juntar/abandonar um grupo. Esta mensagem é entregue sempre ao encaminhador PIA mais próximo, sendo composta por o endereço *anycast* e a sua métrica. As mensagens PIA Join são sempre iniciadas por encaminhadores PIA (o *DR* no segmento de rede) que recebem a informação de uma alteração na sua rede local (mensagens ARD). O caminho destas mensagens é sempre percorrido em direção ao *RP*, contendo o endereço *anycast* e a sua métrica. Quando o encaminhador PIA (o próximo em direção ao *RP*) recebe uma mensagem PIA Join, verifica as suas tabelas de encaminhamento e atualiza (ou acrescenta) no caso de uma melhor métrica. Na figura 2.18, o EA 1 recebe duas mensagens PIA Join dos encaminhadores EA 3 e EA 4, acrescentado-as à sua tabela de encaminhamento. Quando chega a altura de propagar a mensagem, este compara as entradas que possui na tabela de encaminhamento e escolhe a de melhor métrica (menor custo). A métrica escolhida é a que ele aprendeu através do EA 3 (a verde na figura 2.18). Como este procedimento é repetido desde o encaminhador PIA mais próximo do servidor até ao *RP*, este último fica com o melhor caminho para o respetivo endereço *anycast*.

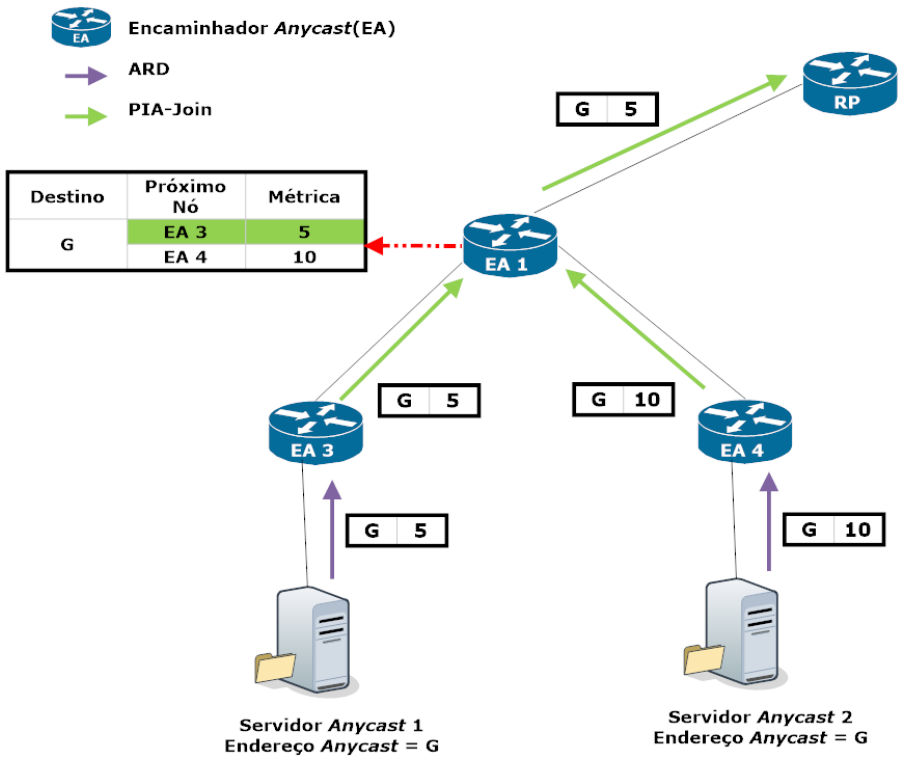


Figura 2.18: Criação da árvores de caminhos inversos

Encaminhamento dos Pacotes *Anycast*

Concluído o processo da escolha do melhor caminho para o grupo *anycast*, chega agora a hora de explicar como funcionária realmente a comunicação. A figura 2.19, ilustra o comportamento da rede quando um cliente pretende efetuar um pedido. Inicialmente o `Cliente`, envia um pacote *anycast* preenchido com o endereço *anycast* no campo de destino. Quando o primeiro encaminhador `PIA` (na figura 2.19, `EA 5`) verifica o endereço *unicast* do *RP* para aquele grupo. Na posse do endereço *unicast*, encapsula o pacote *anycast*, permitindo que seja encaminhado através de encaminhamento *unicast* (setas roxas) até ao *RP* (isto faz com que somente seja necessário um encaminhador próximo do cliente). Este novo tipo de pacote é designado por `PIA Capsule`.

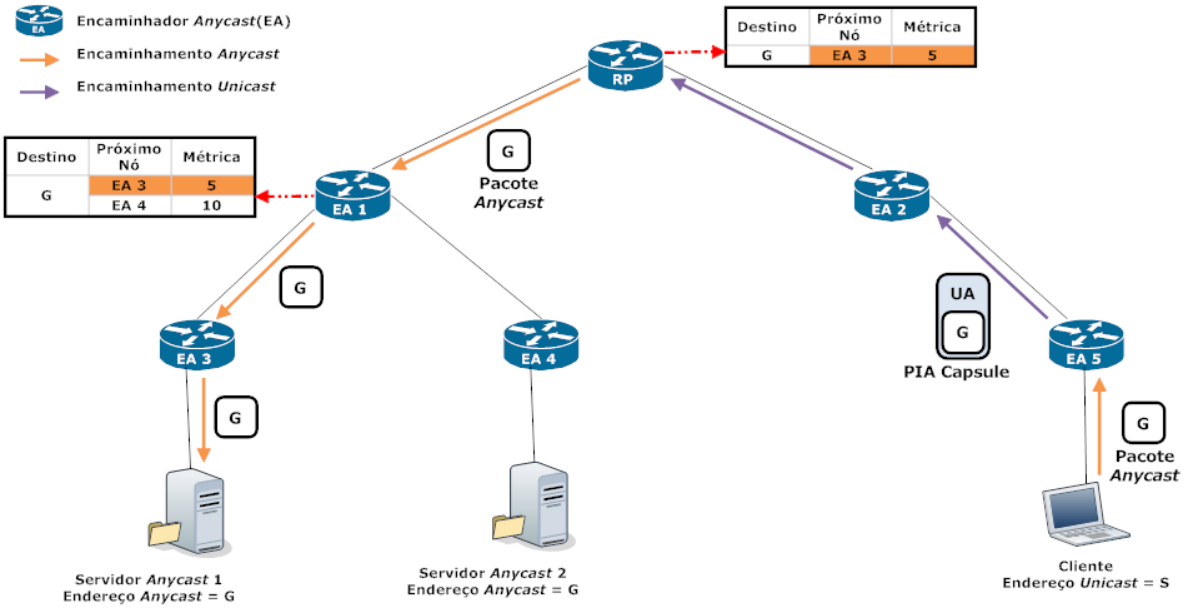


Figura 2.19: Encaminhamento dos pacotes *anycast* pelo *PIA-SM*

Quando o pacote `PIA Capsule` finalmente é entregue ao *RP*, este efetua o seu desencapsulamento, obtendo o pedido original do cliente. Como explicado anteriormente, todos os encaminhadores entre o servidor e o *RP* possuem entradas para o endereço *anycast* (endereço `G` na figura 2.19), procedendo à encaminhamento normal do pedido através da melhor métrica. Na figura 2.19 o pacote viaja do *RP* até ao `Servidor Anycast 1`, com o auxilio de `EA 1` e `EA 3`.

A capacidade para comutar para uma árvore centrada no cliente não é considera pelos autores muito interessante, pois, na opinião dos autores do *PIA*, é pouco provável que um cliente

efetue muitos pedidos (ao contrário do que acontece no *multicast*, onde existem grandes trocas de mensagens). No caso de uma transmissão contínua, o melhor procedimento seria utilizar o endereço *anycast* só para descobrir o servidor mais próximo do cliente e de seguida utilizar o endereço *unicast* para a transferência de dados.

Balanceamento de carga no *PIA-SM*

O protocolo *PIA-SM* não contempla a hipótese de receber pedidos de mais de um cliente, mantendo sempre o mesmo servidor como o melhor destino. Se existirem muitos clientes a efetuarem pedidos, o melhor servidor fica sobrecarregado com a carga total, estando os outros servidores sem receberem pedidos. Para transpor este problema, uma nova abordagem surgiu[27], permitindo ao *PIA-SM* balancear os seus pedidos.

A nova abordagem introduz um novo campo na tabela do *RP*, denominado `BMF`, que é uma combinação entre a métrica anunciada e o estado do servidor (*livre* ou *ocupado*). O *RP* ao receber um pedido, desencapsula o pacote, enviando para o servidor com melhor métrica no campo `BMF`. Adicionalmente, atualiza essa métrica, acrescentado à métrica mais alta, 1.

A figura 2.20 contém um cenário com dois clientes e dois servidores. A métrica anunciada pelo `Servidor Anycast 1` é de 22 e pelo `Servidor Anycast 2` de 20. O *RP* ao receber esse valor, atualiza as suas entradas (métrica e `BMF`). Quando o `Cliente Anycast 1` faz um pedido para o grupo, envia a mensagem para o *RP*. Este procura na sua tabela o melhor servidor e verifica que se trata do `Servidor Anycast 2`, reenviando o pedido (a laranja na figura 2.20). Por fim, atualiza o valor do campo `BMF` para aquele servidor, somando 22 (métrica maior atualmente) com 1. O `Cliente Anycast 2` decide também efetuar um pedido, enviando o seu pacote para o *RP*. Após a consulta da tabela atualizada, o *RP* verifica que o servidor com melhor valor no campo `BMF` é o `Servidor Anycast 1`, reenviando o pedido. Mais uma vez, o *RP* recalcula o valor do campo `BMF` para o `Servidor Anycast 1` e atualiza-o para 24.

A abordagem proposta permite ao protocolo *PIA-SM* conseguir fazer balanceamento de carga, mas obriga a que o *RP* execute mais tarefas para manter atualizado o valor do campo `BMF`. Num protocolo como este, o *RP* é um ponto crítico, podendo esta alteração, aumentar consideravelmente a carga do *RP*.

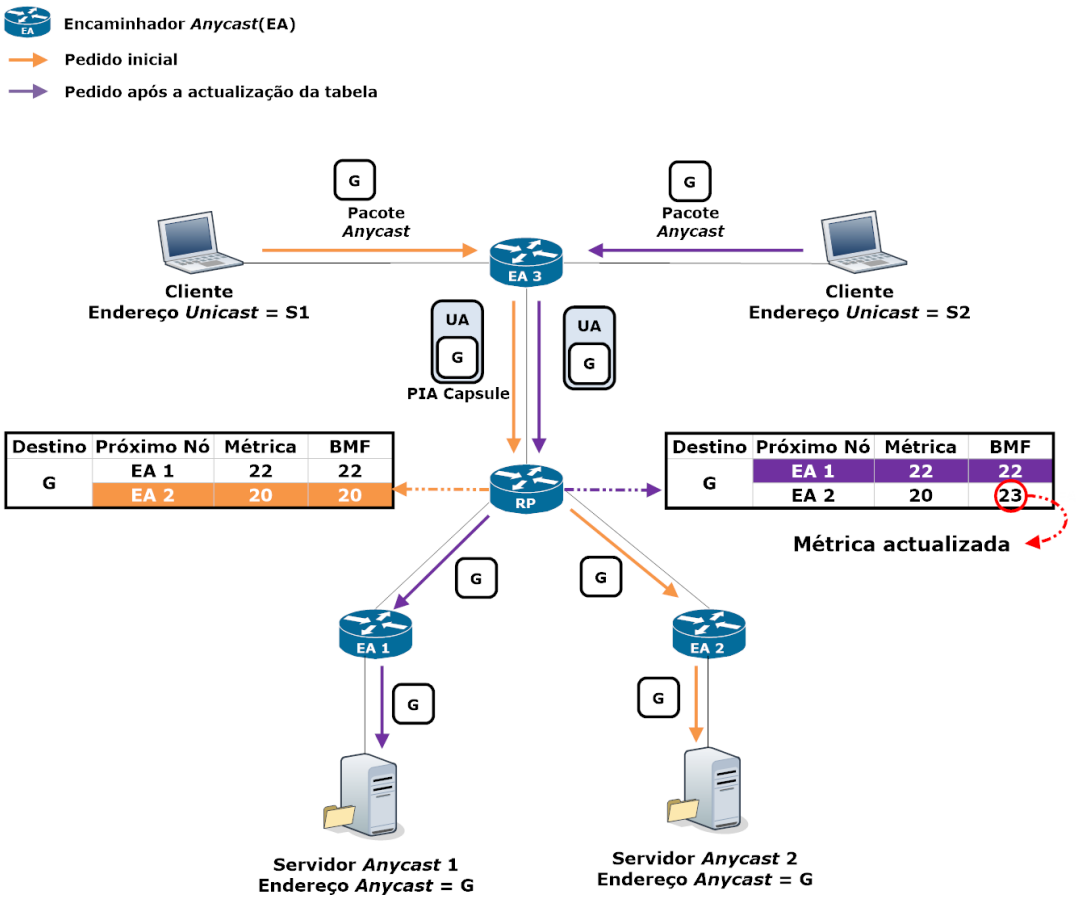


Figura 2.20: Encaminhamento dos pacotes *anycast* pelo *PIA-SM*

2.4 Resumo

Ao longo da primeira secção (2.1) foi realizado um enquadramento mais especifico ao *anycast*, onde foi realizado um levantamento dos principais acontecimentos da sua história e as suas problemáticas.

A secção 2.2, permitiu demonstrar que o encaminhamento intra-domínio *anycast* é atualmente uma realidade, ao passo que, o encaminhamento inter-domínio *anycast* ainda não é funcional. O principal motivo porque este não é funcional, deve-se ao facto de ser necessário manter na tabela encaminhamento, uma entrada para cada servidor de um grupo *anycast* (mesmo que partilhem o mesmo endereço). As propostas dos diversos autores possuem aspetos positivos e negativos, sendo necessário uma comparação entre elas.

A proposta *GIA (Global IP Anycast)*[17], na subsecção 2.3.1, tem como característica distintiva o facto de introduzir um novo sistema de endereçamento, obrigando a uma mudança no *IPv6*. Através do aparecimento do endereço *anycast*, é possível aos encaminhadores reconhecer

e tratar o tráfego de acordo com as especificações *anycast*. Utilizando o argumento de que um SA só deverá possuir rotas para grupos *anycast* que lhe interessam, os autores do *GIA*, propõem uma divisão em três diferentes grupos - `Grupo Interno`, `Grupo Externo Popular` e `Grupo Externo Impopular`.

Quando um domínio possuí um grupo *anycast* no seu interior, é considerado um `Grupo Interno`. Como se trata de encaminhamento intra-domínio, o número de rotas acrescentadas às tabelas não é crítico, possuindo no máximo uma entrada para cada servidor. Se um domínio deteta um número significativo de utilizadores a aceder a um endereço *anycast* externo, catalogará esse serviço como `Grupo Externo Popular`. Quando existe um grande interesse por parte dos utilizadores do domínio num grupo *anycast* especifico, desencadeia-se uma série de procedimentos para tentar encontrar o melhor caminho para esse grupo. O *GIA* define um novo pacote *BGP*[28], que tem como objetivo obter o melhor caminho. O encaminhador fronteira (*border router*) do domínio, envia o pacote, sendo este reenviado através dos encaminhadores até a sua validade expirar ($TTL < 0$) ou um encaminhador responder a esse pedido. Um encaminhador responde a um pedido de procura se conhecer um caminho para o grupo melhor que o atual. Quando o encaminhador que originou a mensagem recebe a resposta, atualiza a sua tabela de encaminhamento *anycast* e cria um túnel entre si e o encaminhador que respondeu. Os pacotes *anycast* seguintes, passam a ser encaminhados através do melhor caminho descoberto. Todos os outros grupos, que não interessam ao domínio, são considerados de `Grupo Externo Impopular`. Assim, o domínio adiciona uma rota padrão (*default route*), não sobrecarregando as suas tabelas de encaminhamento.

A subsecção 2.3.2, introduz uma nova abordagem ao encaminhamento *anycast*, a inclusão de redes sobrepostas. A proposta *PIAS (Proxy IP Anycast Service)*[18] tem como base a distribuição de um grande número de *proxys* pela *Internet*, com o intuito de resolver o problema de escalabilidade do encaminhamento *anycast* ao nível da camada de rede. Os *proxys* funcionam como gestores da rede *anycast*, anunciando rotas para grupos *anycast* através do *BGP*[28]. A responsabilidade da entrega dos pacotes não recai sobre os encaminhadores, sendo estes entregues pelo *proxy* mais próximo, através de encaminhamento *unicast* (figura 2.10). As soluções baseadas em redes sobrepostas nem sempre encaminham pelo melhor caminho, mas consegue melhorar a escalabilidade. Com este sistema de gestão de tráfego *anycast*, excluem-se as rotas de todo o grupo das tabelas de encaminhamento dos encaminhadores normais, introduzindo a melhor. Esta proposta alivia a carga colocada nos encaminhadores, mas acarreta a implemen-

tação de um grande número de *proxies*.

A última proposta da subsecção 2.3.2, denominada *ASTAS (Architecture for Scalable and Transparent Anycast Services)*[19], pode ser considerado um melhoramento à proposta anterior, principalmente na forma como faz o dimensionamento e posicionamento dos serviços na rede sobreposta. Aproveitando a dificuldade do *PIAS* em manter sessões longas, devido à especificação de manter o mínimo de informação complementar de controlo, foi proposta esta nova abordagem. Sempre que um cliente efetua um pedido, é iniciada uma nova sessão para garantir o melhor serviço possível, contrapondo a ideia do *PIAS* que só efetuava uma mudança de sessão quando ocorre-se uma falha no sistema. A alteração efetuada acrescenta uma grande disponibilidade para serviços críticos, aumentando consideravelmente o consumo de recurso por sessão.

O encaminhamento *multicast* e *anycast* possuem muitas propriedades semelhantes, o que levou a que vários investigadores estudassem a adaptação dos principais algoritmos *multicast* para a realidade *anycast*. Um do trabalhos mais notórios nesta vertente[21], escolheu três protocolos (*DVMRP*[22], *MOSPF*[23] e *PIM-SM*[24]) e procedeu ao estudo de uma possível implementação. Surgem três proposta baseadas nos protocolos anteriores, *DVARP* (2.3.3.2), *AOSPF* (2.3.3.2) e o *PIA-SM* (2.3.3.2), sendo abordadas as principais alterações. Os dois primeiros protocolos consomem muitos recursos, sendo propostas para serem aplicadas a pequenas redes. O último protocolo (*PIM-SM*) é um dos protocolos de encaminhamento mais utilizados ao nível global, com muitas vantagens sobre os principais concorrentes. O facto de consumir poucos recursos e ser desenhado para redes alargadas, constitui-se como o maior candidato à adaptação.

Num segundo artigo[25], os investigadores focaram-se na adaptação do *PIM-SM*, para criar um protocolo de encaminhamento *anycast* denominado *PIA-SM*. O *PIA-SM*, mantém a lógica do seu protocolo base e constrói árvores unidirecionais, centradas num ponto de encontro (*RP*). O *RP* é a raiz da árvore que une os membros do grupo. Ao contrário do que acontece no *multicast*, o *RP* e todos os nós da árvore partilhada somente encaminham para uma das interfaces, a que possui melhor métrica (figura 2.19). Quando o servidor recebe o pacote, estabelece a ligação com o cliente, por *unicast*.

A implementação do *PIA-SM*, não contempla o balanceamento de carga. Caso existam vários clientes simultaneamente, o servidor escolhido é sempre o mesmo, mesmo que esteja sobrecarregado. Uma outra variante do *PIA-SM*[27] aborda este problema, introduzindo um outro campo

à tabela de encaminhamento para além da métrica, campo esse que tem em consideração o facto de o servidor estar ocupado. O maior problema é que a manutenção desse campo é realizada pelo *RP*, um nó já muito sobrecarregado de tarefas.

A tabela 2.1 apresenta uma comparação entre os protocolos estudados, com a exceção da variante do *PIA-SM*[27]. Analisando a tabela 2.1 é possível verificar que a única proposta que assume que é necessário distinguir o endereço *anycast* do endereço *unicast* é o *GIA*. As outras propostas tentam adaptar soluções que não obriguem a alteração da norma, mas nunca conseguem uma alternativa verdadeiramente credível. Existem ainda duas alternativas que propõem uma rede sobreposta, para evitar sobrecarregar a *Internet* com mensagens de controlo *anycast*. A proposta *PIA-SM* também consegue uma implementação com poucas mensagens de controlo, mas falha principalmente por não conseguir encontrar o melhor caminho para a comunicação do cliente com o servidor.

Características	***GIA***	***PIAS***	***ASTAS***	***DVARP***	***AOSPF***	***PIA-SM***
Endereçamento	Sim	Não	Não	Não	Não	Não
Rede Sobreposta	Não	Sim	Sim	Não	Não	Não
Overhead	Sim	Não	Não	Sim	Sim	Não
Escalabilidade	Não	Sim	Sim	Não	Não	Sim
Melhor caminho	Não	Não	Não	Sim	Sim	Não

Tabela 2.1: Comparação dos protocolos de encaminhamento *anycast*.

O protocolo *PIA-SM* aparenta ser de todos o melhor candidato, é o protocolo que consegue manter a sua escalabilidade sem recorrer a redes sobrepostas, não obstante ao facto da sua especificação ainda conter alguns problemas. O facto do nó central conservar toda a informação, pode criar um grande fluxo de tráfego na sua direção, provocando congestionamentos. O problema de congestionamento junto do ponto de encontro é um problema também no tráfego *multicast*, sendo talvez o maior problema do protocolo. A última consideração a ser feita é relativo a forma como o *PIA-SM* calcula a métrica. Sendo esta calculada quando os servidores se juntam à árvore partilhada, a métrica obtida é relativa ao *RP*, o que pode originar que o servidor escolhido não é realmente o mais próximo do cliente, mas o mais próximo do *RP*, adulterando o princípio do encaminhamento *anycast*[5]. Atualmente, é possível afirmar que o

encaminhamento *anycast* ao nível da rede ainda não é uma realidade, sendo necessário continuar a explorar diferentes mecanismos para a sua implementação global.

3

Arquitetura Proposta

Neste capítulo é descrito um novo protocolo, denominado *Tree-based Anycast Protocol (TAP)*, que tem como objetivo permitir o encaminhamento *anycast* ao nível global. A arquitetura proposta é baseada no protocolo *multicast* para redes com recetores dispersamente distribuídos *PIM-SM*, sendo especificado o comportamento do sistema consoante a situação. Para facilitar a compreensão do sistema e seus benefícios, é descrito um exemplo prático da utilização do protocolo proposto.

O encaminhamento *anycast* ao nível da rede inter-domínio (global) ainda não pode ser considerado uma realidade devido a uma enorme desvantagem, que é o esforço a que os encaminhadores são obrigados. O desenvolvimento de uma simples rede *anycast* ao nível global, com quatro sistemas terminais espalhados pelos principais continentes (África, América, Ásia e Europa), obriga a que todos os encaminhadores ao longo da rede tenham que acrescentar quatro entradas às suas tabelas de encaminhamento, de forma a conseguirem encaminhar para o sistema terminal mais próximo. Com o aumento de serviços *anycast*, o esforço exigido à rede para suportar estas operações é incomportável, sendo esta a principal razão para a sua dificuldade de aceitação. De modo a superar esta adversidade, foi necessário estudar soluções para problemas semelhantes. Uma metodologia que também encontra dificuldades devido à sua constituição é o *multicast*. Tal como o *anycast*, no *multicast* existe a noção de grupo, sendo necessário conhecer o endereço do grupo para poder comunicar. Analisando o conjunto de protocolos, existe um que se realça devido às suas características, o *PIM-SM* [24].

O *PIM-SM* está otimizado para redes com recetores dispersamente distribuídos ao longo da rede. É designado por independente do protocolo, pois ao contrário de outros protocolos de encaminhamento *multicast* não depende de nenhum protocolo de encaminhamento *unicast* específico. Constrói árvores unidirecionais, centradas num ponto de encontro, normalmente designado por *Rendezvous Point (RP)*. O papel do *RP* é fundamental no protocolo, servindo de ponto de encontro entre as fontes e os recetores. O facto de utilizar um ponto de encontro

para fazer esta ligação, permite aos encaminhadores reduzirem o número de entradas, pois simplesmente precisam de possuir a entrada (*,G). Devido a esta característica, a adaptação deste protocolo *multicast* à realidade *anycast*, parece ser muito vantajosa, tendo já sido realizadas algumas adaptações[25][27].

Um último aspeto a diferenciar entre a metodologia *multicast* e *anycast*, é a forma como decorre a comunicação. Enquanto no tráfego *multicast*, o grupo de clientes estabelece ligação a uma fonte (ou a um conjunto de fontes), no caso de tráfego *anycast* a ligação é entre o cliente e o servidor mais próximo[1]. Comparando diretamente as duas metodologias, um recetor (ou recetores) no *multicast* equipara-se a um grupo de servidores no *anycast*, e a fonte no *multicast* equipara-se a um cliente.

3.1 Atribuição do Endereço *Anycast* a um Grupo

A primeira grande diferença entre o paradigma de comunicação *multicast* e o paradigma de comunicação *anycast* reside no facto de a primeira possuir um endereçamento próprio e distinguível dos endereços *unicast*. Os endereços *anycast* são absolutamente indistinguíveis dos endereços *unicast*[5], o que resulta num desafio extra para o seu encaminhamento na rede. O desafio acontece porque um encaminhador padrão[2] não consegue distinguir o tráfego e deste modo encaminhá-lo de forma distinta.

A abordagem seguida pelos autores de outra proposta que se focou igualmente no *PIM-SM*[25], considera que o endereço *anycast* atribuído a um grupo é o endereço *unicast* do nó inicial[3]. Quando um cliente efetuar um pedido, mesmo que nunca passe por um encaminhador que implemente o protocolo proposto (adiante designado por `Encaminhador Anycast (EA)`), viajará sempre até ao nó inicial.

Um encaminhador *anycast* (`EA`) é um encaminhador especial, que com um protocolo específico *anycast*, consegue identificar e encaminhar o tráfego, como acontece no *multicast*.

[1]Servidor com melhor métrica.

[2]Encaminhador que não possui nenhum protocolo de encaminhamento *anycast*.

[3]Primeiro servidor a ativar-se.

A criação de um esquema de endereçamento próprio é uma alternativa à abordagem anterior. O protocolo *GIA*[17] defende que, para ser possível um correto funcionamento do paradigma de comunicação *anycast*, é necessário conseguir distinguir o endereço *anycast* do *unicast*. A introdução deste novo tipo de endereço, dificulta a aceitação de um novo protocolo, pois o endereçamento utilizado é diferente do previsto na norma (indistinguível dos endereços *unicast*). Apesar desta desvantagem, as comunicações tornam-se mais seguras, pois o endereço *unicast* dos sistemas terminais nunca é conhecido. Os clientes comunicam sempre para o endereço do grupo, e os servidores respondem com o endereço de grupo no campo de origem. Outra vantagem é o facto da redução da carga sobre um EA, pois reduz o número de pesquisas (*lookup*) nas tabelas de encaminhamento. Um EA na primeira proposta (endereço indistinguível do *unicast*), é obrigado a verificar a tabela de encaminhamento *anycast* e, caso não encontre correspondência, também tem de verificar a tabela *unicast* para reencaminhar um pedido.

Na implementação do sistema foi escolhida a introdução de um esquema de endereçamento próprio para o *anycast*.

3.2 Escolha do Melhor Servidor

O paradigma de comunicação *anycast* necessita da introdução de um fator de seleção, que permita ao EA encaminhar o tráfego para o servidor mais capaz. Normalmente, o fator utilizado é o número de saltos, entre o cliente e o servidor.

Como a fixação de uma métrica pode ser algo limitador, é possível definir o cálculo da métrica de acordo com o desejo do criador do grupo *anycast*. Esta métrica pode passar por um sem número de parâmetros como o número de saltos, largura de banda, perda de pacotes, latência, carga atual do servidor, fiabilidade do trajeto, entre outros. Alguns destes parâmetros obrigam ao cálculo da métrica pelo percurso fim-a-fim, onde todos os nós a atualizam. A equação 3.1 é um exemplo de uma métrica combinada, sendo calculada fim-a-fim, onde todos os nós (ao longo da árvore) atualizariam a métrica.

$$\textbf{Métrica} = \mathbf{f}(\text{Nº de Saltos}) \oplus \mathbf{f}(\text{Largura de Banda Disponível}) \oplus \mathbf{f}(\text{Carga do Servidor}) \quad (3.1)$$

Ao longo do capítulo, é adotado como métrica, um parâmetro fixo calculado previamente

pelo servidor, de modo a simplificar a sua explicação e implementação. O parâmetro escolhido foi a carga total do servidor.

3.3 Descoberta de Encaminhadores Vizinhos

O funcionamento do protocolo baseia-se no facto que todos os EA têm conhecimento dos caminhos possíveis na sua topologia. No *PIM-SM* são trocadas mensagens *PIM Hello*, que funcionam como mensagens de reconhecimento de vizinhos. A figura 3.1 ilustra como se procede a localização dos vizinhos. Todos os encaminhadores EA trocam mensagens de reconhecimento entre si (a laranja na figura 3.1), sendo estas enviadas para o endereço *multicast* do segmento de rede. Com todas as ligações identificadas, designa-se um encaminhador responsável (*DR*) pelo envio de mensagens periódicas de aviso para o *RP*. Tal como no *PIM-SM*, o *DR* é eleito baseado no endereço lógico de cada interface, sendo escolhido o de maior valor. Cada segmento da rede tem de possuir um *DR*, como vemos na figura 3.1.

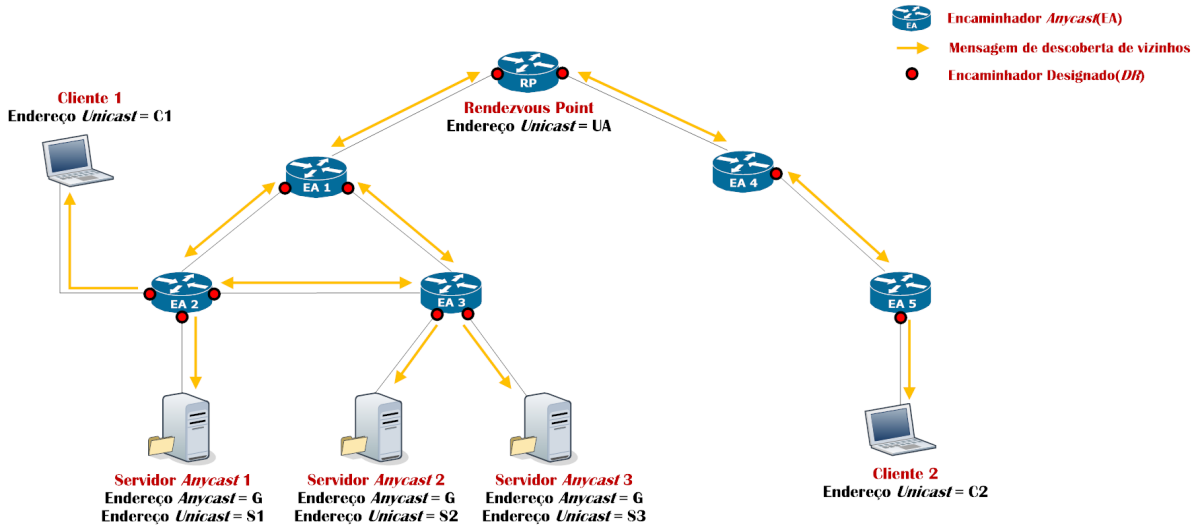


Figura 3.1: Descoberta de vizinhos.

Às mensagens de reconhecimento de vizinhos pode ainda ser adicionada uma opção que permita autenticar os EA, utilizando segurança ao nível da rede (IPSec). Este componente do modelo de segurança do *IPv6*, permite garantir três importantes aspetos: confidencialidade, integridade e autenticação.

A confidencialidade de uma ligação passa por garantir que a troca de informação é sigilosa,

onde a informação nunca é revelada a pessoas exteriores à comunicação. O uso da criptografia, permite garantir a confidencialidade, sendo para isso utilizados dois tipos de cifras: simétricas ou assimétricas. Nas simétricas, a mesma chave é usada para cifrar e decifrar a informação, sendo esta conhecida pelos intervenientes da ligação. As cifras assimétricas são mais seguras, pois utilizam dois tipos de chaves para realizar a cifragem e decifragem da informação, tendo como desvantagem o facto de serem mais complexas. A chave de cifragem, denominada de chave pública, é divulgada de modo a ser utilizada por terceiros, ao passo que a chave utilizada para decifrar, denominada chave privada, é conhecida somente pelo próprio interveniente. A não divulgação da chave privada, garante que só o sistema terminal desejado poderá proceder à decifragem da informação, previamente cifrada com a sua chave pública. As cifras assimétricas podem ainda utilizar a chave privada para cifrar o conteúdo da mensagem, decifrando a mensagem posteriormente com a chave pública. Esta alternativa é mais utilizado nas assinaturas digitais, para autenticar o originador da mensagem. Uma comunicação diz-se íntegra, quando a informação permanece inalterada, desde o seu envio até a sua receção. A utilização de algoritmos *hash* (como *MD5*[29], *HMAC*[30] ou *SHA*[31]), permite garantir a integridade da informação. A autenticação garante que não houve usurpação da identidade de outro, identificando o sistema terminal com que houve troca de informação.

O *IPSec* possuí dois protocolos, o *Authentication Header (AH)*[32] e o *Encapsulating Security Payload (ESP)*[33], que podem ser utilizados em conjunto ou separadamente. O *AH* tem como propósito garantir integridade e autenticidade e o *ESP* proporciona integridade e confidencialidade. A utilização dos dois protocolos em conjunto, permite obter um comunicação que assegura os três aspetos anteriormente referidos.

A implementação de um protocolo baseado no *PIM-SM*, faz com que alguns dos seus problemas de segurança devam ser levados em atenção[34][35], para garantir um protocolo seguro. A adição do cabeçalho de autenticação (AH) nas mensagens de descoberta de vizinhos, providência uma proteção da integridade e permite autenticar os sistemas terminais do grupo. O grupo fica assim protegido de comportamentos indesejáveis, como mensagens alteradas ou de membros não autorizados.

3.4 Criação da Árvore Partilhada

A escolha do *RP* é uma das decisões mais importantes deste tipo de protocolo, existindo diversos estudos para otimizar o seu posicionamento[36][37]. Este é utilizado como ponto de registo dos servidores, para proporcionar o encaminhamento de pacotes. A construção da árvore partilhada pode ser dividida em duas parcelas, o registo dos servidores junto do EA mais próximo da sua origem e deste com o *RP*.

O registo de um servidor *anycast* é semelhante ao que acontece com os recetores no *multicast*. É gerada uma mensagem de ligação ao grupo desejado, enviando esta mensagem para o EA mais próximo. Como um administrador pode pretender implementar mais do que um servidor (para o mesmo grupo *anycast*) no mesmo segmento de rede, o EA deve comunicar sempre com o endereço de origem na mensagem de registo. O endereço de origem identifica o endereço local do servidor, sendo a componente inicial da mensagem de registo, como podemos verificar na figura 3.2.

Figura 3.2: Pacote da mensagem de registo na árvore partilhada.

O registo de um servidor *anycast* junto do *RP* é efetuado pelo EA que é o *DR* no segmento de rede, aquando da receção da mensagem de ligação na rede local (a laranja na figura 3.3). O EA reage ao pedido de admissão do novo servidor, criando na tabela de encaminhamento *anycast*, uma entrada (*,G). Através do envio sucessivo de mensagens de ligação (equivalente ao *join* no *PIM-SM*, a azul na figura 3.3) em direção ao *RP*, constrói-se um ramo da árvore partilhada. O reenvio da mensagem de ligação serve-se sempre do nó do caminho mais curto até ao *RP*, ao longo de cada segmento.

Quando um EA recebe um pedido de ligação para um determinado grupo G, que não consta da sua tabela de encaminhamento *anycast*, necessita de adicionar uma nova entrada. A tabela 3.1 detalha a constituição de uma entrada (*,G).

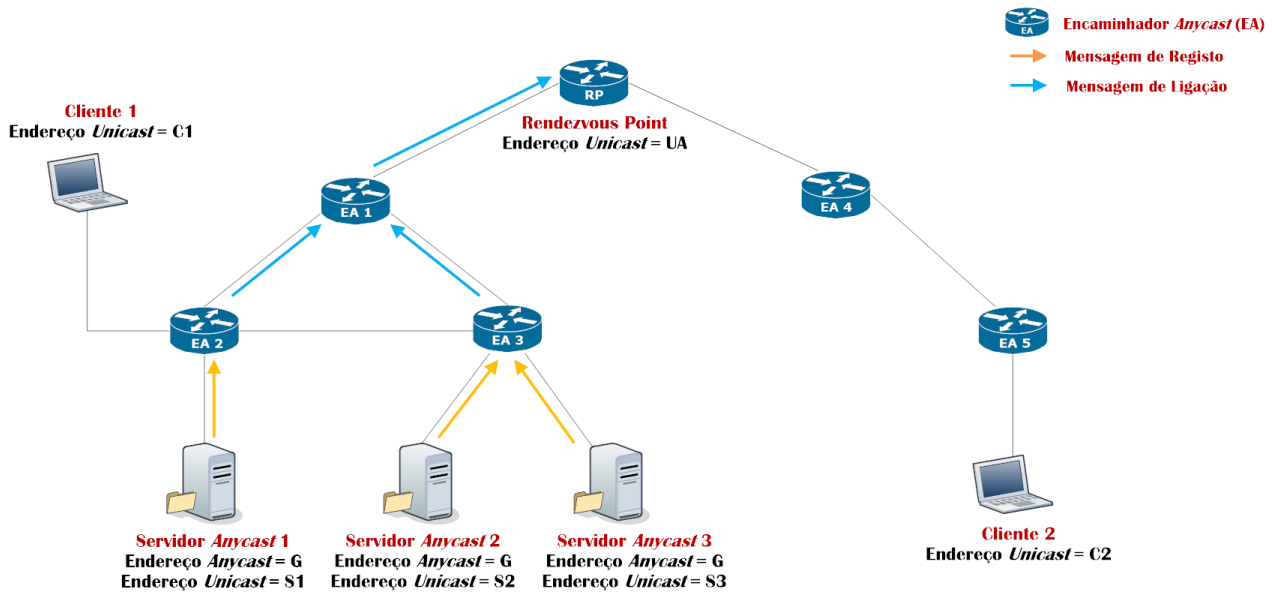


Figura 3.3: Criação da Árvore Partilhada.

Campo	Descrição
Endereço de Grupo	Endereço *Unicast* do nó inicial **(G)**.
Endereço do Cliente	Quando o campo se encontra preenchido com *, significa que não está centrada em nenhum cliente.
Interface de entrada	Interface utilizada para atingir o *RP* pelo melhor caminho.
Interfaces de saída	Conjunto de interfaces utilizados para atingir os sistemas terminais aquando da receção de uma mensagem de descoberta de servidores.
RPT-Bit(1)	O valor ativo (igual a 1) indica que corresponde à informação de estado relativa à árvore partilhada.
SPT-Bit(0)	Neste tipo de entrada, não tem qualquer significado.
Métrica	Neste tipo de entrada, não tem qualquer significado.

Tabela 3.1: Constituição de um entrada (*,G).

As mensagens de ligação trocadas entre os EA, de modo a construir a árvore partilhada, possuem um conjunto de campos semelhantes às mensagens de *join* no *PIM-SM*, como é possível verificar na tabela 3.2.

O papel desempenhado pelo encaminhador designado, não se esgota apenas na receção de pedidos e estabelecimento das respetivas ligações, devendo acrescentar o endereço do *RP* para efetuar periodicamente notificações, de modo a conservar o(s) servidor(es) ativo(s). Um EA

Campo	Descrição
Tipo de mensagem	Este campo pode assumir três tipos: ligação, exclusão ou atualização. Nesta mensagem trata-se de uma ligação.
Endereço de Grupo	Endereço *Unicast* do nó inicial **(G)**.
Lista de ligação	Inclui o endereço do *RP*
Lista de exclusão	Como se trata de uma ligação, o campo é nulo.
WC-Bit(1)	O valor ativo (igual a 1) indica que o encaminhador pretende receber tráfego proveniente de todas os clientes registados para o grupo G.
RPT-Bit(1)	O valor ativo (igual a 1) indica que se trata de um pedido de ligação à árvore partilhada.

Tabela 3.2: Constituição da mensagem de ligação na árvore partilhada

apaga a entrada (*,G) e retira o endereço do *RP* da lista de notificações, quando esse grupo não lhe interessar. O interesse deste pode ser por ter um membro diretamente ligado ou então servir de elo de ligação para outros membros.

3.5 Criação da Árvore Centrada no Cliente

Finalizado o processo da ativação dos servidores *anycast* junto do *RP*, é agora possível aos clientes efetuarem pedidos. A figura 3.4 ilustra o processo de descobrimento dos servidores por parte dos clientes. Quando `Cliente 1 (C1)` deseja iniciar a comunicação com os servidores, "encapsula" o pacote *anycast* de descoberta numa mensagem *unicast*, enviando até ao *RP* do grupo (a azul na figura 3.4). O facto de ser utilizado tráfego *unicast* permite que outros encaminhadores padrão sejam capazes de encaminhar o tráfego, mesmo não possuindo perceção do tráfego *anycast*. Ao receber o pacote, o *RP* "desencapsula" a mensagem e envia o pacote *anycast* com auxílio da árvore partilhada do grupo (a laranja na figura 3.4).

Quando um cliente começa o processo de localização dos servidores, um temporizador é iniciado, aguardando que os servidores construam uma nova árvore centrada no cliente, de modo a poder realizar pedidos. No caso do tempo se esgotar sem nenhum servidor se conectar a si, é enviado um novo pedido e reiniciado o temporizador.

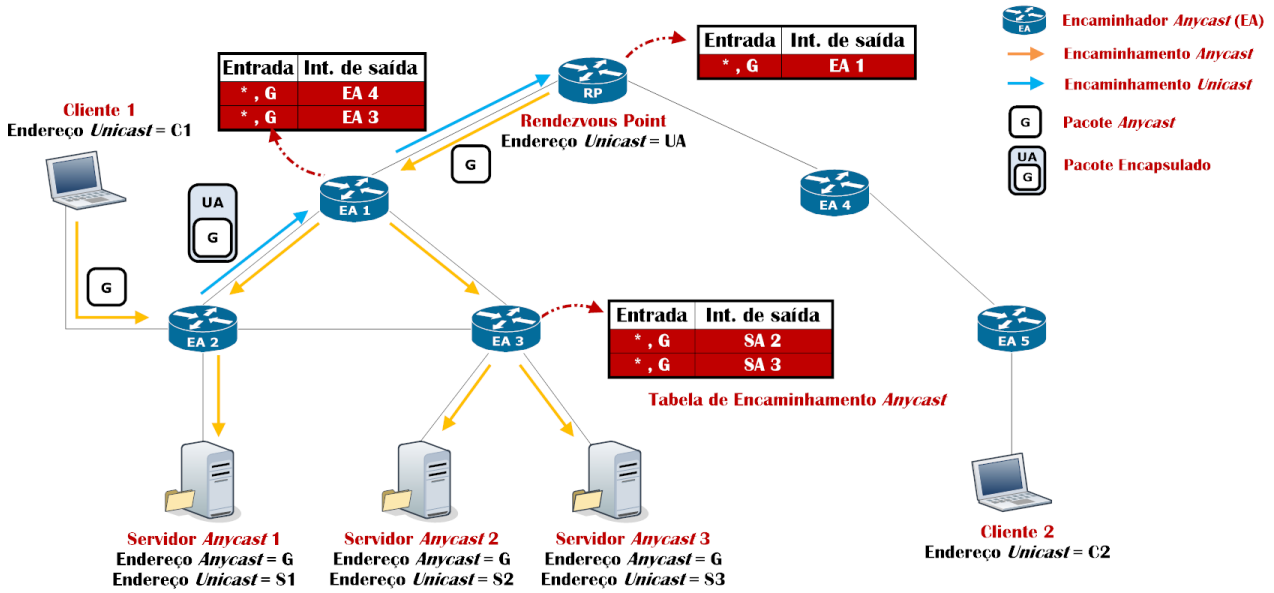


Figura 3.4: Pedido de localização por parte do Cliente 1.

Logo que o pacote de localização alcança um servidor, este calcula a sua carga total. Como acontece na mensagem de registo na árvore partilhada, é enviado um pacote para o `EA` mais próximo. Sendo este processo replicado por todos os servidores do grupo, garante-se que um cliente poderá realmente escolher o servidor com menor carga. A figura 3.5 ilustra a estrutura da mensagem para o registo na árvore centrada no cliente, onde é possível verificar uma alteração no pacote, a inclusão da métrica do servidor.

Figura 3.5: Pacote da mensagem de registo na árvore centrada no cliente.

Calculada a carga total para cada servidor no momento atual, é altura de comutar da árvore partilhada para a árvore centrada na fonte. Como é possível verificar na figura 3.6, todos os servidores efetuam a ligação ao `C1`, anunciando a sua métrica. Inicialmente todos os servidores apresentam a mesma carga, sendo escolhido o servidor que enviou a mensagem primeiro. Na figura, o `C1` receberia primeiro a mensagem do `SA 1`, pois este está mais próximo da sua localização.

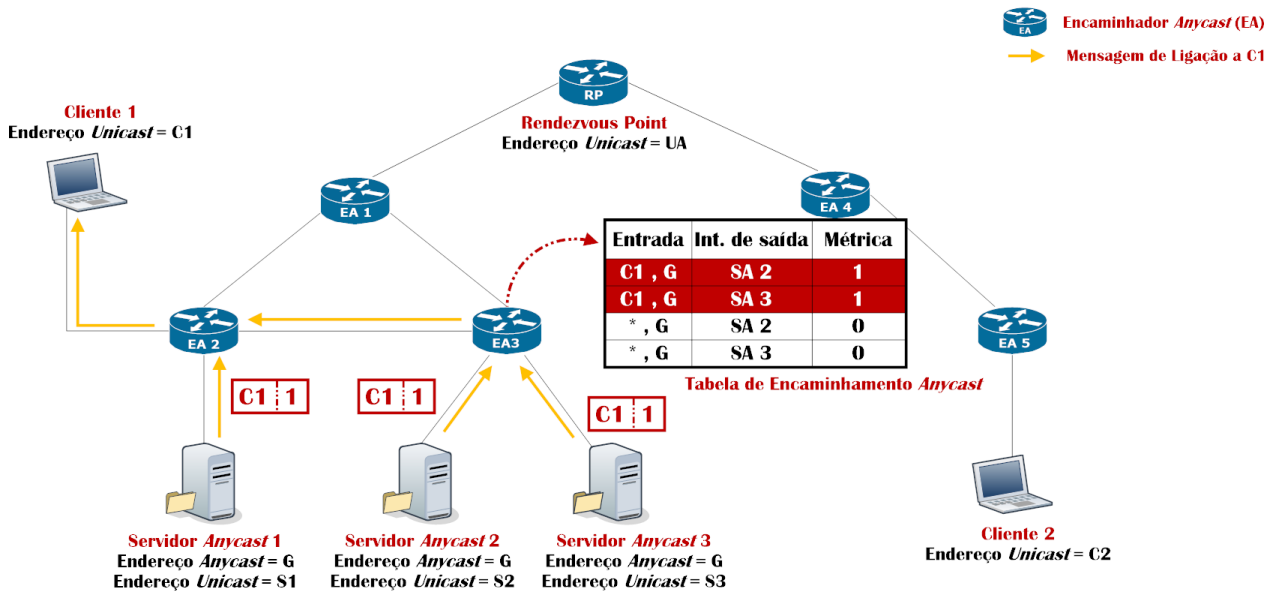


Figura 3.6: Criação da árvore centrada no cliente.

Quando um servidor decide centrar-se no cliente, o encaminhador designado (*DR*) do seu segmento de rede, começa por adicionar uma entrada (C,G) à sua tabela de encaminhamento *anycast*. A constituição da nova entrada (explicada na tabela 3.3) possui uma grande diferença em relação ao comportamento do *PIM-SM*. Enquanto o *PIM-SM* inicialmente coloca a 0 a *flag* `SPT-BIT`, aqui é colocada a 1, pois a comunicação deverá ser automaticamente comutada para a árvore centrada no cliente. É enviado um pedido de exclusão do grupo em direção ao *RP*, de modo a alertá-lo (e aos outros `EA`), que a partir daquele momento a comunicação decorrerá através da árvore centrada no cliente.

Campo	Descrição
Endereço de Grupo	Endereço *Unicast* do nó inicial **(G)**.
Endereço do Cliente	No caso do exemplo representado na figura 3.6, seria `C1`.
Interface de entrada	Melhor caminho para atingir o `C1`.
RPT-Bit(0)	O valor inativo (igual a 0) indica que não diz respeito à árvore partilhada.
SPT-Bit(1)	Significa que entrada está ativa e será utilizada para as restantes comunicações.
Interface de saída	Inclui o interface com o servidor com melhor métrica.
Métrica	Melhor métrica atual.

Tabela 3.3: Constituição de um entrada (C,G).

As mensagens de ligação trocadas entre os EA, de modo a construir a árvore centrada no cliente, possuem um conjunto de campos semelhantes às mensagens de *join* no *PIM-SM*. Como é possível verificar na tabela 3.4, a única diferença é a inclusão da métrica na mensagem, permitindo a cada encaminhador reconhecer o melhor caminho. A mensagem de atualização é em tudo semelhante à mensagem de ligação, alterando o tipo de mensagem, que permite aos EA tratarem-na condignamente.

Campo	**Descrição**
Tipo de mensagem	Este campo pode assumir três tipos: ligação, exclusão ou atualização. Nesta mensagem trata-se de uma ligação.
Endereço de Grupo	Endereço do grupo **(G)**.
Lista de ligação	Inclui o endereço do C1
Lista de exclusão	Como se trata de uma ligação, o campo é nulo.
WC-Bit(0)	O valor inativo (igual a 0) indica que o encaminhador pretende receber tráfego proveniente do C1 .
RPT-Bit(0)	O valor inativo (igual a 0) indica que se trata de um pedido de ligação à árvore centrada num cliente.
Métrica	Menor valor recebido.

Tabela 3.4: Constituição da mensagem de ligação à árvore centrada no cliente.

A exemplo do que acontece na construção de uma árvore partilhada, a árvore centrada no cliente deve adicionar o endereço do cliente à lista de notificações periódicas. Os EA, existentes entre o cliente e os servidores, devem atualizar/criar as suas entradas (C,G) em conformidade com o seu funcionamento.

3.6 Abandono do Grupo por parte dos Servidores

O abandono por parte de um servidor do grupo pode acontecer por dois motivos, por pedido do servidor ou por não efetuar a renovação da ligação.

Se o servidor decide abandonar o grupo, pode abandonar por completo (árvore partilhada e

todas as centradas no cliente de que faça parte) ou então simplesmente uma árvore centrada no cliente. O primeiro caso está normalmente relacionado com o desejo de terminar todas as comunicações no sistema terminal, começando o `EA` por percorrer todas a entradas e remover a ligação de saída para o servidor. Caso o `EA` verifique que a lista de interfaces de saída ficou vazia, deve notificar o próximo nó do abandono, nó do caminho mais curto em relação ao destino final (*RP* para árvores partilhadas e cliente para árvores centradas). O segundo caso pode acontecer quando um servidor estiver com demasiadas comunicações e decide abandonar alguns clientes para prestar um melhor serviço. Esta consideração caberá sempre ao gestor dos servidores. A aplicação, utilizada pelo cliente, deverá enviar para o *RP*, periodicamente, pedidos de localização. O envio deste pedido tem como objetivo não deixar expirar o ramo de ligação ao servidor e permitir a novos servidores efetuarem as suas ligações.

A mensagem de abandono é semelhante às abordadas anteriormente (ligação e atualização), possui o endereço de grupo, a lista de ligação a nulo e lista de abandono. Caso se trata de árvore centrada no cliente deverá conter o endereço do cliente e caso, árvore partilhada, o *RP*. Pode ainda ser um conjunto, no caso do abandono geral. As *flags* de sinalização mudam consoante o caso específico como se pode comprovar pela tabela 3.5.

Campo	**Descrição**
WC-Bit(1), RPT-Bit(1)	Abandono da árvore partilhada.
WC-Bit(0), RPT-Bit(0)	Abandono da árvore centrada no cliente.
WC-Bit(0), RPT-Bit(1)	Abandono de um cliente da árvore partilhada.

Tabela 3.5: Condições possíveis das *flags* aquando do abandono de um servidor.

A figura 3.7 representa um diagrama temporal sobre o processo de abandono da árvore partilhada e da árvore centrada no `C1`. Quando o servidor *anycast* 1 (`SA 1`) inicia o seu processo de abandono do grupo, propaga a mensagem para o endereço *multicast* da sua rede local. Quando o *DR* do segmento (`EA 2`) capta o pacote, apaga esse interface das suas de ligação de saída. Ao verificar que não possui nenhum interface de saída interessado na árvore partilhada, apaga a entrada e envia o pedido de abandono em direção ao *RP*, através do caminho mais curto. De seguida apaga o interface da lista ligada com as métricas e verifica se ainda tem interessados no cliente. Ao verificar que possui uma ligação para o `EA 3`, envia uma mensagem de atualização

para o cliente, com a nova métrica. Quando o EA 1 recebe o pedido de abandono da árvore partilhada, apaga esse interface de saída, mas não apaga a entrada pois continua a ter nós interessados (EA 3). Como a entrada não é apagada da tabela de encaminhamento, o EA 1 não necessita de propagar a mensagem em direção ao *RP*.

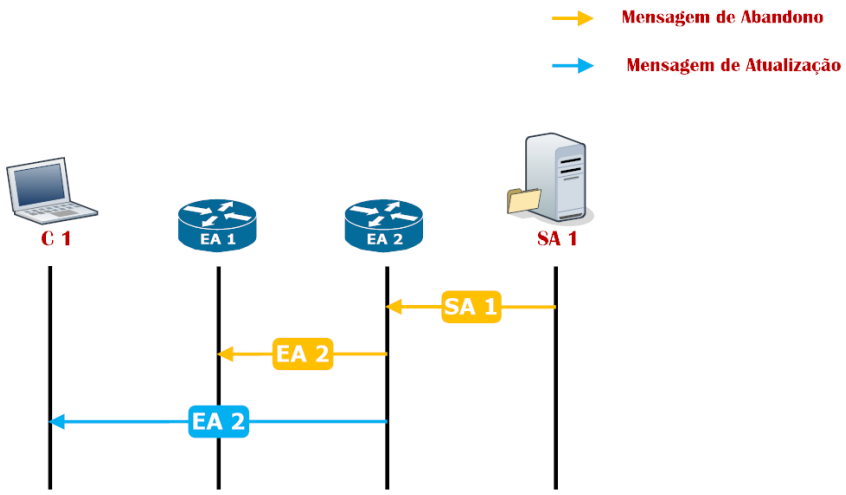


Figura 3.7: Diagrama Sequencial do Pedido de abandono por parte do SA 1.

A arquitetura proposta proporciona uma funcionalidade que permite comutar automaticamente para outro servidor *anycast*, sem voltar a refazer o serviço de localização. Tendo o SA 1 abandonado o grupo, passa a ser o servidor com a segunda melhor métrica até então (SA 2), a responder aos pedidos do C1, como podemos verificar pela figura 3.8.

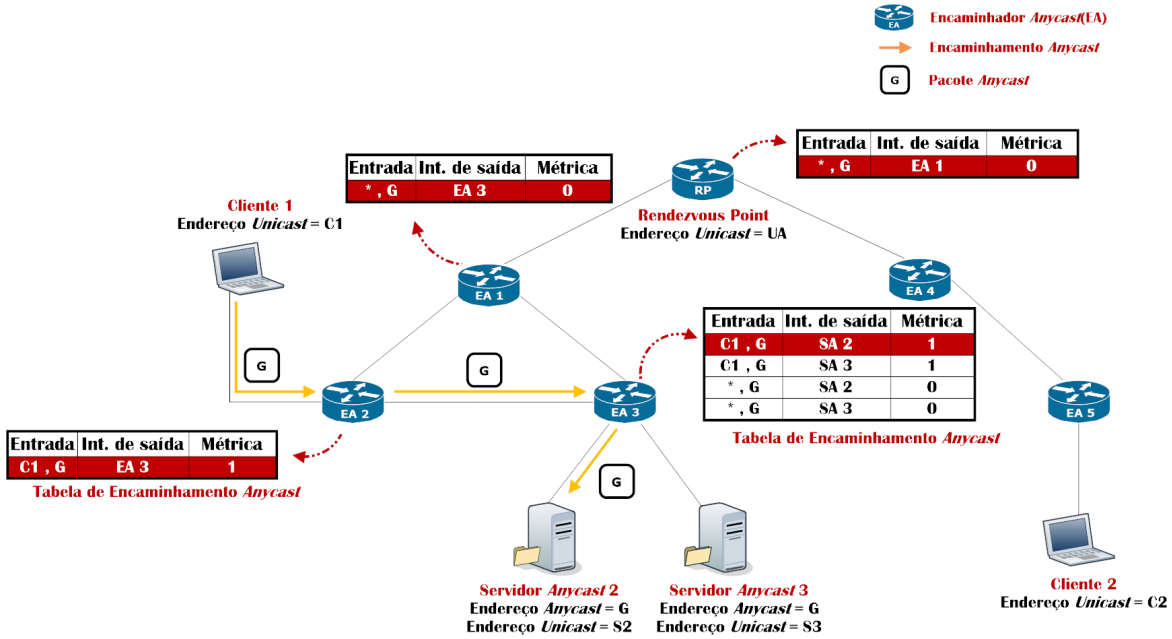


Figura 3.8: Resposta a pedido do C2 após o abandono do SA 1

3.7 Atualização da Métrica de um Cliente por parte dos Servidores

O balanceamento de carga é uma técnica consensual para quem quer implementar um serviço e manter uniforme a carga de trabalho dos seus sistemas terminais. Como o valor do atraso, na resposta dos sistemas terminais, é proporcional à carga que eles suportam no momento, é necessário repartir a carga por vários sistemas terminais, para evitar possíveis congestionamentos. A solução proposta engloba a possibilidade de balanceamento de carga, através da atualização da métrica do servidor.

A figura 3.9 demonstra o cenário de quando a métrica do SA 2 se degrada. Esta degradação pode passar por vários fatores, como por exemplo estar a receber demasiados pedidos. O SA 2 recalcula a sua métrica e comunica-o ao EA 3. Por sua vez, este verifica a sua tabela de encaminhamento e atualiza-a. Sendo que a menor métrica não sofreu nenhuma alteração (SA 3 tinha a mesma métrica), o EA 3 não necessita de informar o *EA 2*, o próximo encaminhador em direção ao cliente. Se houvesse uma alteração na menor métrica, o *EA 3* deveria reenviar a mensagem de atualização pelo caminho mais curto em direção do C1, sendo o processo repetido em todos os encaminhadores.

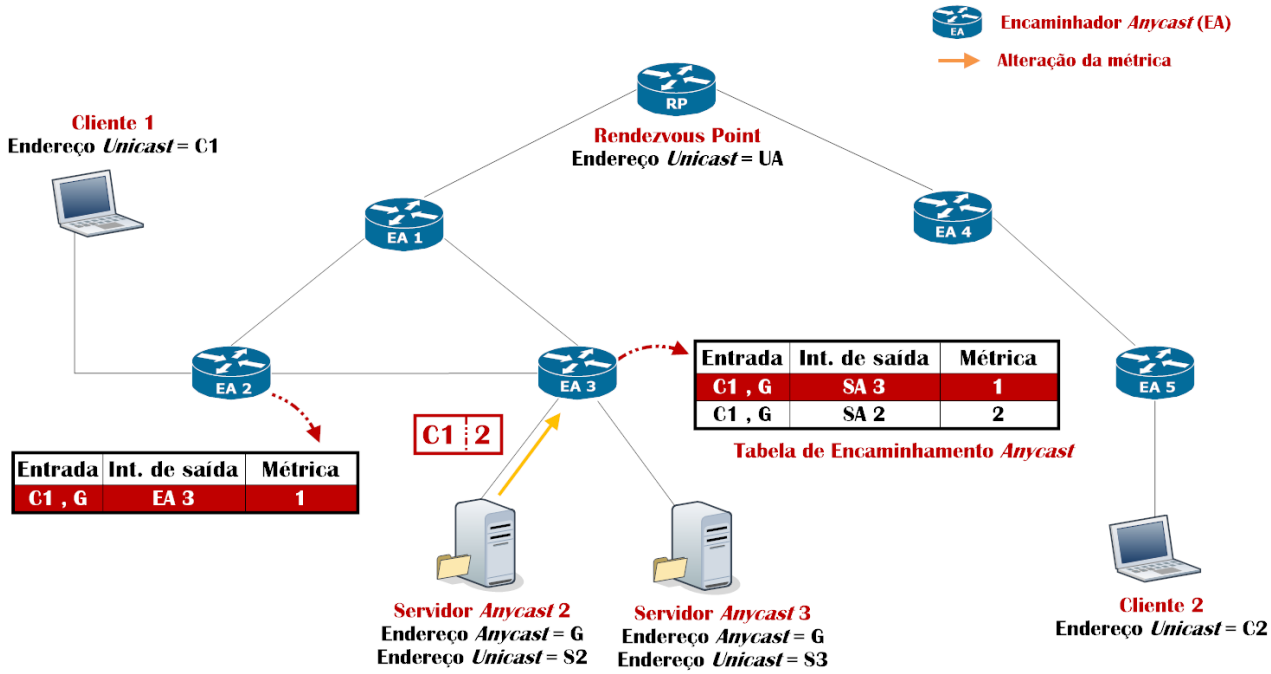


Figura 3.9: Alteração da métrica do SA 3

Perante a atualização da tabela de encaminhamento de EA 3, este agora passa a direcionar

os pacotes provenientes do C1 para o SA 3. A figura 3.10 demonstra o processo de encaminhamento após atualização da métrica, por parte do SA 2.

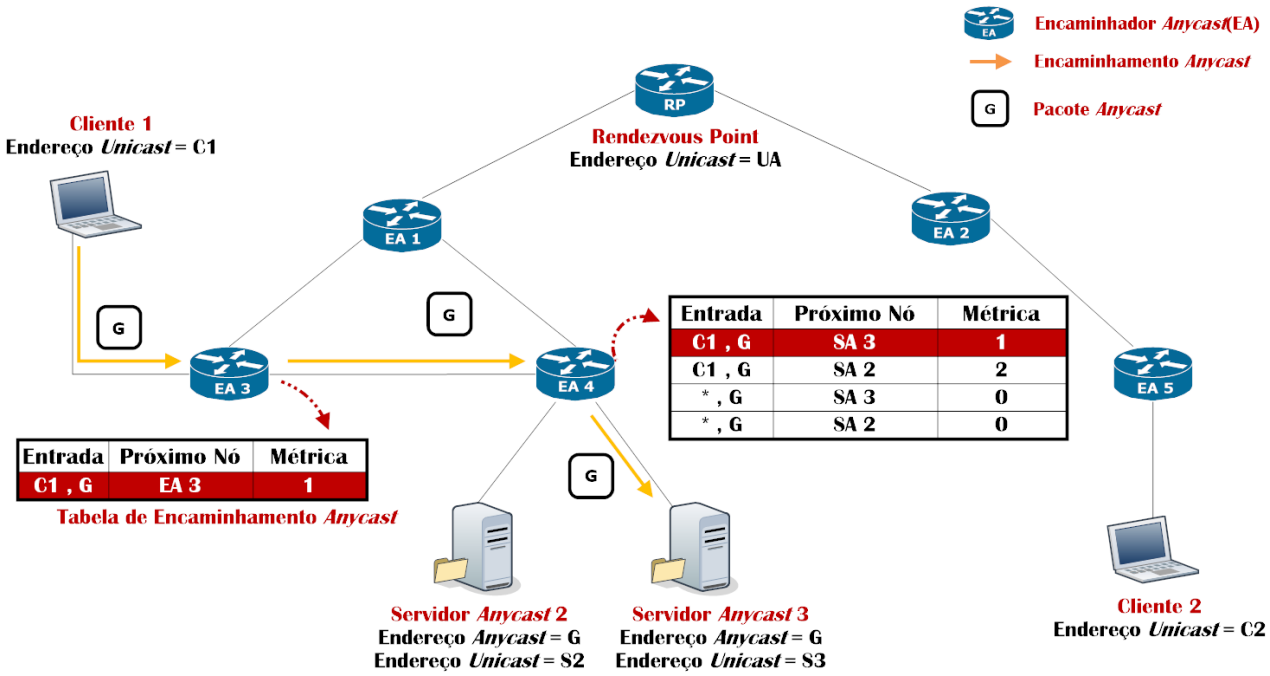


Figura 3.10: Comutação dos pedidos de C2 para SA 1

3.8 Aplicações beneficiadas

O protocolo especificado pode ser benéfico para enumeras aplicações, tais como, serviços dependentes da localização, descoberta de serviços ou balanceamento de carga, como explicado na introdução deste documento. Como as explicações podem ser sempre um pouco gerais, é necessário contextualizar um caso específico, de modo a evidenciar as funcionalidades do novo protocolo. Grande parte dos utilizadores da *Internet*, usam-na para encontrar serviços que estão alojados normalmente num sítio *web*. O tempo de resposta das páginas é fundamental para garantir um bom serviço, existindo muitas empresas dedicadas a aumentar o seu desempenho e disponibilidade.

A maior empresa neste sector é a *Akamai*, conhecida pelo serviço que oferece aos seus clientes, uma plataforma para alojar o seu conteúdo na *Internet*, ficando a *Akamai* responsável pela sua eficiência e disponibilidade. Para melhor perceber o seu funcionamento, é necessário aplicar a sua explicação a um exemplo prático. Uma pequena empresa portuguesa construiu um sítio *web*, que teve um aumento do número de visitas de forma exponencial. Passado algum tempo,

a procura é tanta e de tantos locais diferentes, que o serviço já não consegue responder com o desempenhado desejado. Nesse momento, os administradores da empresa consideram as suas possibilidades, para continuar a providenciar um serviço com qualidade. Com naturalidade, o próximo passo é de aumentar o número de locais onde o sítio está alojado, para permitir distribuir os pedidos por diferentes servidores. O problema surge porque a criação de um sistema completo de distribuição em todo o mundo acrescenta bastante custo, que irá sobrecarregar as contas de uma pequena empresa como a do exemplo. É aqui, que uma empresa como a *Akamai* entra em questão, permitindo à pequena empresa, mediante um custo menor, utilizar as suas infraestruturas com replicadores espalhados por todo o mundo. A pequena empresa, consegue aumentar a disponibilidade do seu sítio sem sobrecarregar as suas contas e a *Akamai* angaria um novo cliente, assegurando um desempenho satisfatório do novo serviço.

O protocolo desenvolvido no âmbito da dissertação pode ser uma grande ajuda para empresas com serviços idênticos à *Akamai*, permitindo escolher o servidor mais próximo da localização do pedido. Antes de explicar a forma como este protocolo permite ajudar, é necessário perceber como funciona um pedido *HTTP*. A função do *HTTP* é funcionar como um protocolo de pedido-resposta num modelo cliente-servidor, ou seja, estabelece-se a ligação entre o cliente e o servidor, permitindo ao cliente fazer pedidos ao servidor e receber a resposta deste último. O *HTTP* é um protocolo que possui várias versões, tendo acrescentada uma função na sua versão 1.1[38] importante para o protocolo especificado neste capítulo. A nova função, *Chunked transfer encoding*, permite fragmentar um ficheiro em blocos, cada um deles sinalizado com o seu tamanho em *bytes*. O último bloco de um ficheiro tem um tamanho zero, o que indica que é o fim da mensagem. Se este tipo de codificação estiver a ser utilizada para obter uma página *web*, o sistema terminal que requisitou a página(cliente), não necessita de aguardar que a página seja carregada na sua totalidade, sendo esta construída à medida que os pacotes chegam.

A utilização desta codificação em conjunto com o protocolo especificado neste capítulo, permitiria a uma empresa de alojamento de conteúdos, aumentar consideravelmente a sua eficiência. Utilizando a topologia da figura 3.1 para exemplificar as suas vantagens, imagine-se que uma empresa deste alojamento de conteúdo multimédia, possui um servidor na Europa (`Servidor Anycast 1`) e outros dois no continente americano (`Servidor Anycast 2` e `Servidor Anycast 3`). Quando um cliente, na figura 3.1 o `Cliente 1`, situado em Portugal pretende obter um ficheiro alojado nesses servidores, inicia a aplicação necessária para obter o ficheiro, por exemplo um *browser*. Essa aplicação, ao verificar que se trata de um endereço *anycast*,

envia um pedido de descoberta de servidores para o *RP*, que este depois encaminha para todos os servidores que compõem o grupo. Os servidores ao receberem o pedido de localização, calculam a sua métrica e criam uma ligação até ao `Cliente 1`. Ao receber a primeira ligação, o `Cliente 1` sabe que pode começar a efetuar pedidos, requisitando um ficheiro. Assumindo que o `Servidor Anycast 1` é o que possui melhor métrica, por estar mais próximo, os dois começam a comunicar. O cliente efetua o pedido do ficheiro desejado, começando a receber blocos deste ficheiro, à medida que a troca de informação entre os dois vai acontecendo. Se a comunicação ocorrer sem problemas, o cliente comunicará apenas com um servidor mas um servidor pode estar a receber demasiados pedidos ou mesmo falhar, causando o término da comunicação. Normalmente, se a comunicação não for completada, o cliente necessita de reiniciar o pedido. O novo protocolo obriga que todos os servidores se liguem ao cliente, para caso aconteça uma falha, não seja necessário reiniciar a comunicação, sendo o pedido comutado automaticamente para outro servidor. Regressando ao exemplo anterior, se o `Servidor Anycast 1` falhasse, os pacotes seriam automaticamente enviados para o segundo melhor servidor, mantendo a comunicação. O `Cliente 1` receberia os restantes blocos do ficheiro do `Servidor Anycast 2`, não obrigando a que o processo seja reiniciado.

3.9 Resumo

O presente capítulo faz um descrição detalhada da arquitetura proposta para permitir o encaminhamento *anycast* ao nível global, especificando os pontos fundamentais da especificação: atribuição do endereço *anycast* a um grupo, escolha do melhor servidor, descoberta dos encaminhadores vizinhos, criação da árvore partilhada, criação da árvore centrada no cliente, abandono do grupo por parte dos servidores, atualização da métrica por partes dos servidores.

Durante a secção que descreve a arquitetura proposta, duas importantes considerações foram tomadas, relativamente ao endereçamento e ao cálculo da métrica, apesar de serem estudadas outras abordagens. Foi decidido que o endereço *anycast* passaria a ser distinguível do *unicast*, o que facilitará a implementação do sistema num simulador. O valor utilizado como métrica é a carga do servidor, apesar de existirem outras alternativas, esta permite exemplificar as vantagens para o balanceamento de carga dos servidores.

Em conclusão à especificação do sistema, foi criada uma secção para permitir perceber as

vantagens de aplicação do novo protocolo num contexto real. A empresa *Akamai*, do sector de alojamento de conteúdo ao nível mundial, é utilizada como exemplo, sendo criado um cenário que demonstra a eficácia do sistema desenvolvido.

A principal vantagem deste novo protocolo é o facto de o cliente comunicar realmente com o servidor mais próximo (com melhor métrica) do seu local, através da utilização do pedido de localização de servidores. A segurança é outra das suas vantagens, pois os clientes nunca conhecem o endereço do servidor, comunicando sempre para o grupo. A última grande vantagem é a grande disponibilidade da rede. No caso da falha de um servidor, os pacotes são automaticamente comutados para outro servidor. O potencial entrave à aceitação desta proposta prende-se com o facto de ser necessário um endereçamento distinguível do *unicast*, obrigando a uma alteração da norma[5].

4

Implementação do Sistema

Este capítulo começa por fazer uma discussão crítica, acerca de qual o simulador que deverá ser utilizado para efetuar a implementação do sistema. Escolhido o simulador, este é descrito, bem como os parâmetros utilizados para implementação do sistema especificado no capítulo anterior. Por último, são apresentados aspetos de implementação do protocolo especificado no capítulo anterior, sendo explicados os seus principais métodos.

A implementação de um novo protocolo num ambiente real é normalmente muito dispendiosa e complexa. Os simuladores de redes de comunicações permitem efetuar um estudo prévio sem a necessidade de um laboratório de suporte, prestando estes um apoio fundamental aos investigadores. Um simulador tenta representar o mais fielmente possível um sistema real, permitindo assim tirar conclusões sobre o protocolo, através de um ambiente controlado.

O sistema especificado na secção anterior, foi implementado com auxílio do simulador de rede *Network Simulator 2 (NS2.35)*[11]. O *NS-2* é um simulador de distribuição gratuita e código aberto, o que lhe garante uma posição de destaque junto da comunidade científica. Normalmente descrito como um simulador de eventos discretos para redes de comutação de pacotes, possuí uma longa curva de aprendizagem. Atualmente, uma nova versão deste simulador de rede foi lançada (*NS-3*[39]), o que levou a comunidade académica a lentamente adotar esta nova versão. Em 2008, um estudo afirmava que cerca de 8000 *downloads* do *NS-2* eram feitos por mês[40]. A figura 4.1 permite verificar que durante o mês de Setembro de 2012, aproximadamente 13500 *downloads* do *NS-2* foram realizados, permitindo afirmar que, apesar da introdução de uma nova versão do simulador, a anterior ainda continua a ser muito utilizada.

Uma das desvantagens apontadas ao *NS-2* é o número elevado de códigos com defeitos (*bugs*), apontando a falta de verificação destes como o principal problema. O código base utilizado na implementação do protocolo proposto é baseado numa implementação do *PIM-SM* existente[41], tendo esta sido submetida a variadíssimos testes que comprovam o seu correto

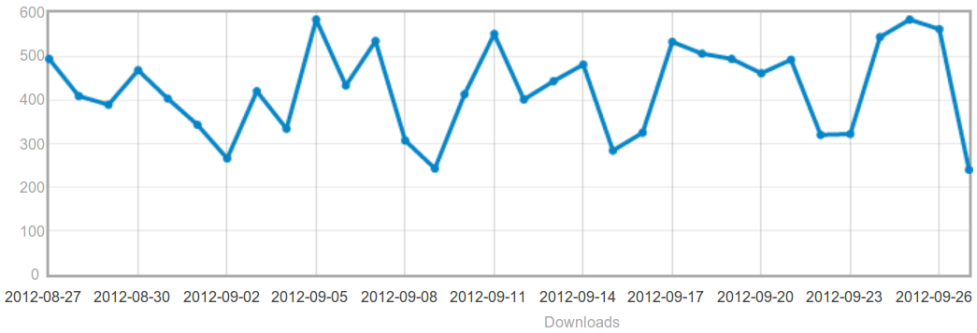


Figura 4.1: Gráfico downloads do *NS-2* em Setembro de 2012.[1]

funcionamento. O encaminhamento *multicast* no *NS-3* ainda se encontra pouco desenvolvido, não existindo ainda planos para efetuar uma implementação[1] do *PIM-SM*. Devido ao estado atual do encaminhamento *multicast* no *NS-3*, o protocolo proposto foi implementado no *NS-2*, com o auxílio da implementação do *PIM-SM* disponível, o que nos permitiu focar nos problemas relativos ao encaminhamento *anycast*, obtendo uma solução mais completa.

4.1 *Network Simulator 2*

O *NS-2* foi desenvolvido inicialmente pela Universidade da Califórnia sediada em Berkeley. O projeto *VINT (Virtual InterNetwork Testbed)*[42] trouxe um novo impulso ao simulador, contando com apoios do departamento de defesa dos Estados Unidos da América (*DARPA*), dos laboratórios da *Xerox Parc*, entre outros. O simulador é escrito em duas linguagens orientadas a objetos, o `C++` para execuções mais rápidas e o `OTcl` para ações que exigem mais flexibilidade. O `C++` é utilizado para tarefas mais frequentes, e que manipulem grandes quantidades de informação, ao passo que o `OTcl`, devido a ser mais fácil e rápido de escrever, é utilizado para configurações, mudanças ocasionais e definição do cenário de simulação.

A figura 4.2 ilustra a grande flexibilidade do simulador. Ao desenvolver um protocolo é possível optar por três tipos de objetos: objetos `C++` puros, objetos `OTcl` puros, ou objetos mistos entre o `C++` e o `OTcl`. A tentação de apostar numa só linguagem é grande, sendo difícil encontrar o compromisso certo. O `C++` apesar da sua menor complexidade, obriga a um aumento do tamanho do programa, ao passo que o `OTcl` é o contrário. A escolha da linguagem é sempre reversível, migrando os métodos de uma linguagem para a outra posteriormente. A

[1] `https://groups.google.com/forum/#!msg/ns-3-users/NA8_2z2hKsw/tQKMT2kme5EJ`

partilha de métodos e de variáveis entre as duas linguagens anteriores, é possível utilizando um conjunto de funções (`TclCL`, que proporcionam conectividade entre o `C++` e o `OTcl`.

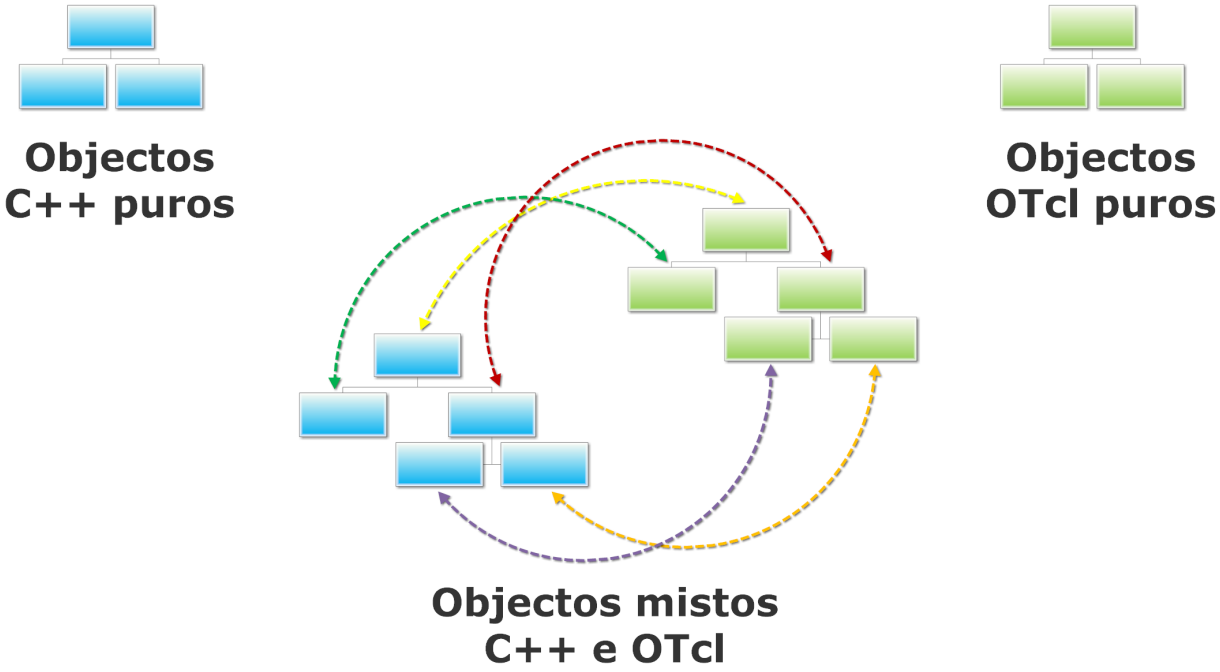


Figura 4.2: Dualidade das linguagens do *NS-2*.

O *NS-2* não possui uma interface gráfica para criar simulações, utilizando para isso *scripts* escritas em `OTcl`. Após um utilizador criar uma nova *script*, esta é interpretada e executada por um interpretador *OTcl*. Ao elaborar um cenário de simulação é necessário ter sempre presente que é preciso manter um certo grau de abstração da realidade. O primeiro impulso é o de incluir o máximo de detalhe possível, mas é necessário perceber que quanto maior o detalhe, mais recursos (memória, tempo de execução) são exigidos. A lista de tarefas habitual de uma simulação no *NS-2* é habitualmente a seguinte:

1. Criar o escalonador de eventos - este é responsável por acionar os eventos existentes na lista de eventos no tempo de simulação especificado e acionar o objeto que realizará a interpretação do evento.

2. Ativar as opções de rastreamento (*tracing*) para obter os dados desejados.

3. Criar a topologia da rede

4. Ativar o encaminhamento (*unicast* e *multicast*)

5. Introduzir erros e falhas nas ligações

6. Criar conexões de transporte (*TCP* e/ou *UDP*)

7. Transmitir dados entre as aplicações.

4.2 Encaminhamento *Multicast* no *NS-2*

O desenvolvimento de um cenário de simulação implica normalmente a utilização dos seguintes aspetos: nós, ligações, agentes e aplicações. Um nó, quer seja emissor, recetor ou um simples comutador de rede, ao ser criado, cria dois objetos, um classificador de endereços e um classificador de portas. O papel dos classificadores é selecionar o destino correto, quer seja um agente ou uma próxima ligação. A informação de cada nó engloba uma listagem dos nós vizinhos e outra com os agentes diretamente ligados a si. Dependendo do tipo do protocolo de encaminhamento, *unicast* ou *multicast*, um diferente classificador deve ser escolhido, estando armazenado num módulo de encaminhamento no nó. A ligação entre dois nós pode ser unidirecional ou bidirecional, sendo esta característica que define parâmetros como a largura de banda, o atraso de propagação, entre outros. O envio de informação no *NS-2*, acontece sempre de um agente para outro. Os agentes são aplicados ao nó desejado, definindo o protocolo de transporte a utilizar (*TCP* ou *UDP*). Um agente pode ter dois modos de funcionar, como emissor ou como recetor. Uma aplicação é encarregue do envio de pacotes, ou seja, de criar uma fonte de tráfego para a comunicação estabelecida. Esta comunicação entre o nó de origem e o nó de destino, pode utilizar um gerador com taxa constante ou aleatória, permitindo alterar diversos parâmetros da comunicação (tamanho de pacotes, taxa de transmissão, entre outros).

A implementação da estratégia de encaminhamento *multicast* encontra-se maioritariamente escrita em `OTcl`, sendo responsável pela troca de mensagens entre os nós, pela implementação do algoritmo e a construção das tabelas de encaminhamento. A componente que processa o envio de tráfego *multicast* em cada nó está implementada em `C++`.

A composição de um nó *multicast* está detalhada na figura 4.3. Comparando com um nó *unicast*, este necessita de mais um classificador, para encaminhar o tráfego *multicast* condignamente. A classificação de um pacote acontece normalmente pelo endereço e porta deste, identificando o destinatário, podendo tratar-se de um agente (dentro do próprio nó) ou então uma ligação (*link*) para outro nó.

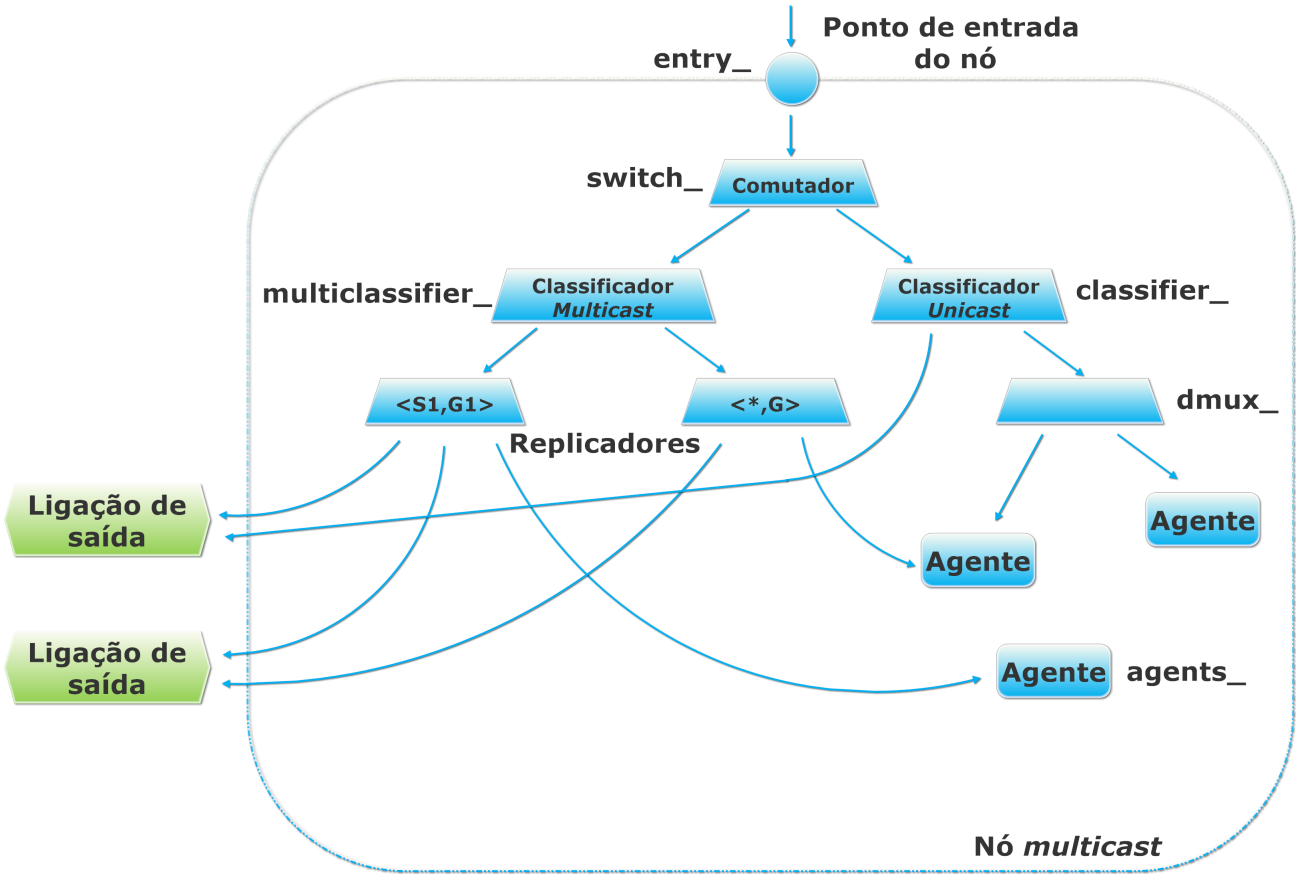


Figura 4.3: Encaminhamento num nó *multicast*.

O envio (ou reenvio) de pacotes entre os nós, acontece sempre com a invocação do método **recv**. A figura 4.4 permitir verificar a sequência dos acontecimentos entre os nós da rede, quando um emissor deseja enviar um pacote para um recetor. No exemplo ilustrado, o nó `1` pretende enviar um pacote para o nó `3`, começando o seu agente por gerar um pacote com informação, colocando o endereço do nó `3` como destinatário. De seguida invoca o método **recv**, enviando o pacote para o seu vizinho. O nó `2` ao receber o pacote, entrega-o ao seu classificador, decidindo este a quem o entregar. Ao identificar que se trata de outro nó, invoca o método **recv** e envia o pacote para a sua ligação de saída em direção ao nó *3*. Por fim, ao receber o pacote, o nó 3 classifica-o e, ao aperceber-se de que se trata de um pacote direcionado ao seu agente, entrega-o ao seu destino final. De modo a obter qual é o próximo destinatário, é invocado o método **find**, enviando de seguida o pacote.

O método invocado pelo *find* para descobrir o próximo destino é o **classify**. Consoante o tipo de classificador (*unicast* ou *multicast*) um ou mais destinos poderão ser encontrados. Sempre que pelo menos um destino é encontrado, é retornado para o método **recv** a sua ligação de saída, que enviará o pacote para este. Na eventualidade de não encontrar nenhum destino,

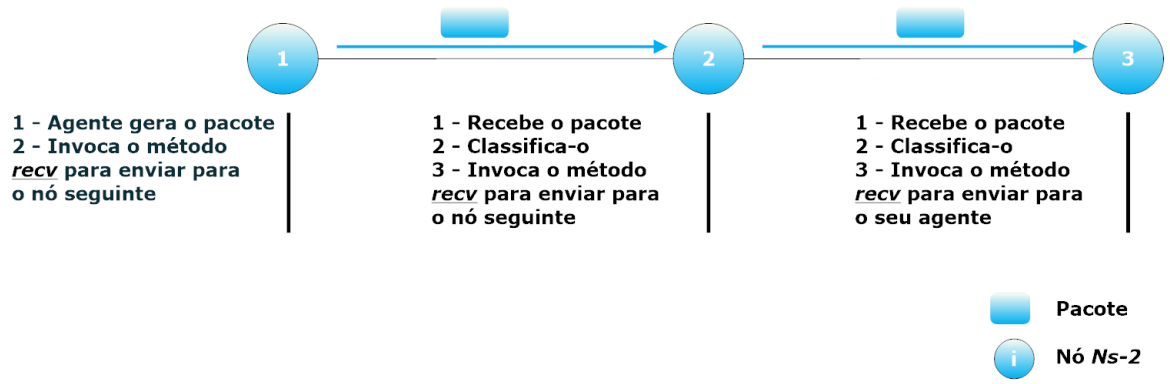


Figura 4.4: Envio de pacotes entre os nós no *NS-2*.

verifica se possui um *default target*, e em caso afirmativo, retorna-o. Caso as opções anteriores falhem, o método **classify** é novamente invocado, na tentativa de encontrar o destinatário.

O ponto de entrada de um nó, referenciado pela variável **entry_**, é o primeiro a ser invocado aquando da receção de um pacote. Um nó *multicast* encaminha de seguida o pacote para um comutador, que irá selecionar o classificador correto, de acordo com o endereço de destino. O comutador, identificado pela variável **switch_**, verifica o primeiro *bit* do endereço do pacote recebido e, se este for igual a 0, invoca o classificador *unicast*(**classifier_**), se for igual a 1, invoca o classificador *multicast* (**multiclassifier_**).

O classificador *unicast* ao receber um pacote, começa por verificar se é o destinatário. Se o pacote tiver sido dirigido ao nó, este invoca o classificador de portas (**dmux_**), selecionando e entregando ao agente desejado. Da mesma forma, se o pacote tiver como destino um nó diferente, é selecionada a ligação de saída para o próximo nó (em direção ao destinatário) e o pacote enviado.

Sempre que o pacote seja destinado a um grupo, o comutador invoca o classificador *multicast* de modo a encaminhar o pacote. O classificador *multicast*, ao receber um pacote, verifica as suas entradas, procurando por entradas com árvores centradas na fonte ou por árvores partilhadas, invocando um replicador para encaminhar os pacotes por todos os interfaces. O replicador possui essencialmente uma tarefa, fazer chegar um cópia do pacote de dados original aos membros do grupo.

A tabela e a lógica de encaminhamento do protocolo desejado, é sempre implementada no classificador. De modo a criar o protocolo *PIM-SM*, foi necessário alterar a estrutura de dados

bem como o classificador *multicast* padrão, introduzindo novas funções[41].

A figura 4.5 específica a estrutura de dados do classificador implementado para permitir o funcionamento do algoritmo *PIM-SM*. Além dos parâmetros habituais do *multicast* (origem, destino, *slots*, interface de entrada de pacotes de dados) foram acrescentados três campos: *SPT-Bit*, *RPT-Bit* e um temporizador (*Timer*). As *flags* acrescentadas, *SPT-Bit* e *RPT-Bit*, indicam se a árvore é centrada na fonte ou partilhada, respetivamente. A *flag* está ativa quando o valor é igual a 1 e inativa se o valor é igual a 0. O seu valor passa a ativo, quando recebe um pacote da fonte à qual se conectou. O temporizador criado para o *PIM-SM* é de contagem regressiva, definido com um valor específico, sendo esse valor decrementado até chegar a zero. Se um valor chegar a zero, ou seja, o nó não tiver recebido um *join* periódico por parte do nó vizinho em questão, a entrada é eliminada.

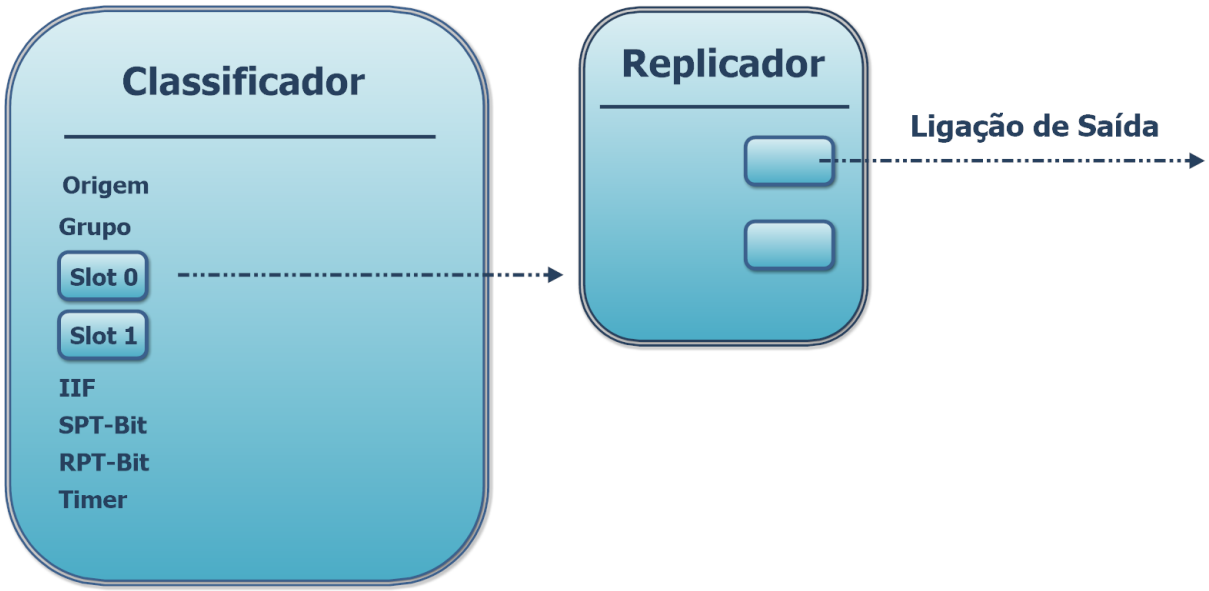


Figura 4.5: Estrutura de dados do classificador *PIM*.

O classificador *multicast* também necessitou de sofrer algumas alterações. Para isso, o método **classify** for reescrito, passando a classificar os pacotes recebidos de acordo com a tabela 4.1. O novo classificador, com o auxílio do métodos já existentes, **lookup** (árvores centradas na fonte) e **lookup_star** (árvores partilhadas), procura as entradas associadas e retorna os objetos associados. O classificador verifica primeiro as entradas do tipo (S,G), só procurando depois as do tipo (*,G). Durante o processo de verificação, se for detetado que uma entrada centrada na fonte ainda não está ativa (o *SPT-bit* com o valor a zero), é invocado um procedimento para finalizar a comutação da árvore centrada para a partilhada.

Origem	Destino	Int. entrada	SPT-Bit	RPT-Bit	Ação
S	G	iif	1	x	Re-envia
S	G	iif	0	x	Re-envia Se (iif != IIF(*,G)) - Ativa SPT-BIT - envia PRUNE (*,G)
S	G	não coincide	1	x	Descarta
S	G	não coincide	0	x	Ação baseada em (*,G)
*	G	iif	x	x	Re-envia
*	G	não coincide	1	x	Descarta

Tabela 4.1: Comportamento do classificador para o *PIM-SM*.

A inserção e remoção de rotas para permitir o encaminhamento *multicast*, é realizada noutros módulos. Os autores da implementação do *PIM-SM* [41], tiveram a necessidade de criar um novo agente de controlo *multicast*, para encaminhar os dados de acordo com o protocolo. De modo a construir, manter e destruir as árvores (partilhadas e as centradas nas fontes), foi necessário criar as mensagens de **join** e **prune**, sendo estas utilizadas pelo novo agente *PIM-SM* para enviar, receber e processar mensagens de controlo.

4.3 Classificador *anycast*

A implementação da abordagem especificada no capítulo anterior, obrigou a alterar a grande maioria das funções do *PIM-SM*, para que um nó consiga distinguir qual o melhor caminho (o servidor com métrica mais baixa). A primeira tarefa a realizar foi criar um classificador *anycast*, que é bastante semelhante ao classificador *multicast*, mas que possui uma nova lista ligada. A nova lista permite a cada nó verificar qual o melhor destino para atingir um servidor *anycast*. O código 4.1, possui a declaração da nova estrutura de dados. A estrutura é composta por três campos: a ligação de saída para o objeto em questão (linha 2), a sua métrica (linha 3) e um apontador para o próximo objeto (linha 4).

Exemplo de código 4.1: Lista ligada com a métrica e a ligação de saída para cada nó

```
1 typedef struct sMetricArray{
2         NsObject* obj;
3         int mtr;
4         struct sMetricArray *next;
5 }Nmetricarray, *LMA;
```

A estrutura de dados do classificador *anycast*, representada na figura 4.6, manteve os mesmos campos que o classificador *PIM-SM*, acrescentando a lista ligada representada pelo código 4.1. Os dois primeiros campos, `Origem` e destino `Destino`, são o endereço de origem do originador do pedido e o endereço de destino do grupo, respetivamente. A combinação destes dois campos permite identificar a entrada em questão. O `slot` é na prática, a posição dentro de um *array* onde está uma referência ao objeto `Replicador` que, por sua vez, tem também um array com `slots`, mas que apontam para ligações de saída por onde os pacotes são replicados. O campo `IIF`, representa o interface entrada por onde os dados devem chegar. As *flags*, `SPT-BIT` e `RPT-Bit`, e o temporizador (`Timer`), possuem a mesma lógica da explicada anteriormente para o classificador *PIM-SM*. Por fim, encontra-se a lista ligada (a verde na figura 4.6), criada para o encaminhamento *anycast*.

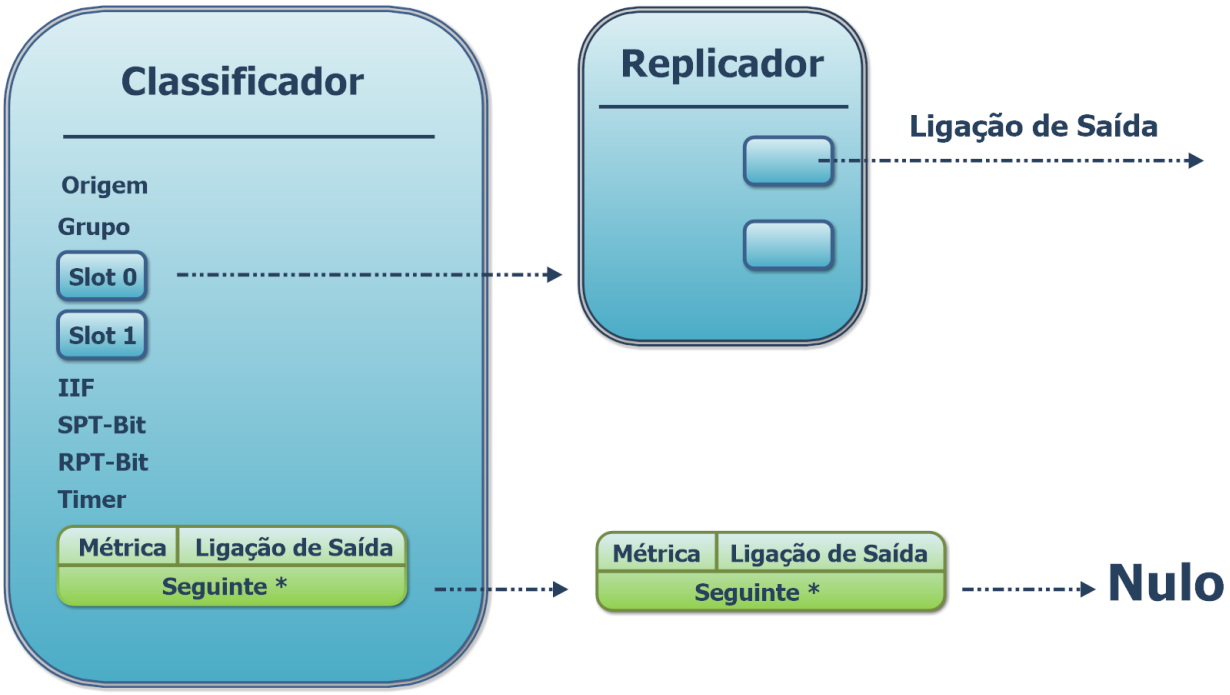


Figura 4.6: Estrutura de dados do classificador *TAP*.

A lista ligada obrigou a criação de novos métodos para proceder à sua gestão. Os seguintes métodos são descritos sucintamente, de modo a que posteriormente no documento, quando um método se relacionar com estes, não existam dúvidas sobre o seu objetivo.

- **addMetric** - possui como argumentos a lista ligada atual, um nó e a sua métrica. Ao ser invocado, adiciona uma nova entrada à lista ligada, retornando a lista completa.
- **checkMetric** - mais uma vez, recebe como argumentos a lista ligada atual, um nó e a sua métrica. O valor que retorna pode assumir quatro valores: **-1** para quando a métrica desse objeto na lista é mais alta do que a passada por argumento, **0** se a lista ligada se encontra vazia, **1** para quando a métrica é igual e **2** para quando a métrica antiga é mais baixa.
- **minMetric** - a lista ligada atual é o único argumento que recebe. Calcula o valor mínimo na lista e retorna-o. Na eventualidade de a lista ligada estar vazia, o valor **-1** é retornado.
- **updateMetric** - recebe como argumentos a lista ligada atual, um nó e uma nova métrica. De seguida, percorre a lista ligada até encontrar o nó em questão, atualizando a sua métrica.
- **getMetricNode** - é invocado quando se deseja o valor da métrica de um determinado nó. Os seus argumentos são a lista ligada atual e o nó que se deseja saber a métrica. A lista ligada é percorrida e, quando o nó é encontrado, o seu valor é retornado. Se a lista ligada estiver vazia, é retornado o valor **-1**, e caso o objeto não exista na lista, é retornado o valor **-2**.
- **removeMetric** - este método remove a entrada de um determinado nó. Recebe como argumentos a lista ligada atual e um nó. Começa por verificar o primeiro nó na lista ligada e caso se trate de o objeto em questão, apaga-o, apontando diretamente para o seguinte. Na eventualidade de o objeto não estar na primeira posição, percorre a lista ligada até encontrar o nó desejado. Ao encontrar o nó, remove-o, passando o objeto que apontava para o nó desejado, a apontar para o nó seguinte.
- **bestNode** - o seu único argumento é a lista ligada atual. A sua função é percorrer a lista ligada e calcular qual é o melhor nó, retornando o valor da sua ligação de saída.

Além dos métodos criados de raiz, outros métodos necessitaram de ser reescritos, para passar a funcionar de acordo com o *anycast*. O método **find** assume um papel importante na implementação, pois é o primeiro método a ser invocado aquando da receção de um pacote. De acordo com o modelo especificado na secção anterior, quando um cliente deseja efetuar um pedido pela primeira vez, envia um pacote de descoberta de servidores. Esse pacote deverá ser entregue a todos os servidores, a exemplo do que acontece no *multicast*, sendo utilizados os replicadores em cada `slot` para fazer chegar a todos os servidores, os pacotes. A descoberta de servidores não necessitou de alterar o encaminhamento, pois continua a ter um comportamento semelhante ao *multicast*. Os servidores ao receberem o pedido, devem efetuar a ligação à árvore centrada do cliente, podendo iniciar-se a comunicação. O encaminhamento durante a comunicação, deixa de ser para todos os membros do grupo e passa a ser para um único membro, o de melhor métrica. Esta nova característica, obrigou a alterar o comportamento do classificador *anycast*, para permitir que o nó soubesse se deveria enviar para todos ou somente para o de melhor métrica. O método **classify** foi reescrito, alterando a lógica da tabela 4.5. Os primeiros casos da tabela, quando o nó recebe um pacote e possui uma entrada de uma árvore centrada no cliente, passa a retornar um valor pré-definido (**-1**), deixando de retornar o valor do `slot`. O método **find** altera então o seu funcionamento, passando a comparar, inicialmente, o valor retornado pela função **classify** com o valor pré-definido. Se o valor for igual, ou seja, o valor retornado na função **classify** é **-1**, encaminha o tráfego para o melhor nó, como o código 4.2 exemplifica. Começa por retirar os dados do pacote (`Origem`, `Destino` e o `IIF`), para identificar a entrada em questão. O método **bestNode** é invocado de seguida, com a lista ligada da respetiva entrada, retornado a ligação de saída para o melhor nó. Por fim, é retornado o valor para a função **recv**, que trata de enviar o pacote. Se o pacote fosse de descoberta de servidores e ainda não existisse a entrada da árvore centrada no cliente, o procedimento era tratado como na implementação do *PIM-SM*.

Exemplo de código 4.2: Extrato do método **find** utilizado para encontrar o melhor servidor.

```
1 NsObject* node = NULL;
2 int cl = classify(p);
3 if(cl == -1) /* when is "-1", S,G,iff entry found. We need to find the best
      Node and return the outlink. */
4 {
5   hdr_cmn* h = hdr_cmn::access(p);
6   hdr_ip* ih = hdr_ip::access(p);
```

```
7    nsaddr_t src = ih->saddr();
8    nsaddr_t dst = ih->daddr();
9    int iface = h->iface();
10   /* get the pimhashnode to this set(src,dst,iface) */
11   pimhashnode* p = (pimhashnode *) lookup(src, dst, iface);
12   /* find the node with the best metric */
13   node = bestNode(p->l);
14 } else{
15 (...)
```

Dois métodos que também sofreram alterações foram o **set_hash** e o **unset_hash**. Estes permitem a alocação e remoção das entradas na memória, respetivamente. O método **set_hash** manteve a configuração do *PIM-SM*, acrescentando a inicialização da lista ligada da entrada a nulo. O método **unset_hash**, o oposto do método anterior, passou a proceder à completa remoção da lista ligada da memória.

Todos os métodos, descritos no C++, são invocados no agente de controlo (escrito em OTcl), através do **multiclassifier_**. O método que recebe os pedidos do agente de controlo vindos do **OTcl** e retorna a informação desejada é o **command**. O método **command** é inquirido pelo o **multicassifier_** (OTcl), invoncado os métodos desejados (no C++), para retornar novamente a informação para o OTcl.

4.4 Agente de Controlo *Anycast*

O agente utilizado como base do *PIM-SM*, tinha como funções enviar, receber e processar mensagens de controlo para construir, manter e destruir a árvore partilhada e as árvores centradas nas fontes. As mensagens de controlo, de ligação e de abandono (doravante designadas de **join** e **prune**, respetivamente), serão alteradas, para permitir a inclusão da métrica e uma propagação de acordo com a especificação *anycast*.

Uma nova mensagem de controlo foi criada, para permitir que os servidores possam atualizar a sua métrica. A mensagem de atualização (doravante designada por **update**), obrigou a declaração de um novo tipo de pacote na classe `packet.h`. A `mcast_ctrl.cc` também

necessitou de alterações, para reconhecer este novo tipo de pacote e classifica-lo como tal.

4.4.1 Métodos de registo, abandono e atualização por partes dos servidores.

Um servidor pode gerar três tipos de mensagens - registo (**join**), abandono (**prune**) e de atualização (**update**). A mensagem **join** gerada pelo servidor é semelhante à mensagem do *PIM-SM*, possuindo a diferença de passar a métrica (`mtr`), como é possível observar entre as linhas 1 e 3 do código 4.3. Na implementação do sistema, o cálculo da métrica é um parâmetro fixo calculado previamente pelo servidor, como a carga atual do servidor, introduzido na *script* de simulação. A seguinte mensagem de **prune** (entre as linhas 4 e 6 do código 4.3) é enviada sempre que um servidor deseja terminar a sua conexão (em relação a um cliente ou todas as suas conexões). Em relação ao método semelhante do *PIM-SM*, acrescenta o seu agente (`agent`), para facilitar a sua remoção da lista ligada. A mensagem de **update** (entre as linhas 7 e 9 do código 4.3), acrescentada para o *anycast*, tem como argumentos o agente e a sua nova métrica, para os nós alterarem ao longo do caminho o valor da métrica anterior.

Exemplo de código 4.3: Métodos de registo, abandono e atualização por partes dos servidores.

```
1 mrtObject instproc join-group { grp mtr src } {
2   eval $self all-mprotos join-group $grp $mtr $src
3 }
4 mrtObject instproc leave-group { grp src agent } {
5   eval $self all-mprotos leave-group $grp $agent $src
6 }
7 mrtObject instproc update-group { grp mtr agent src} {
8   eval $self all-mprotos update-group $grp $agent $mtr $src
9 }
```

Um nó *anycast*, através das mensagens anteriores, está sujeito a duas situações: o agente incluído no nó decide ligar-se, abandonar ou atualizar-se em relação a um grupo ou a um cliente ou o nó recebeu uma mensagem de *join*, *prune* ou *update* de um nó vizinho. As seguintes secções explicam como se comporta o nó, consoante a situação.

4.4.2 Mensagens de ligação entre os nós

As mensagens de ligação são iniciadas pelo nó mais próximo de um servidor, servidor esse que requereu o seu registo anteriormente, desencadeando a troca de mensagens entre os restantes membros. Existem dois métodos para efetuar a ligação, o **join-rpt** e o **join-spt**, que enviam as mensagens em direção ao *RP* e ao cliente, respetivamente. Os dois métodos são invocados pelo método **join-group**, que é basicamente um interpretador, que escolhe qual deles deve executar, de acordo com os parâmetros que lhe são passados. Se o objetivo for juntar-se à árvore partilhada, apenas o grupo é passado como argumento, invocando o método **join-rpt**. A ligação a uma árvore centrada no cliente é sempre posterior a uma ligação à árvore partilhada, sendo sempre necessário indicar qual é o grupo, o cliente e sua métrica, para passar estes com argumentos para o método *join-spt*. A receção de pedidos de ligação é tratado pelo método **recv-join**

O método de ligação à árvore partilhada é semelhante ao implementado pelo *PIM-SM*. A métrica não é calculada em relação ao *RP*, pois esta ligação serve para a propagação da mensagem de descoberta de servidores. O registo de um servidor no grupo pode acontecer perante três diferentes situações:

1. a entrada (*,G) ainda não está criada. Nessa situação é criada a entrada na tabela de encaminhamento, ativando a sinalização da condição, `RPT-Bit` com o valor 1. Por fim, envia uma mensagem de **join** para o nó mais próximo em direção ao *RP*.

2. existe a entrada (*,G), mas esta não possuí nenhuma ligação de saída. Mais uma vez, envia a mensagem para o nó mais próximo em direção ao ponto de encontro da árvore.

3. existe a entrada (*,G), contendo uma ou mais ligações de saída. Neste caso não é necessário propagar a mensagem, pois já foi enviado previamente.

Por seu lado, o **join-spt** foi reescrito, para passar a incorporar a métrica na troca de mensagens.Primeiro verifica-se se o nó já possui entrada para o cliente ao qual deseja ligar-se, como é possível verificar no código 4.4. No caso de não possuir uma entrada, é acrescentada a entrada e invocado o método **handle-setSPTBit**, que tem como função fazer um **prune** de um cliente na árvore partilhada. O valor do interface de entrada também alterado, passando a ser o próximo nó em direção ao cliente. Concluído este processo, é enviado o *join* para o nó seguinte no caminho mais curto para o cliente. O método **handle-setSPTBit** passa a ser invocado dentro

do **join-spt**, ao contrário de antigamente, onde era invocado quando recebia tráfego através da árvore centrada no cliente, pois é expectável que nem todos os servidores recebam pedidos do cliente (só recebe o servidor com melhor métrica). Na eventualidade de a entrada (C,G) já existir, é necessário verificar se o `RPT-Bit` está ativo, passando a inativo e alterando ainda o valor do interface de entrada. Envia para o nó seguinte um **join**, para que este possa fazer a sua verificação. Se o valor do `RPT-Bit` já estiver inativo, é verificada a lista de ligações saída. Caso esta esteja vazia, efetua um *join* para o vizinho.

Exemplo de código 4.4: Extrato do método **join-spt**.

```
1 set r [$node_ getReps $src $group]
2 set rpt 0
3 set wc 0
4 if { $r == "" } {
5       set iif [expr {($node_ == $src) ? -1 : [$node_ from-node-iface $src]}]
6       set spt 0
7       $self dbg "********* add entry <src:[$src id], group:$group, iif:$iif,
           spt:$spt, rpt:$rpt>"
8       $node_ add-entry $src $group $iif $spt $rpt
9       $self dbg "********* send join <group:$group, joinList:[$src id],
           wc:$wc, rpt:$rpt classId:$classId>"
10      $self handle-SetSPTBit  [$src id] $group $iif
11      $self send-ctrl "join" $src $group $mtr $wc $rpt $node_ $classId
12      # end of join if destination node
13      if { $node_ == $src } { set join_request_end 1 }
14 } else {
15 (...)
```

Ao receber um pedido de registo ou ligação, o método **recv-join** verifica a que tipo de árvore se pretendem ligar, árvore partilhada ou centrada num cliente. Esta seleção é realizada utilizando os campos do pacote *join*, `WC-Bit` e `RPT-Bit`, e caso estejam os dois ativos trata-se de um pedido para uma árvore partilhada senão para uma árvore centrada no cliente.

Numa ligação à árvore partilhada, a primeira verificação é a existência da entrada (*,G). Se a entrada não existir, é criada e a interface de entrada alterada para o próximo nó em direção ao *RP*. O próximo passo é verificar qual o interface por onde recebeu o pedido, adicionando

à lista de interfaces de saída e enviando um *join* até ao *RP*. Caso exista a entrada (*,G) dois cenários podem surgir, consoante a lista de interfaces de saída esteja vazia ou possua algum elemento. Ao ser detetado que a lista de interfaces de saída está vazia, esta é inicializada e um *join* enviado para o *RP*. Por outro lado, caso a lista já possua elementos, simplesmente é acrescentada uma entrada à lista de interfaces de saída.

Ao receber um pedido para a ligação a uma árvore centrada no cliente, o procedimento já sofre mais alterações em relação ao *multicast*, pois obriga a incluir a métrica. No caso de ser a primeira vez que chega uma mensagem, ou seja, não exista entrada, esta é criada (com o melhor interface de entrada para atingir o cliente), inicializando a lista ligada e propagando o **join** em direção ao cliente. Se já possuir uma entrada, acrescenta uma entrada à lista de interfaces de saída e verifica se a nova métrica é melhor que a anterior. O código 4.5 representa essa verificação. A métrica mínima é calculada, e se a métrica mínima for maior que a recebida (métrica atual é melhor), é enviada uma mensagem de **update** em direção ao cliente em questão, senão é sinal que a métrica não se altera e não é preciso avisar os nós seguintes.

Exemplo de código 4.5: Extrato do método **recv-join** para uma ligação centrada no cliente.

```
1 (...)
2 set min [$node_ get-minMetric [$to id] $group]
3 if { $mtr < $min} {
4       $self send-ctrl "update" $to $group $mtr $wc $rpt $node_
5 } else {
6         # just to annotate a successfull join...
7       set join_request_end 1 }
8 (...)
9 set outlink [$node_ iif2oif $iface]
10 $node_ add-metric [$to id] $group $outlink $mtr
11 (...)
```

4.4.3 Mensagens de abandono entre os nós

Se um servidor enviar uma mensagem de abandono, o método **leave-group** é invocando, verificando se este pretende abandonar um grupo ou deixar de receber pedidos de um cliente.

Consoante o número de argumentos passados, o **leave-group** efetua uma seleção de qual é o método mais apropriado para proceder à remoção. Se receber dois argumentos, o endereço de grupo e o agente em questão, é sinal que pretende abandonar todas as comunicações para o respetivo grupo, invocando o método **leave-rpt** e o **leave-spt**. Ao serem passados três argumentos, endereço de grupo, endereço de cliente e agente, é invocado o **leave-spt** para deixar de receber pedidos desse mesmo cliente. A receção de pedidos de abandono é realizada pelo método **recv-prune**

O método **leave-rpt** é utilizado para remover um servidor da árvore partilhada, começando por remover o valor do seu agente na lista de interfaces de saída. Se esta ficar vazia, envia um **prune** em direção ao *RP*, com o `RPT-Bit` e `WC-Bit` a 1, para sinalizar o seu abandono aos outros nós.

A remoção de um nó a uma árvore centrada no cliente, implica, mais uma vez, uma variável adicional, a métrica. O método invocado é o *leave-spt*, começando por eliminar a entrada da lista de interfaces de saída. Consoante o valor do `RPT-Bit`, duas situações podem acontecer. Se este estiver ativo, a entrada (C,G) nunca é apagada mesmo que a lista de interfaces de saída se encontre vazia. Apesar de manter a entrada na tabela, envia um **prune**, com `RPT-Bit` a 1 e `WC-Bit` a 0, em direção ao cliente. Se o `RPT-Bit` estiver inativo, procede-se à eliminação da sua entrada, enviando o *prune*, com `RPT-Bit` e `WC-Bit` a 0, pelo caminho mais curto até ao cliente.

Ao receber um pedido de abandono, o método **recv-prune** verifica o valor dos campos `RPT-Bit` e `WC-Bit`, podendo ocorrer três situações diferentes:

1. `RPT-Bit` e `WC-Bit` a 1 - significa que se trata de um *prune* de um nó na árvore partilhada, sendo o endereço do *RP* que está inserido na lista de *prunes*. Se existir uma entrada (*,G), o interface por onde recebeu a mensagem, é apagado da lista de interfaces de saída. Após a remoção deste elemento, verifica a lista de interfaces de saída e se esta ficar vazia, apaga a entrada e propaga a mensagem em direção ao *RP*.

2. `RPT-Bit` e `WC-Bit` a 0 - trata-se de um pedido de abandono de um nó na árvore centrada no cliente, sendo o endereço do cliente que aparece na lista de *prunes*.

3. `RPT-Bit` a 1 e `WC-Bit` a 0 - é utilizado quando um nó pretende deixar de receber pedidos de um cliente na árvore partilhada, contendo, na lista de *prunes*, o endereço do cliente

do qual não quer receber mais pedidos de localização.

Independentemente do valor dos campos, existem sempre duas situações que são verificadas. Se existir uma entrada (C,G), verifica-se o valor do `RPT-Bit` da entrada. Se o valor for igual a 1, inicialmente é calculada a métrica mínima, como é possível verificar no código 4.6. De seguida, obtém-se o valor do objeto a eliminar, procedendo à sua remoção. Volta-se a calcular o valor mínimo da métrica após a remoção. Se o valor mínimo for igual a **-1**, é sinal que a lista se encontra vazia e o *prune* com `RPT-Bit` e `WC-Bit` a 0, é propagado através da árvore centrada no cliente. Caso exista um novo mínimo, envia-se uma mensagem de **update** para atualizar o caminho até ao cliente. Se o valor for igual a 0, envia um **prune**. com `RPT-Bit` a 1 e `WC-Bit` 0, em direção ao *RP*.

Exemplo de código 4.6: Extrato do método **update**.

```
1 (...)
2 set min [$node_ get-minMetric [$to id] $group]
3 set outlink [$node_ iif2oif $iface]
4 $node_ rmv-metric [$to id] $group $outlink $mtr
5 set newMin [$node_ get-minMetric [$to id] $group]
6 if { $newMin == -1 } {
7       if { $node_ != $to } {
8           $self send-ctrl "prune" $to $group $mtr $wc $rpt $node_    }
9 } else {
10      if { $min != $newMin } {
11          $self send-ctrl "update" $to $group $newMin $wc $rpt $node_  } }
12 (...)
```

4.4.4 Mensagens de atualização entre os nós

O método **update** é invocado quando um nó pretende alterar a sua métrica em relação a um cliente. Tem como parâmetros o endereço de grupo, endereço do cliente, o agente e a sua métrica, estando representado no código 4.7. Através do endereço de grupo e de cliente identifica a entrada em questão, atualizando o valor da métrica de um determinado agente. Propaga então essa alteração de métrica para o nó seguinte em direção ao cliente.

Exemplo de código 4.7: Extrato do método **recv-update**.

```
1 TAP instproc update-group  { group agent mtr src } {
2     $self instvar node_ ns_
3     $self next $group
4     set rpt 0
5     set wc 0
6     $node_ update-metric [$src id] $group $agent $mtr
7     $self send-ctrl "update" $src  $group $mtr $wc $rpt $node_ }
```

Quando um nó recebe uma mensagem do tipo **update** (código 4.8), calcula o valor mínimo da lista ligada para, posteriormente verificar se este se alterou. De seguida, calcula a ligação de saída, através do interface por onde recebeu a mensagem e atualiza o seu valor na lista ligada. Se o valor sofreu alterações, reenvia a mensagem recebida em direção ao cliente, para que todos os nós possam efetuar a sua atualização. Na eventualidade de o valor mínimo ficar igual, não precisa de atualizar os restantes nós pois, eles já conhecem o melhor valor.

Exemplo de código 4.8: Extrato do método **recv-update**.

```
1 TAP instproc recv-update { from to group mtr iface wc rpt orig {classId ""} }
2 {
3     $self instvar  node_ ns_ id_ encaps_ pgenerator_
4     set min [$node_ get-minMetric [$to id] $group]
5     set outlink [$node_ iif2oif $iface]
6     $node_ update-metric [$to id] $group $outlink $mtr
7     set newMin [$node_ get-minMetric [$to id] $group]
8     if { $newMin < $min} {
9                 if { $node_ != $to } {
10                    $self send-ctrl "update" $to $group $mtr $wc $rpt $node_
11            } } }
```

4.4.5 Mensagens de Periódicas de Ligação e Notificações

Um servidor pode abandonar um grupo ou um cliente, requisitando especificamente, como explicado anteriormente. O problema é quando este apresenta uma falha e não consegue enviar

essa mensagem, não informando os nós da sua situação. Os pacotes continuarão a ser encaminhados para esse sistema terminal, até que o cliente se aperceba da falha e volte a emitir um pacote para descobrir novamente os servidores. Para prevenir o protocolo contra esta condição, o método **periodicJoinPrune** foi criado, sendo executado sucessivamente para manter a ligação do servidor. Cada entrada possui um campo relativo a um temporizador (`Timer`), tendo este o limite máximo para receber um *join* do servidor. A frequência de envio de mensagens pelo método **periodicJoinPrune** é menor que o temporizador associado, para permitir uma comunicação fluída. Um nó sabe que aconteceu uma falha num servidor, quando não recebe um **join** periódico antes de esgotar o tempo do temporizador associado à entrada, procedendo de seguida à sua eliminação.

A falha de um elemento da rede, nó ou a sua ligação, gera uma notificação em todos os nós da rede. Quando o protocolo *unicast* se apercebe de uma alteração da sua topologia, invoca o método **notify**, procedendo à atualização da tabela de encaminhamento. O método **notify**, implementado para o encaminhamento *anycast*, começa por verificar se alguma das ligações diretas ao nó falharam, quer sejam interfaces de entrada ou de saída. Na eventualidade de o estado de uma ligação falhar (ir a *down*), guarda-se a ligação num lista de ligações que falharam e ativa-se um sinalizador desta falha. O passo seguinte é verificar se esse sinalizador está ativo e caso seja uma ligação de saída, calcula-se o mínimo atual antes da remoção da ligação. De seguida, remove-se a ligação de saída da lista de ligações de saída e recalcula-se o valor mínimo. Se o valor mínimo atual for menor que o que possuí anteriormente, é necessário verificar se a lista de ligações de saída não ficou vazia. No caso de ficar vazia, é enviada uma mensagem **prune** pelo o novo interface de entrada do nó. Se o valor mínimo for menor que o anterior, mas a lista de ligações de saída não ficou vazia, é enviada uma mensagem de **update**, mais uma vez pelo novo interface de entrada. Para permitir escolher o interface de entrada por onde a mensagem é enviada, foi necessário modificar a função **send-ctrl**, para selecionar o correto interface.

O método **notify** necessita ainda de fazer mais algumas considerações. O próximo caso é verificar se o nó que recebeu a notificação é um servidor ou simplesmente um nó normal à comunicação. Para saber se o nó é um servidor, é necessário verificar se este possui agentes, e, caso este tenha algum agente, sinaliza-lo apropriadamente. O código 4.9 é executado de seguida, começando por verificar se o nó que recebeu a notificação é uma fonte. Caso não seja, verifica duas situações. A primeira situação verifica se o interface de entrada foi alterado e o nó possuí agentes, sendo neste caso atualizado o interface de entrada. Os únicos nós que conseguem

entrar neste caso são os servidores, sendo por isso invocado o método **periodicJoinPrune**, para os obrigar a recalcular as suas árvores. A segunda situação é quando o interface de entrada foi alterado e o nó não possuí nenhum agente, sendo este nó utilizado como intermediário entre os clientes e os servidores. Nesta situação, o nó envia uma mensagem de abandono pelo anterior interface de entrada e apaga a entrada da tabela de encaminhamento, aguardando pelas mensagens que irão ser invocadas pelos servidores, através do método **periodicJoinPrune**, para recalcular as suas entradas.

Exemplo de código 4.9: Extrato do método **notify**.

```
1 (...)
2 if {($node_ != $src)} {
3   set newiif [$node_ from-node-iface $src]
4     $self dbg "notify: found a <src:$src, group:$group> entry (
          oldiif:$oldiif, newiif:$newiif)"
5     if { $oldiif != $newiif && $flagPeriodic == 1} {
6         $self dbg "notify: changing iif from $oldiif to $newiif  for ([$src
              id],$group)"
7         $node_ change-iface $src $group $oldiif $newiif
8         $self periodicJoinPrune
9     } elseif { $oldiif  != $newiif && $flagPeriodic == 0 } {
10    set wc 0
11    set rpt 0
12    set min [$node_ get-minMetric [$src id] $group]
13    $self send-ctrl "prune" $src $group $min $wc $rpt $node_ "" $oldiif
14    $self dbg "********* send prune <group:$group, joinList:[$src id],
          wc:$wc, rpt:$rpt>"
15    $node_ del-entry $src $group
16    }
17 }
18 (...)
```

4.4.6 Aplicações implementadas

O *NS-2* possui vários tipos de aplicações, mas nenhum possuí um comportamento ideal para ser utilizado para o novo sistema especificado. Por isso, foi necessário derivar duas aplicações,

uma para ser utilizada nos clientes e outra para ser utilizada nos servidores.

A primeira aplicação a ser implementada foi a do cliente, derivando a classe `CBR-Traffic.cc`, para permitir a troca de mensagens de acordo com o novo protocolo. O funcionamento desta aplicação é bastante simples, sendo uma aplicação que debita pacotes, de acordo com uma taxa constante especificada, não podendo este ritmo ser alterado durante a sua execução. Esta aplicação utiliza o protocolo *UDP* para a sua comunicação, sendo a classe `udp.cc` a primeira a sofrer alterações. O método **recv** desta classe passou a permitir, quando selecionado esse comportamento, a notificação da aplicação aquando da receção de um pacote. À medida que os pacotes vão sendo recebidos, a aplicação vai recebendo a sua informação, o que permitiu implementar a característica da descoberta de servidores. Inicialmente a aplicação é colocada a funcionar a um ritmo bastante baixo, enviando um pacote e aguardando a sua resposta. Na eventualidade da resposta não chegar, é enviado novamente um pacote. Quando um pacote é recebido, a aplicação verifica se foi o primeiro pacote, para permitir um aumento do debito de pacotes. Para isso, invoca o método exemplificado no código 4.10, que para os pedidos de localização, ajusta a nova taxa e inicializa a comunicação.

Exemplo de código 4.10: Método **process_data**, em *tcl*, do **CBR−Traffic**.

```
1 Application/Traffic/CBR instproc process_data { } {
2   global ns
3   $self instvar node_ cbr_
4   puts "Vou fazer o stop::set_rate::start"
5   $ns at [$ns now] "$self stop"
6   $ns at [$ns now] "$self set rate_ 448000"
7   $ns at [$ns now] "$self start"
8 }
```

Os servidores foram implementados, derivando a classe `LossMonitor.cc`, que é um agente que recebe pacotes, calcula estatísticas e descarta os pacotes. O principal motivo para o interesse por uma classe como a `LossMonitor` são as estatísticas que consegue calcular, como pacotes perdidos, pacotes transmitidos ou mesmo número de *bytes* transmitidos. O seu comportamento padrão foi alterado, mantendo a receção de pacotes e o cálculo das estatísticas. O pacote deixa de ser descartado, passando a ser reenviado para o cliente *anycast*, como é possível verificar

pelo código 4.11. O endereço de origem passa a ser o do grupo *anycast*, como especificado anteriormente, o que permitirá acarretar enormes valias na segurança dos servidores. O endereço de destino passa a ser do cliente que efetuou o pedido e as portas de origem e destino também são trocadas. Criado o pacote de resposta, é agendado o seu envio através do escalonador de eventos, que o enviará para o próximo nó em direção ao cliente.

Exemplo de código 4.11: Extrato do método **recv** do **LossMonitor**.

```
1 if( echo_reply_ == 1 ){
2   hdr_cmn* h = hdr_cmn::access(pkt);
3   hdr_ip* ih = hdr_ip::access(pkt);
4   nsaddr_t src = ih->saddr();
5   nsaddr_t dst = ih->daddr();
6   hdr_ip::access(pkt)->saddr() = dst;
7   hdr_ip::access(pkt)->daddr() = src;
8   int tmp = hdr_ip::access(pkt)->sport();
9   hdr_ip::access(pkt)->sport() = hdr_ip::access(pkt)->dport();
10  hdr_ip::access(pkt)->dport() = tmp;
11  Scheduler::instance().schedule(target_, pkt, 0.0);
12 }
```

4.5 Resumo

O sistema especificado no capítulo 3 foi implementado com auxílio do simulador de rede *NS-2.35*[11], começando este capítulo por fundamentar a escolha, comparativamente com outro simulador[39]. O código implementado para o simulador escolhido é escrito em duas linguagens orientadas a objetos, o C++ para as tarefas amiudamente executadas (mais rápido) e o OTcl para o controlo de ações (mais flexível).

O código base utilizado na implementação do sistema é baseado numa implementação do *PIM-SM* existente[41], o que permitiu construir uma solução mais completa. Na segunda secção deste capítulo, é feita uma introdução ao estado do encaminhamento *multicast* no simulador escolhido e ao código do protocolo *PIM-SM*. Apesar do protocolo *PIM-SM* possuir um classificador e um agente para o protocolo *PIM-SM*, foi necessário adaptá-lo ao sistema especificado.

As classes `classifier-tap.cc` e `classifier-tap.h` são responsáveis pelo reenvio das mensagens entre nós, sendo escritas em C++. A implementação do agente do *TAP* ocorre na classe `TAP.tcl`, escrita em OTcl, sendo esta responsável pela troca de mensagens entre os nós, pela implementação do algoritmo e a construção das tabelas de encaminhamento.

O classificador implementado na abordagem base[41], reenvia os pacotes de acordo com os endereços de origem e o destino (grupo). Ao receber um pacote, o classificador verifica a sua tabela e encaminha os pacotes para todas as interfaces(nós) nele registados. Como no tráfego *anycast* é necessário escolher de acordo com os endereços de origem e destino mais a métrica, foi necessário acrescentar essa possibilidade. O método de encaminhamento *multicast* (para todos) foi conservado, sendo utilizado para as mensagens de descoberta de servidores. Foi necessário criar uma nova estrutura, com um par de campos interligados, métrica e respetivo nó. Assim, quando cada servidor se centrar no cliente, cria uma nova entrada, com a sua respetiva métrica. O encaminhamento passa a ser efetuado de acordo com o nó com melhor métrica, passando o cliente a comunicar com um único servidor. A figura 4.6 mostra os campos constituintes do novo classificador *anycast*.

Os métodos tradicionais do *PIM-SM* para a manutenção das árvores (partilhada e centrada) tiveram que sofrer alterações, para que passassem a contemplar a métrica nas suas mensagens. A classe `PIM.tcl` sofreu várias alterações nas suas principais funções (**join-group**, **leave-group**, **recv-join**, entre outras), para passar a tratar o tráfego segundo o encaminhamento *anycast*. Foi ainda necessário incluir um novo tipo de mensagem de atualização **update**, não disponível no protocolo base. Quando um nó recebe uma mensagem de atualização, atualiza a métrica do nó que originou a mensagem e verifica se o valor mínimo se alterou. Tendo este valor sido alterado, é enviado então uma mensagem para o nó seguinte, em direção ao destino, *RP* e/ou cliente(s). O processo descrito é repetido sucessivamente até que a métrica mínima não seja alterada ou chegue ao destino. De modo a proceder à criação da nova mensagem de atualização, foi necessário modificar a classe `packet.h` e `mcast_ctrl.cc`. A função **periodicJoinPrune** sofreu pequenas alterações para passar a atualizar a métrica mínima que possui, ao passo que a função **notify** sofreu grandes alterações, para permitir aumentar a disponibilidade da rede. A verificação da alteração dos interfaces de saída e de entrada, permitiu ao protocolo comutar rapidamente para uma nova árvore, diminuindo assim o tempo de resposta por parte dos servidores. Para finalizar, foi necessário implementar duas aplicações, para os clientes e servidores, de modo a simular o comportamento especificado no capítulo anterior. A

nova aplicação no cliente, efetua pedidos de descoberta de localização até receber a primeira mensagem de resposta de um servidor, passando de seguida a enviar pedidos para o servidor com a melhor métrica. A aplicação que corre nos servidores responde aos pedidos vindos dos clientes automaticamente, simulando o comportamento real de servidor, divulgando somente o seu endereço *anycast*. Em suma, nas duas últimas secções são apresentados os principais métodos e as suas respetivas alterações.

5

Testes e Resultados

O presente capítulo apresenta a validação do *Tree-based Anycast Protocol (TAP)* criado no âmbito da dissertação. O seu funcionamento é comprovado utilizando dois cenários diferentes. O primeiro cenário tem como objetivo retratar todos os casos abordados durante a especificação do sistema, utilizando a topologia apresentada nesse capítulo como base. O segundo cenário é o mais imprevisível, contendo uma topologia com 18 nós. A localização do *RP*, dos três clientes e dos quatro servidores é gerada aleatoriamente. O conjunto de eventos é enumerado no final da próxima secção, sendo os seus resultados analisados. Finalmente, é realizado um teste comparativo com um *PIA*, relativamente à distância entre o cliente e o servidor escolhido.

5.1 Validação da Implementação

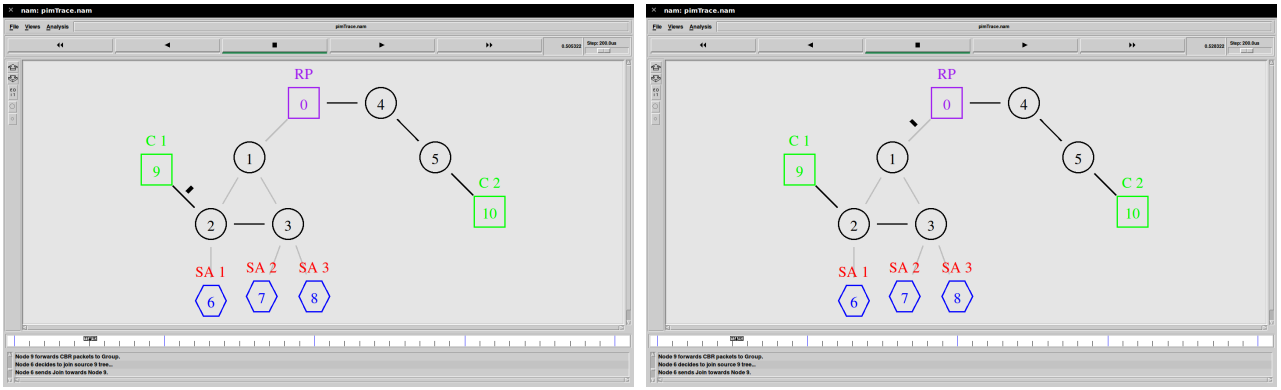
A validação da implementação do *TAP* foi testada no *NS-2* utilizando duas *scripts* diferentes. O comportamento do protocolo foi aferido ao longo desta secção, através de um cenário em que o comportamento é expectável e de outro onde as diferentes variáveis são geradas aleatoriamente.

5.1.1 Cenário previsível

O cenário, utilizado durante a especificação do sistema no capítulo 3, foi o primeiro a ser implementado, tendo sido elaborado um conjunto de eventos que permitem observar todas as situações abordadas. Foi elaborada uma *script* de simulação, disponível no anexo C.1, que com o auxílio de uma aplicação de suporte do *NS-2*, o *NAM*[1], permite ilustrar o comportamento do *TAP*. O *NAM* é uma ferramenta de animação, que permite criar topologias e visualizar a animação dos pacotes durante o seu encaminhamento.

[1] `http://www.isi.edu/nsnam/nam/`

Sempre que um cliente pretende comunicar com o grupo de servidores *anycast*, deve começar por enviar uma mensagem de descoberta de servidores. A aplicação desenhada contempla essa primeira fase da comunicação, como é possível observar na figura 5.1. Inicialmente, o `Cliente 1 (C1)` envia o pedido de localização encapsulado para o `Nó 2` (figura 5.1(a)), sendo este encaminhado para o próximo nó em direção ao *RP*, o `Nó 1`. O *RP* recebe o pedido do `C1` através do `Nó 1` (figura 5.1(b)), desencapsulando o pedido de localização.



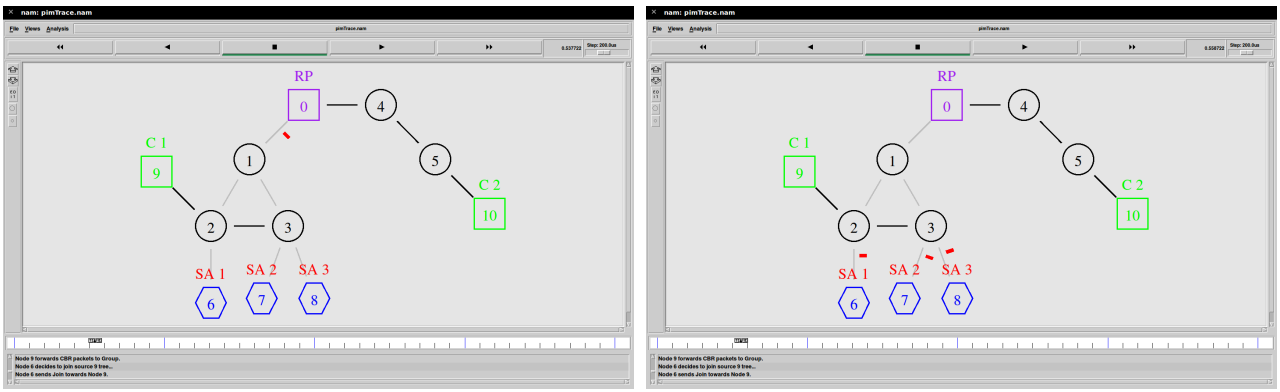
(a) Envio do pedido do `C1` para o *RP* através do `Nó 2`.

(b) Receção do pedido por parte do *RP*.

Figura 5.1: Processo de descoberta dos servidores - do `C1` até ao *RP*.

O *RP* contém as localizações de todos os servidores do grupo, sendo por isso um ponto importante no protocolo. Quando o *RP* recebe o pedido de localização, desencapsula o pacote enviando-o para todos os servidores, como é possível observar na figura 5.2. Começa por enviar o pacote de localização para o próximo nó em direção aos servidores (figura 5.2(a)), o `Nó 1`. Este, ao receber o pacote verifica a localização dos servidores, através da sua tabela de encaminhamento e repara que possui dois diferentes interfaces de saída para o grupo. Envia o pacote de localização em direção ao `Nó 2` e replica-o para enviar para o `Nó 3`. A figura 5.2(b) ilustra o momento onde os servidores recebem os pedidos de localização através dos seus vizinhos. O `SA 1` recebe através do `Nó 2` e os servidores `SA 2` e `SA 3` recebem pelo `Nó 3`.

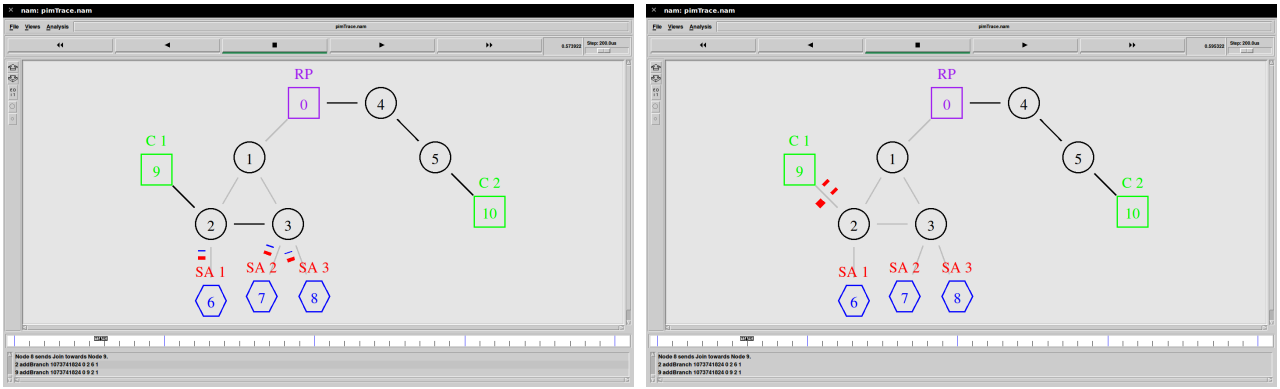
Os servidores do grupo, ao receberem o pedido de localização, adicionam uma entrada para o `C1` e enviam duas mensagens até este. A primeira mensagem é de **join** ao cliente, pacote a azul na figura 5.3(a), que irá criar a árvore centrada no cliente. A segunda mensagem é um pacote de confirmação da receção do pedido, pacote a vermelho na figura 5.3(a), que permitirá à aplicação do cliente começar a efetuar pedidos quando receber a primeira resposta. Todos os servidores anunciam a mesma métrica, pois nenhum se encontra a responder a pedidos, sendo



(a) Envio do pedido do *RP* para todos os Servidores. (b) Receção do pedido por parte dos Servidores.

Figura 5.2: Processo de descoberta dos servidores - do *RP* até aos Servidores.

escolhido o melhor servidor através da primeira mensagem a ser recebida. A mensagem entre o `SA 1` e o `C1` só passa pelo `Nó 2`, ao passo que as mensagens do `SA 2` e `SA 3` passam pelo `Nó 3` e de seguida pelo `Nó 2`, até chegar ao cliente. A figura 5.3(b) permite verificar, que apesar de ainda não ter recebido a confirmação da receção do pedido, por parte dos servidores `SA 2` e `SA 3`, o `C1` começa a efetuar pedidos, diminuindo assim o tempo da comunicação.



(a) Envio da mensagem **join** e do pacote de resposta por parte dos Servidores. (b) Início da comunicação entre o `C1` e o `SA 1`.

Figura 5.3: Criação da árvore centrada no `C1`.

O processo anteriormente é repetido para todos os clientes que desejam começar uma nova comunicação ou que deixem esgotar o tempo da entradas da árvore centrada em si, sendo obrigados a reiniciar a construção da árvore. Quando o `Cliente 2 (C2)` começar a comunicar com o grupo, enviará um pedido para o *RP*, que por sua vez encaminhará a mensagem até aos servidores. A diferença é que o `SA 1` como se encontra a comunicar com o `C1`, aumentará a sua métrica, o que fará com que passe a comunicar com um dos outros dois servidores. Mais

uma vez, como as métricas anunciadas por SA 2 e SA 3 são iguais, o servidor escolhido é o que enviou a mensagem primeiro, no caso do exemplo do SA 2. A figura 5.4 demonstra as duas comunicações ativas, de C1 para SA 1 e de C2 para SA 2.

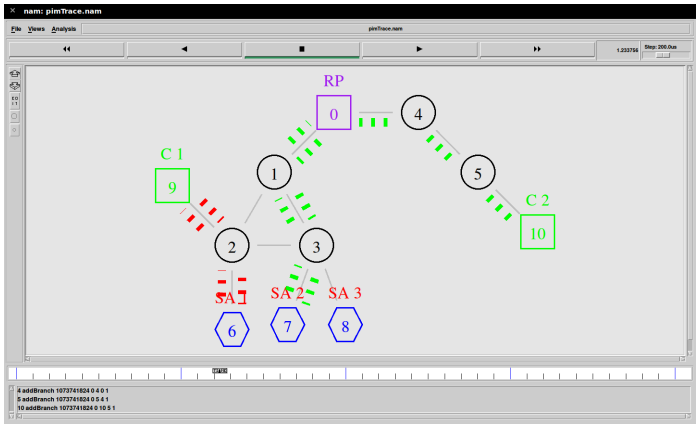
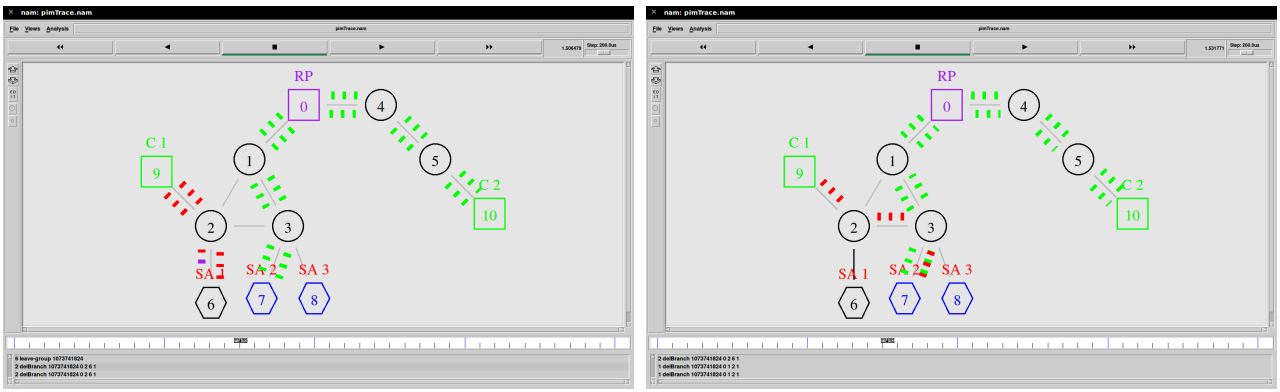


Figura 5.4: Comunicação dos clientes com o Servidores.

O abandono de um servidor é um dos aspetos críticos da comunicação. O protocolo implementado permite que apesar do abandono de um servidor, que esteja a receber pedidos, os pacotes sejam automaticamente enviados para outro servidor, de modo a maximizar a eficiência do grupo. A figura 5.5 retrata o abandono do servidor do SA 1 do grupo. Quando um servidor pretende abandonar um grupo, envia uma mensagem de **prune** para o próximo nó (mensagem a roxo na figura 5.5(a)), informando que pretende abandonar tanto as árvores centradas em clientes, como a árvore partilhada. O vizinho do SA 1, o Nó 2, ao receber o **prune** elimina o interface de saída. Como o Nó 2 possui outros interfaces de saída em relação ao C1, não necessita de propagar a mensagem, diminuindo assim o número de mensagens de controlo e permitindo que o fluxo de pedidos continue, sendo comutados automaticamente para o SA 2 (figura 5.5(b)). O Nó 2 propaga ainda as mensagem de **prune** em relação à árvore centrada no C2 e à árvore partilhada até ao Nó 1. Como o Nó 1 possui melhores métricas para as duas entradas, não necessita de propagar as mensagens até ao *RP*.

O abandono do SA 1, faz com que os pedidos do C1 e do C2, sejam respondidos pelo SA 2, como se pode observar na figura 5.6(a). Isto acontece, porque o primeiro **join** efetuado pelo SA 2 ainda não tinha nenhuma comunicação ativa, ou seja, a métrica anunciada é diferente da atual. Quando começa a receber pedidos dos dois clientes, este opta por atualizar a métrica em relação a C1. Na figura 5.6(a) é possível observar um pacote azul, que é a mensagem de **update** em relação ao C1. O Nó 3 ao receber esta mensagem faz atualização da métrica do SA

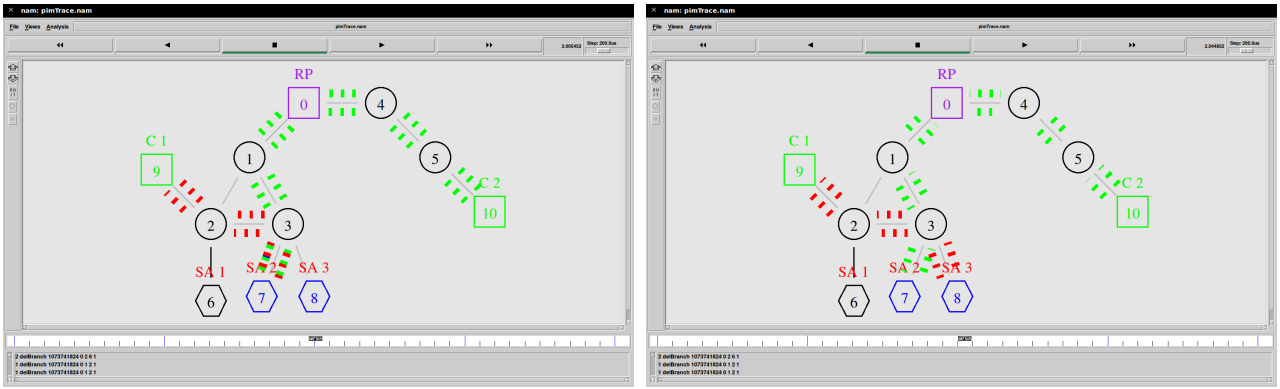


(a) Envio do mensagem **prune** por parte do SA 1.

(b) Pedidos do C1 que era atendido pelo SA 1, são reencaminhados para o SA 2.

Figura 5.5: Abandono do grupo por parte do SA 1.

2 e passa a enviar os pacotes para o SA 3, o servidor com melhor métrica para o C1, como é possível observar na figura 5.6(b). De realçar, que apesar do abandono do SA 1 e da posterior comutação de SA 2 para SA 3, não houve perda de pacotes, o que permite atestar a eficiência do protocolo.



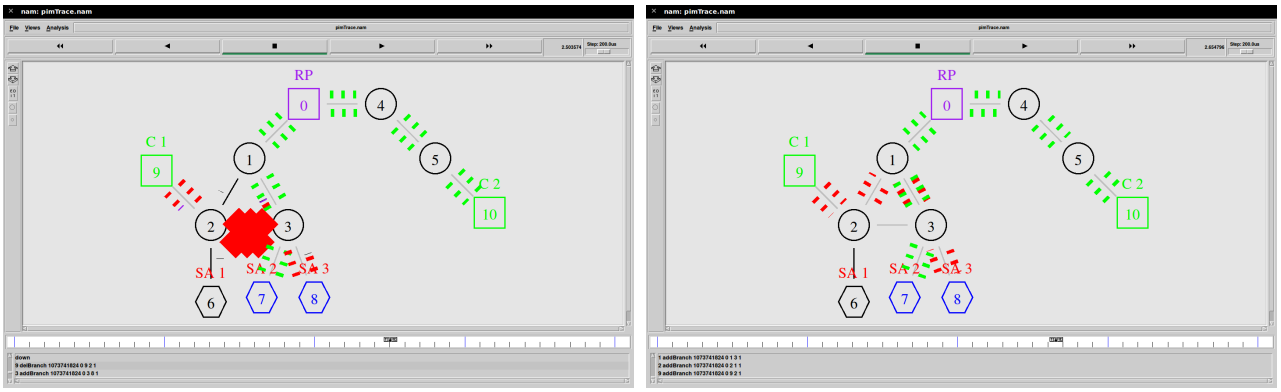
(a) Envio do mensagem **update** por parte do SA 2.

(b) Pedidos do C1 são reencaminhados para o SA 3.

Figura 5.6: Atualização da sua métrica por parte do SA 2.

A única situação onde acontece perdas de pacotes é quando uma ligação falha entre dois nós, estando esses dois nós a servir de intermediários entre um cliente e um servidor. A figura 5.7(a) retrata o momento em que a ligação entre o Nó 2 e o Nó 3 falha, sendo possível observar a vermelho, os pacotes perdidos. Quando a falha acontece, o método **notify** é invocado, começando por informar o C1 da falha da ligação, parando este, momentaneamente, de enviar pedidos, pois não possui um servidor disponível noutra área. Se o SA 1 estivesse ativo, receberia automaticamente os pedidos, nunca sendo interrompida a comunicação. Os servidores ao aperceberem-se da falha, voltam a criar a árvore centrada no C1, mas desta vez através do

caminho pelo Nó 1 e não diretamente pelo Nó 2. A figura 5.7(b) mostra a comunicação entre o C1 e o SA 3 através do novo percurso.



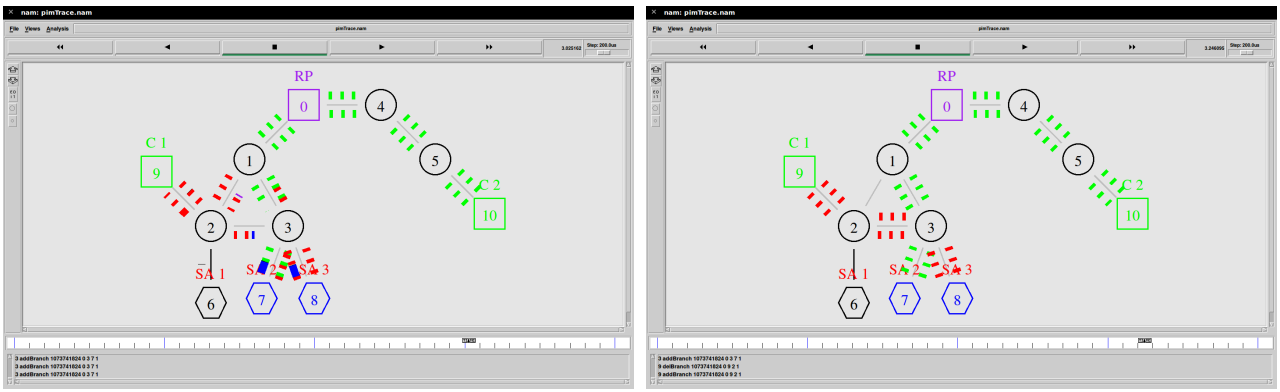
(a) Atualização das entradas e dos seus vizinhos por parte de vários nós devido à falha.

(b) Recuperação da comunicação com o C1.

Figura 5.7: Falha da ligação entre o Nó 2 e o Nó 3.

O protocolo proposto foi desenhado para que, quando uma ligação nova fosse adicionada, os nós, que conseguem encontrar um caminho melhor que o atual, reconstruam as suas árvores. Este é um ponto sensível do protocolo, faltando realizar mais testes ao desempenho desta abordagem, para verificar se realmente existe um ganho considerável. Não obstante a esta consideração, o protocolo implementando pressupõe essa possibilidade. A figura 5.8(a) permite ilustrar o funcionamento do protocolo quando a ligação entre o Nó 2 e Nó 3 volta a ficar ativa. Quando o Nó 3 se apercebe que tem um melhor caminho para o C1 do que o atual, envia uma mensagem de **join** pelo melhor caminho, para passar a receber tráfego por esse interface. É ainda necessário destruir a árvore antiga para C1, sendo enviado um **prune** para o Nó 1. Na figura 5.8(b) é possível verificar que a comunicação entre o C1 e o SA 3 passa a ser encaminhada pelo melhor caminho disponível.

O cenário descrito é bastante explícito, mas tem como desvantagem o facto de toda a implementação ser pensada com este cenário em mente. É necessário realizar um cenário imprevisível, onde a localização dos elementos do grupo e dos clientes seja aleatória, para poder comprovar a exequibilidade da implementação.



(a) Atualização das entradas e dos seus vizinhos por parte de vários nós devido à nova ligação.

(b) Recuperação do melhor caminho para comunicar com o `C1`.

Figura 5.8: Ligação entre o `Nó 2` e o `Nó 3` volta a ficar disponível.

5.1.2 Cenário aleatório

Uma nova *script* de simulação foi elaborada, disponível no anexo C.2, com uma topologia com 18 nós com ligações simétricas entre si, como é possível observar na figura 5.9. Na simulação, são criados três clientes e quatro servidores, bem como um *RP*. A posição destes elementos é gerada, aleatoriamente, para cada simulação.

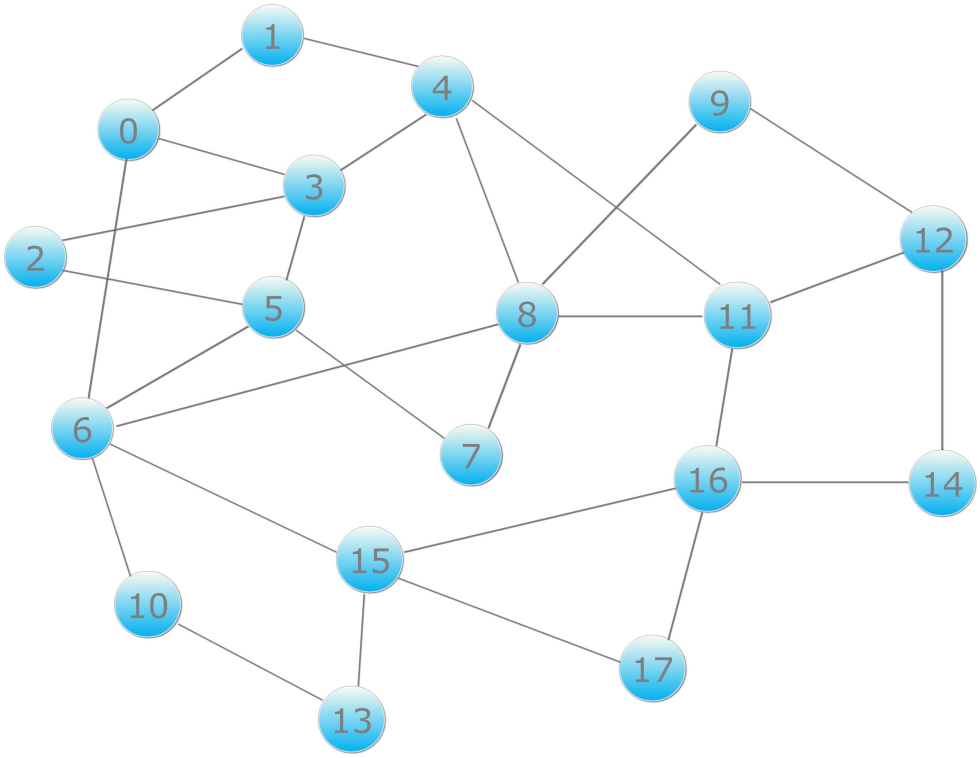


Figura 5.9: Topologia implementada com 18 nós.

Os valores das métricas para cada cliente também são geradas aleatoriamente, tendo como valor um número dentro do intervalo [1,100]. Para estudar o comportamento do protocolo, foram escalonados os seguintes eventos, em diferentes instantes de tempo (IT):

1. Os servidores ligam-se ao *RP* (IT= 0.1 segundos).
2. O cliente 1 inicia o processo de descoberta de servidores. Posteriormente, os servidores ligam-se ao cliente 1, iniciando a troca de pacotes (IT = 2.0 segundos).
3. Todos os servidores atualizam a sua métrica em relação ao cliente 1 de forma aleatória (IT = 4.0 segundos).
4. O cliente 2 inicia o processo de descoberta de servidores. De seguida, os servidores ligam-se ao cliente 2, iniciando a troca de pacotes (IT = 6.0 segundos).
5. O servidor 1 decide abandonar o grupo. Efetua o **prune** da árvore partilhada e das árvores centradas nos clientes 1 e 2 (IT = 8.0 segundos).
6. O cliente 1 termina a sua ligação (IT = 10.0 segundos). Todos os servidores (ainda ativos) fazem **prune** da árvore centrada no cliente 1 (IT = 11.0 segundos). O cliente 3 inicia o processo de descoberta de servidores. Depois, os servidores (ainda ativos) ligam-se ao cliente 3, iniciando a troca de pacotes (IT = 10.0 segundos).
7. Falha a ligação entre dois nós. Os nós são escolhidos aleatoriamente (IT = 12.0 segundos).
8. A ligação que falhou no evento anterior, volta a ficar disponível (IT = 14.0 segundos).
9. Os servidores 2 e 3 abandonam o grupo. Efetuam o **prune** da árvore partilhada e das árvores centradas nos clientes 2 e 3 (IT = 16.0 segundos).
10. Os cliente 2 e 3 terminam as suas ligações. O servidor 4 faz **prune** das árvores centradas nos clientes 2 e 3 (IT = 18.0 segundos).
11. Para terminar, o servidor 4 decide abandonar o grupo, efetuando o **prune** da árvore partilhada(IT = 19.5 segundos).

A simulação aleatória foi realizada 100 vezes, para verificar o funcionamento do *TAP*. O gráfico representado na figura 5.10 representa a média do número de mensagens de controlo recebidas (**join**, **prune** e **update**), ao longo dos diferentes eventos. É possível verificar que,

quando a ligação falha entre dois nós, nos 12 segundos, existe claramente um aumento significativo de troca de mensagens. Do mesmo modo, quando essa ligação volta a ficar disponível, aos 14 segundos, a troca de mensagens de controlo aumenta para encontrar novamente o melhor caminho. É ainda possível afirmar, que a troca de mensagens de controlo só acontece de acordo com o previsto, permitindo atestar o correto funcionamento do novo protocolo especificado.

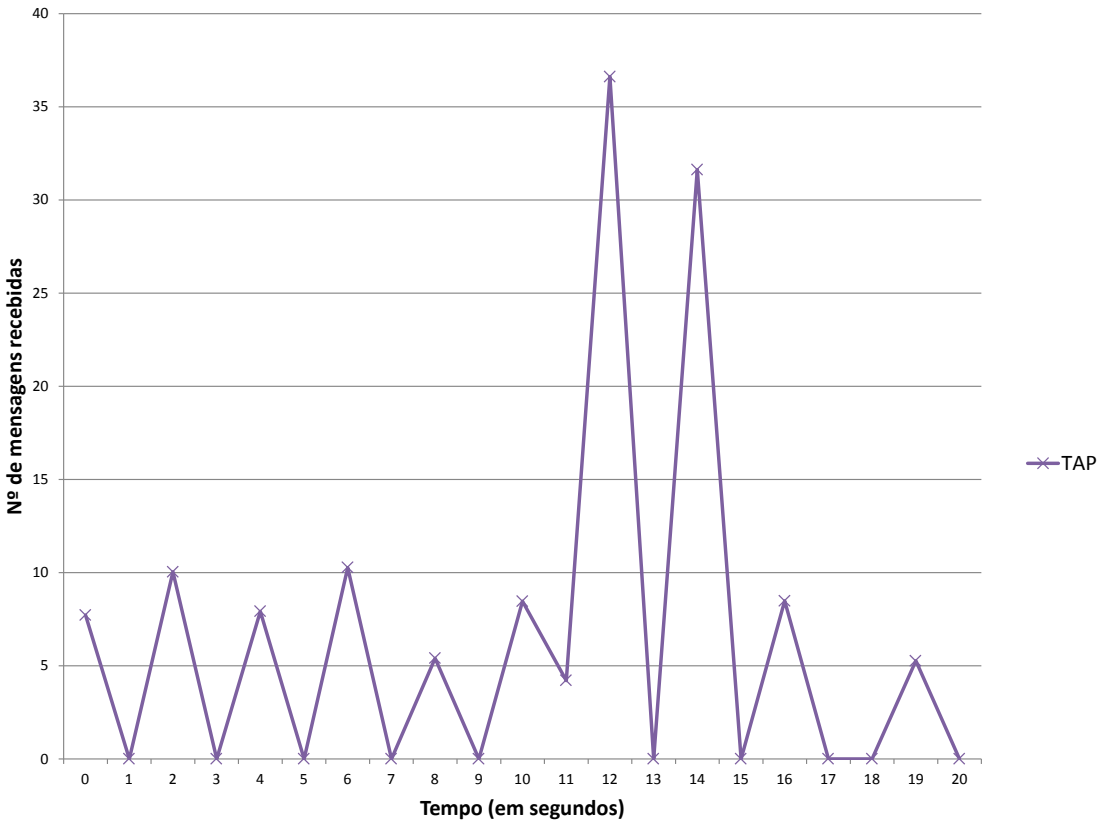


Figura 5.10: Média das mensagens de controlo recebidas no *TAP* durante as 100 simulações aleatórias.

Os resultados obtidos das 100 simulações permitiram ainda comparar o número de mensagens de controlo do *TAP* com o número de mensagens trocadas entre os clientes e os servidores (*CBR*) e o número de mensagens de controlo do tráfego de *unicast* (*RTPROTO*). O gráfico na figura 5.11 permite verificar que o número de mensagens de controlo do *TAP* (a roxo no gráfico) é bastante inferior ao número de mensagens trocadas entre os clientes e os servidores (a vermelho no gráfico), o que permite afirmar que o impacto das mensagens de controlo do novo protocolo, especificado no tráfego da rede, não é relevante. Caso o número de mensagens de controlo do *TAP* (a roxo no gráfico) seja comparado com o número de mensagens de controlo do tráfego *unicast* (a verde no gráfico), é possível verificar que o número de mensagens *unicast* é sempre

superior ou, em alguns eventos, igual as do *TAP*.

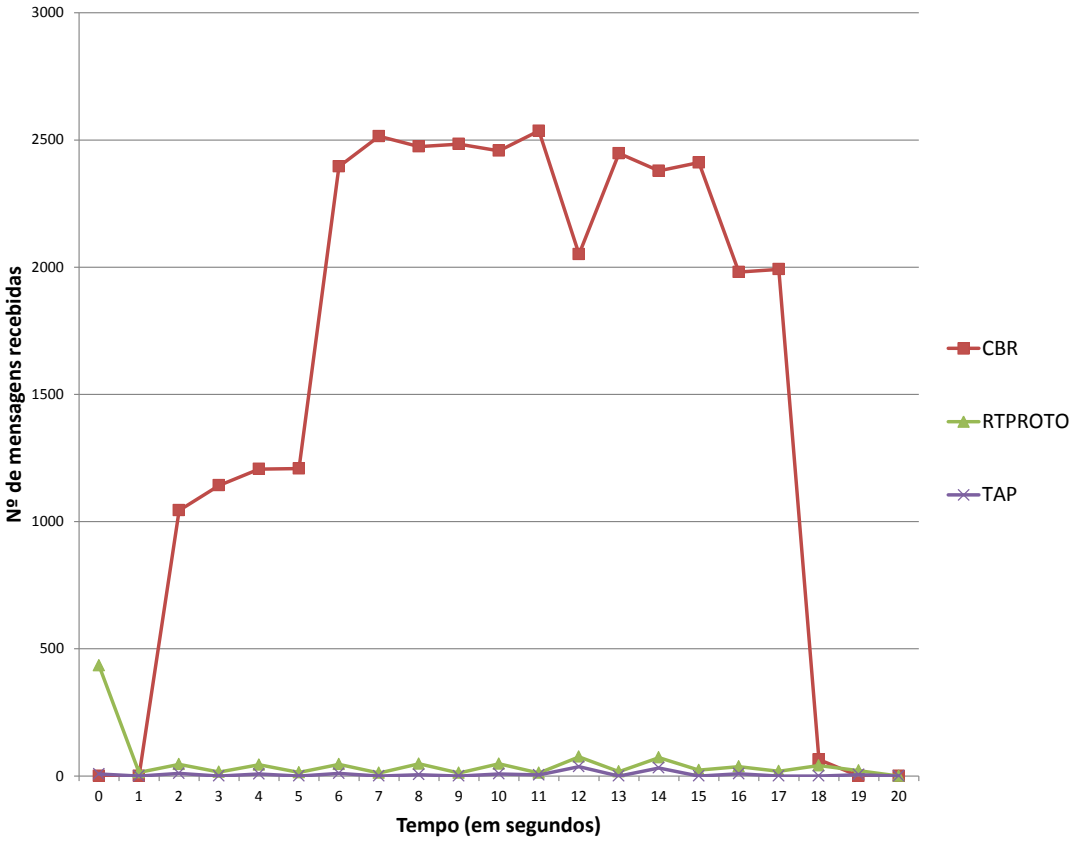


Figura 5.11: Comparação da média da mensagens de controlo recebidas de todos protocolos durante as 100 simulações aleatórias.

Por fim, é possível verificar no gráfico da figura 5.11 que, quando acontece uma falha, aos 12 segundos, existe um decréscimo no número de mensagens recebidas entres os clientes e os servidores. Este decréscimo acontece quando a ligação que vai a baixo é a que é utilizada para a comunicação entre um cliente e um servidor, o que já era expectável. O que é importante de realçar é que apesar deste decréscimo momentâneo, o *TAP* consegue encontrar outro caminho para a comunicação, sendo possível verificar que o número de mensagens recebidas da comunicação entre o cliente e o servidor volta a aumentar.

Apesar dos resultados serem positivos, o teste ao desempenho não deve ficar por aqui. Futuramente, devem ser realizados outros testes, com diferentes topologias e medidos outros dados do desempenho, como o tempo de criação de árvores.

5.2 Análise dos resultados em relação ao *PIA-SM*

Um dos objetivos definidos no início da dissertação era o de comparar o *TAP* com outros protocolos *anycast*. Devido à inexistência de implementações para o *NS-2* dos protocolos estudados, foi necessário implementar um destes, para permitir efetuar uma comparação com o *TAP*. O *PIA-SM*[25] foi o protocolo escolhido, principalmente por este também utilizar uma abordagem baseada no *PIM-SM*. Embora o protocolo tenha sido implementado com sucesso, não houve tempo para desenvolver uma aplicação que tivesse um comportamento como especificado[25]. Esta limitação não permitiu efetuar grandes testes entre os dois protocolos, ficando a faltar a realização de um melhor conjunto de testes no futuro.

No entanto, foi possível realizar uma comparação entre os dois protocolos, medindo a distância do servidor escolhido até ao cliente. A topologia presente na figura 5.9 foi utilizada como cenário para os testes, sendo posicionados aleatoriamente nesses nós um *RP*, um cliente e quatro servidores. O cenário foi simulado 100 vezes para os dois protocolos, *TAP* e *PIA-SM*, sendo o gráfico presente na figura 5.12 o seu resultado.

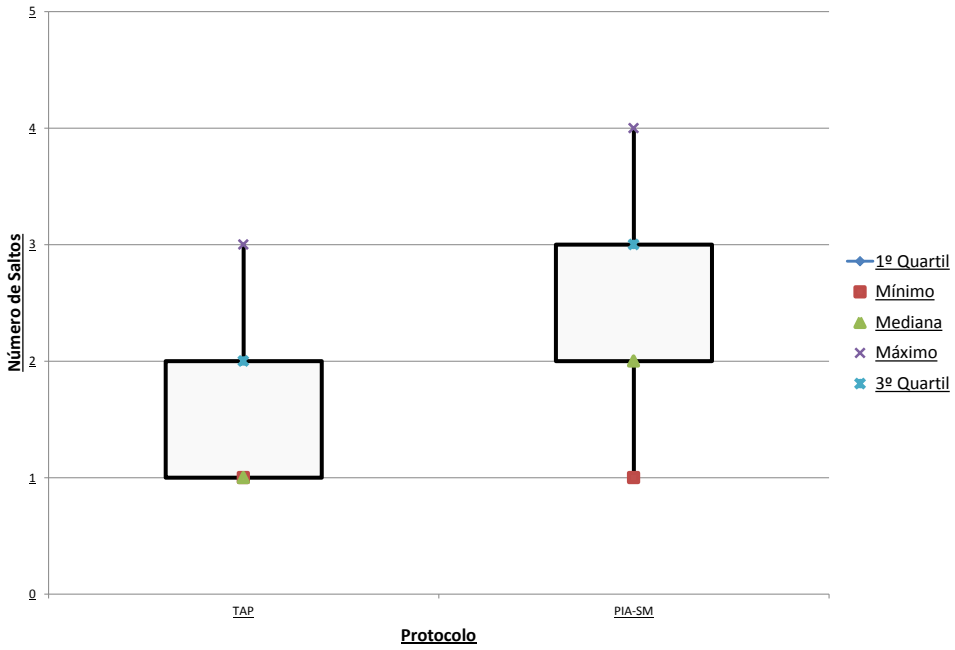


Figura 5.12: Distância entre o cliente e o servidor escolhido para os dois protocolos (*TAP* e o *PIA-SM*).

Ao observar o gráfico na figura 5.12 é possível afirmar que o protocolo *PIA-SM* nem sem-

pre escolhe o servidor mais próximo da localização. Uma importante conclusão a retirar das simulações é que o valor das localizações centrais das amostras é menor no *TAP*, o que atesta a eficiência do novo protocolo em relação ao *PIA-SM*. Contudo, é necessário para trabalho futuro realizar um conjunto de testes exaustivos ao *TAP*, obtendo novos resultados em relação ao *PIA-SM*.

5.3 Resumo

A implementação do *TAP* foi testada ao longo do presente capítulo, de modo a sustentar o protocolo concebido. Os testes foram realizados no simulador de rede *NS-2.35*[11], começando o capítulo por simular o cenário elaborado no capítulo 3, relativo à especificação do sistema. Esse cenário permitiu demonstrar o comportamento do *TAP* para as seguintes circunstâncias - pedido de descoberta de servidores, pedido de abandono de um servidor, pedido de atualização da métrica por parte de um servidor, falha de uma ligação entre dois nós na rede e adição ou recuperação de uma ligação entre dois nós na rede. Como o cenário elaborado para a realização dos testes anteriores era muito previsível, foi desenhado um novo cenário, para testar o funcionamento do *TAP*. Foi criada uma topologia com 18 nós, contendo um *RP*, três clientes e quatro servidores. Estes elementos foram dispostos pela topologia aleatoriamente, sendo este teste repetido 100 vezes para comprovar o seu funcionamento. Por fim, foi realizada uma comparação do novo protocolo com o *PIA-SM*[25], relativamente ao servidor que é escolhido quando um cliente quer efetuar uma comunicação. Mais uma vez, foi utilizada uma topologia com 18 nós, mas contendo apenas um *RP*, um cliente e quatro servidores, sendo simulado as suas posições 100 vezes.

A partir dos testes realizados é possível afirmar o correto funcionamento do *TAP*. O novo protocolo só apresenta algumas perdas de pacotes quando uma ligação entre um cliente e um servidor falha mas, mesmo nesse caso o número de pacotes perdidos não é significativo, sendo rapidamente recuperada a comunicação. É ainda possível afirmar que o número de mensagens de controlo introduzidas pelo *TAP* não é significativo quando comparado com o número de mensagens trocadas entre os clientes e os servidores. Quando comparado com o *PIA-SM*, o *TAP* apresenta melhorias significativas na escolha do melhor servidor. Para trabalho futuro, fica a necessidade de um maior conjunto de testes ao desempenho do *TAP* e à sua comparação com o *PIA-SM*.

6

Conclusão

A presente dissertação é concluída neste capítulo. Ao longo do mesmo, é realizada uma reflexão sobre os objetivos propostos e enunciadas as principais contribuições. É criada ainda uma diretiva para investigações futuras, identificando os aspetos, que devido à limitação de tempo, não foram possíveis de alcançar.

6.1 Objetivos Alcançados e Contribuições

O trabalho elaborado, ao longo da dissertação, tinha como finalidade estudar, aprofundadamente, o paradigma de comunicação *anycast*, fazendo um levantamento das soluções existentes para o problema do encaminhamento *anycast*. É proposto um novo protocolo de encaminhamento *anycast* baseado no protocolo *PIM-SM*, denominado *Tree-based Anycast Protocol (TAP)* sendo o seu funcionamento avaliado, recorrendo ao *Network Simulator 2 (NS2.35)*[11]. Considerando os objetivos inicialmente propostos, concluí-se:

- O paradigma de comunicação *anycast*, em relação ao encaminhamento ao nível da rede, ainda está pouco desenvolvido, tendo sido realizado um levantamento dos principais acontecimentos da sua história e as suas limitações. Impunha-se ainda, a implementação do encaminhamento *anycast* num emulador de rede, para comprovar o estado real dos seus protocolos. Foi possível concluir que o encaminhamento intra-domínio *anycast* é atualmente uma realidade, ao passo que, o encaminhamento inter-domínio *anycast* ainda não é funcional, principalmente por não possuir um protocolo de encaminhamento normalizado. O principal foco da dissertação foi a elaboração de um protocolo de encaminhamento *anycast* ao nível de rede.

- Um dos objetivos fundamentais do trabalho realizado passava por realizar um estudo aprofundando das propostas para o encaminhamento *anycast* inter-domínio[17][18][19][21][25][27],

apresentando, individualmente, cada uma das propostas. Ao longo de cada proposta foram identificados os problemas que estas pretendem resolver e os que deixam em aberto, sendo realizado uma comparação final entre todas as propostas.

- A especificação do novo protocolo foi baseada no protocolo *PIM-SM*, um protocolo *multicast* para redes com nós dispersamente distribuídos, semelhante ao que acontece no *anycast*. A principal vantagem do *TAP*, em relação às propostas estudadas, é o facto de o cliente comunicar realmente com o servidor mais próximo do seu local, mantendo-se fiel ao encaminhamento *anycast*. A segurança e o nível de disponibilidade da rede também são aumentados, devido às características do protocolo. O facto do *TAP* necessitar de um endereçamento distinguível do *unicast*, provocando uma alteração da norma[5], é a principal desvantagem deste.

- O protocolo proposto foi implementado recorrendo ao simulador de rede *NS-2*[11], sendo apresentadas as razões para a sua escolha, no capítulo 4. O maior problema da implementação foi a curva de aprendizagem necessária para trabalhar com as linguagens C++ e OTcl[1] (em especial a última), quer na implementação quer no *debug* do código. Não obstante a este entrave, o protocolo foi implementado com sucesso, utilizando uma implementação do *PIM-SM* existente[41]. Foi criado um cenário com 18 nós, sendo distribuídos aleatoriamente o *RP*, os três clientes e os quatro servidores, permitindo confirmar assim, o correto funcionamento da solução proposta.

- O último objetivo proposto era de comparar o trabalho realizado com abordagens semelhantes. O principal obstáculo a essa comparação, foi o facto de não existirem implementações públicas dos protocolos estudados, sendo necessário implementar o código de outra proposta, para ser possível a comparação. Apesar de, esta implementação já não fazer parte do âmbito da dissertação, foi escolhida uma abordagem[25] para ser implementada, tendo sido desenvolvido o seu protocolo para o *NS-2*. Devido à limitação do tempo, não foi possível implementar uma aplicação que funcione de acordo com as especificações da abordagem[25], não tendo sido possível submete-la a rigorosos testes. Apesar desta contrariedade, foi no entanto possível comparar as duas abordagens num cenário aleatório, onde a distância entre o cliente e o servidor escolhido é medida. É possível afirmar a vantagem do *TAP* na escolha do melhor servidor, mas é necessário efetuar um conjunto de testes mais aprofundados, que comprovem a eficiência do protocolo proposto.

[1] `http://www.cs.sjsu.edu/faculty/melody/NS2_Tutorial.htm`

A principal contribuição do trabalho realizado foi a especificação e implementação de um novo protocolo de encaminhamento *anycast* no *NS-2*. Ao longo do período da investigação desta dissertação, foi submetido e aceite um artigo[43] à *12º Conferência sobre Redes de Computadores (CRC 2012)*, que se revela um contributo adicional.

6.2 Trabalho Futuro

O trabalho realizado deixa em aberto algumas questões por explorar, como explicado ao longo do documento. O tempo disponível não foi suficiente para responder a todas as questões, mas permitiu reafirmar a validade do encaminhamento *anycast* ao nível da rede.

Uma das questões que necessita, claramente, de um estudo aprofundado é a métrica relativa a cada servidor. A forma como esta é calculada e os parâmetros utilizados para obter o seu valor, devem ser refletidos.

A segurança dos servidores é um dos aspetos fulcrais para cativar os futuros utilizadores do protocolo. Apesar de já possuir um mecanismo aliciante, o facto de o endereço *unicast* do servidor nunca ser conhecido, é necessário investigação posterior, nomeadamente em relação à autenticação dos servidores em relação ao *RP*.

A realização de novos testes para comprovar a eficiência do protocolo proposto é imperativa, nomeadamente em relação aos tempos envolvidos na criação das árvores de encaminhamento. Deve ser criada uma ferramenta que permita medir adequadamente o protocolo criado, nas suas mais importantes variáveis. A comparação com outra abordagem também carece de mais testes, sendo necessário terminar a aplicação correspondente à outra proposta[25].

Finalmente seria interessante, caso existisse um bom desempenho nos testes adicionais, a implementação do protocolo numa rede real, para apurar a funcionalidade fora do paradigma académico.

Referências

[1] "Downloads NS-2 estatísticas." http://sourceforge.net/projects/nsnam/files/stats/timeline?dates=2012-08-27+to+2012-09-27. Acedido: 30/09/2012.

[2] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification." RFC 2460 (Draft Standard), Dec. 1998. Updated by RFCs 5095, 5722, 5871, 6437, 6564.

[3] I. Keslassy, S.-T. Chuang, K. Yu, D. Miller, M. Horowitz, O. Solgaard, and N. McKeown, "Scaling internet routers using optics," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '03, (New York, NY, USA), pp. 189–200, ACM, 2003.

[4] C. Partridge, T. Mendez, and W. Milliken, "Host Anycasting Service." RFC 1546 (Informational), Nov. 1993.

[5] R. Hinden and S. Deering, "IP Version 6 Addressing Architecture." RFC 1884 (Historic), Dec. 1995. Obsoleted by RFC 2373.

[6] S. Weber and L. Cheng, "A survey of anycast in ipv6 networks," *Comm. Mag.*, vol. 42, pp. 127–132, Jan. 2004.

[7] G. Held, *A Practical Guide to Content Delivery Networks*. Boca Raton, FL, USA: CRC Press, Inc., 2nd ed., 2010.

[8] "Dns attack factsheet 1.1." http://www.icann.org/en/news/announcements/announcement-08mar07-en.htm, March 2007.

[9] "Rssac 29 meeting minutes." http://www.icann.org/en/groups/rssac/meetings/rssac-29-en.pdf, December 2007.

[10] I. Cisco Systems, "Graphical Network Simulator 3." http://www.gns3.net/, 2007.

[11] S. Mccanne, S. Floyd, and K. Fall, "ns2 (network simulator 2)." http://www.isi.edu/nsnam/ns/index.html.

[12] R. Hinden and S. Deering, "IP Version 6 Addressing Architecture." RFC 4291 (Draft Standard), Feb. 2006. Updated by RFCs 5952, 6052.

[13] J. Abley and K. Lindqvist, "Operation of Anycast Services." RFC 4786 (Best Current Practice), Dec. 2006.

[14] J. Veiga, "Uma solução ipsec para comunicações seguras anycast em redes ipv6," Master's thesis, Escola de Engenharia, Universidade do Minho, 2011.

[15] M. Levine, B. Lyon, and T. Underwood, "Operational Experience with TCP and Anycast." `http://www.nanog.org/meetings/nanog37/presentations/matt.levine.pdf`, June 2006.

[16] J. Moy, "OSPF Version 2." RFC 1247 (Draft Standard), July 1991. Obsoleted by RFC 1583, updated by RFC 1349.

[17] D. Katabi and J. Wroclawski, "A framework for scalable global ip-anycast (gia)," *SIGCOMM Comput. Commun. Rev.*, vol. 30, pp. 3–15, Aug. 2000.

[18] H. Ballani and P. Francis, "Towards a global ip anycast service," *SIGCOMM Comput. Commun. Rev.*, vol. 35, pp. 301–312, Aug. 2005.

[19] T. Stevens, M. De Leenheer, C. Develder, F. De Turck, B. Dhoedt, and P. Demeester, "Astas: Architecture for scalable and transparent anycast services," *J. Commun. Netw.*, vol. 9, no. 4, pp. 1229–2370, 2007.

[20] R. Hinden and S. Deering, "Internet Protocol Version 6 (IPv6) Addressing Architecture." RFC 3513 (Proposed Standard), Apr. 2003. Obsoleted by RFC 4291.

[21] S. Doi, S. Ata, H. Kitamura, and M. Murata, "Ipv6 anycast for simple and effective service-oriented communications," *IEEE Communications Magazine*, vol. 42, pp. 163–171, 2004.

[22] D. Waitzman, C. Partridge, and S. Deering, "Distance Vector Multicast Routing Protocol." RFC 1075 (Experimental), Nov. 1988.

[23] J. Moy, "MOSPF: Analysis and Experience." RFC 1585 (Informational), Mar. 1994.

[24] B. Fenner, M. Handley, H. Holbrook, and I. Kouvelas, "Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised)." RFC 4601 (Proposed Standard), Aug. 2006. Updated by RFCs 5059, 5796, 6226.

[25] S. Matsunaga, S. Ata, H. Kitamura, and M. Murata, "Design and Implementation of IPv6 Anycast Routing Protocol: PIA-SM," in *Advanced Information Networking and Applications*, vol. 2, pp. 839–844, 2005.

[26] S. Deering, W. Fenner, and B. Haberman, "Multicast Listener Discovery (MLD) for IPv6." RFC 2710 (Proposed Standard), Oct. 1999. Updated by RFCs 3590, 3810.

[27] A. Sulaiman, B. Ali, S. Khatun, and G. Kurup, "An enhanced ipv6 anycast routing protocol using protocol independent multicast-sparse mode (pim-sm)," in *Telecommunications and Malaysia International Conference on Communications, 2007. ICT-MICC 2007. IEEE International Conference on*, pp. 588 –593, may 2007.

[28] k. claffy, "Border Gateway Protocol (BGP) and Traceroute Data Workshop Report," tech. rep., Cooperative Association for Internet Data Analysis (CAIDA), Oct 2011.

[29] R. Rivest, "The MD5 Message-Digest Algorithm." RFC 1321 (Informational), Apr. 1992. Updated by RFC 6151.

[30] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication." RFC 2104 (Informational), Feb. 1997. Updated by RFC 6151.

[31] D. Eastlake 3rd and P. Jones, "US Secure Hash Algorithm 1 (SHA1)." RFC 3174 (Informational), Sept. 2001. Updated by RFCs 4634, 6234.

[32] S. Kent, "IP Authentication Header." RFC 4302 (Proposed Standard), Dec. 2005.

[33] S. Kent, "IP Encapsulating Security Payload (ESP)." RFC 4303 (Proposed Standard), Dec. 2005.

[34] P. Savola, R. Lehtonen, and D. Meyer, "Protocol Independent Multicast - Sparse Mode (PIM-SM) Multicast Routing Security Issues and Enhancements." RFC 4609 (Informational), Oct. 2006.

[35] P. Savola and J. Lingard, "Host Threats to Protocol Independent Multicast (PIM)." RFC 5294 (Informational), Aug. 2008.

[36] F. Font and D. Mlynek, "Choosing the set of rendezvous points in shared trees minimizing traffic concentration," in *Communications, 2003. ICC '03. IEEE International Conference on*, vol. 3, pp. 1526 – 1530 vol.3, may 2003.

[37] D. Katabi, "The use of ip-anycast for building efficient multicast trees," in *Global Telecommunications Conference, 1999. GLOBECOM '99*, vol. 3, pp. 1679 –1688 vol.3, 1999.

[38] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1." RFC 2616 (Draft Standard), June 1999. Updated by RFCs 2817, 5785, 6266, 6585.

[39] T. Henderson, S. Roy, S. Floyd, and G. Riley, "ns3 (network simulator 3)." `http://www.nsnam.org/overview/what-is-ns-3/`.

[40] T. Henderson, "Improving Simulation Credibility Through Open Source Simulations," tech. rep., University of Washington, Mar 2008. Simutools Conference.

[41] A. S. e. V. F. António Costa, Maria João Nicolau, "Implementação e Teste do PIM-SM no Network Simulator," tech. rep., Conferência sobre Redes de Computadores (CRC2002), Faro, Portugal, Sep 26-27 2002.

[42] "Virtual InterNetwork Testbed (VINT)." A Collaboration among USC/ISI, Xerox PARC, LBNL, and UCB `http://www.isi.edu/nsnam/vintindex.html`, 2003.

[43] A. C. e. M. J. N. Hugo Ferreira, "Encaminhamento Anycast em Redes IPv6: uma proposta," tech. rep., Conferência sobre Redes de Computadores (CRC2012), Aveiro, Portugal, Nov 15-16 2012.

A

Tradução Adoptada

VoIP *Chamadas de Voz através da Internet;*

End-devices/Host *Sistemas Terminais;*

NAT *Tradutor de Endereços de Rede;*

DDoS *Ataques Distribuídos de Negação de Serviço;*

Root Servers *Servidores Raiz;*

TLD *Domínio de Alto Nível;*

IPSec *Segurança ao Nível da Rede;*

TCP *Comunicação com Conexões Contínuas;*

ECMP *Múltiplos Caminhos de Igual Custo;*

Cookies *Testemunho de Conexão;*

Default Route *Rota Padrão;*

Anycast Indicator[17] *Indicador Anycast;*

Home Domain's Unicast Prefix[17] *Prefixo Unicast do Domínio Principal;*

Group ID[17] *Identificador de Grupo;*

Border Router *Encaminhador fronteira;*

Originator Border Router *Encaminhador fronteira de origem;*

TTL *Validade;*

Overlay Network *Rede sobreposta;*

Overhead *Informação complementar de controlo;*

RAP[18] *UCP;*

JAP[18] *PL;*

IAP[18] *PA;*

Anycast Router[21][25] *Encaminhador Anycast;*

Seed Node[21][25] *Nó Incial;*

Bugs *Defeitos;*

B

Configurações dos encaminhadores(*GNS3*)

Nome	Descrição
Sistema Operativo	Ubuntu 12.04 LTS(Precise Pangolin)
Emulador	GNS3-0.7.4
Imagem do *IOS*	Imagem do encaminhador do LAP 3 da Escola de Engenharia.

Tabela B.1: Requisitos utilizados na implementação do enquadramento prático

Exemplo de código B.1: Configurações do R1.

```
1 !
2 version 12.4
3 service timestamps debug datetime msec
4 service timestamps log datetime msec
5 no service password-encryption
6 !
7 hostname R1
8 !
9 boot-start-marker
10 boot-end-marker
11 !
12 no aaa new-model
13 !
14 resource policy
15 !
16 memory-size iomem 5
17 !
18 ip cef
19 !
20 ipv6 unicast-routing
```

```
21 !
22 interface FastEthernet0/0
23  no ip address
24  duplex auto
25  speed auto
26  ipv6 address 2001:14::1/64
27  ipv6 enable
28  ipv6 nd prefix 2001:14::/64
29  ipv6 ospf 1 area 0
30 !
31 ip http server
32 no ip http secure-server
33 !
34 ipv6 router ospf 1
35  router-id 1.1.1.1
36  log-adjacency-changes
37 !
38 control-plane
39 !
40 line con 0
41 line aux 0
42 line vty 0 4
43  login
44 !
45 end
```

Exemplo de código B.2: Configurações do R2.

```
1 !
2 version 12.4
3 service timestamps debug datetime msec
4 service timestamps log datetime msec
5 no service password-encryption
6 !
7 hostname R2
8 !
9 boot-start-marker
10 boot-end-marker
```

```
11 !
12 no aaa new-model
13 !
14 resource policy
15 !
16 memory-size iomem 5
17 !
18 ip cef
19 !
20 ipv6 unicast-routing
21 !
22 interface FastEthernet0/0
23  no ip address
24  duplex auto
25  speed auto
26  ipv6 address 2001:24::2/64
27  ipv6 enable
28  ipv6 nd prefix 2001:24::/64
29  ipv6 ospf 1 area 0
30 !
31 interface FastEthernet1/0
32  no ip address
33  duplex auto
34  speed auto
35  ipv6 address 2001:22::2/64
36  ipv6 address 2001:22::/64 anycast
37  ipv6 enable
38  ipv6 nd prefix 2001:22::/64
39  ipv6 ospf 1 area 2
40 !
41 ip http server
42 no ip http secure-server
43 !
44 ipv6 router ospf 1
45  router-id 2.2.2.2
46  log-adjacency-changes
47 !
48 control-plane
49 !
50 line con 0
```

```
51 line aux 0
52 line vty 0 4
53  login
54 !
55 end
```

Exemplo de código B.3: Configurações do R3.

```
 1 !
 2 version 12.4
 3 service timestamps debug datetime msec
 4 service timestamps log datetime msec
 5 no service password-encryption
 6 !
 7 hostname R3
 8 !
 9 boot-start-marker
10 boot-end-marker
11 !
12 no aaa new-model
13 !
14 resource policy
15 !
16 memory-size iomem 5
17 !
18 ip cef
19 !
20 ipv6 unicast-routing
21 !
22 interface FastEthernet0/0
23  no ip address
24  duplex auto
25  speed auto
26  ipv6 address 2001:34::3/64
27  ipv6 enable
28  ipv6 nd prefix 2001:34::/64
29  ipv6 ospf 1 area 0
30 !
```

```
31 interface FastEthernet1/0
32  no ip address
33  duplex auto
34  speed auto
35  ipv6 address 2001:33::3/64
36  ipv6 address 2001:22::/64 anycast
37  ipv6 enable
38  ipv6 nd prefix 2001:33::/64
39  ipv6 ospf 1 area 3
40 !
41 ip http server
42 no ip http secure-server
43 !
44 ipv6 router ospf 1
45  router-id 3.3.3.3
46  log-adjacency-changes
47 !
48 control-plane
49 !
50 line con 0
51 line aux 0
52 line vty 0 4
53  login
54 !
55 !
56 end
```

Exemplo de código B.4: Configurações do R4.

```
1 !
2 version 12.4
3 service timestamps debug datetime msec
4 service timestamps log datetime msec
5 no service password-encryption
6 !
7 hostname CLOUD-R4
8 !
9 boot-start-marker
```

```
10 boot-end-marker
11 !
12 no aaa new-model
13 !
14 resource policy
15 !
16 memory-size iomem 5
17 !
18 ip cef
19 !
20 ipv6 unicast-routing
21 !
22 interface FastEthernet0/0
23  no ip address
24  duplex auto
25  speed auto
26  ipv6 address 2001:14::4/64
27  ipv6 enable
28  ipv6 nd prefix 2001:14::/64
29  ipv6 ospf 1 area 0
30 !
31 interface FastEthernet1/0
32  no ip address
33  duplex auto
34  speed auto
35  ipv6 address 2001:24::4/64
36  ipv6 enable
37  ipv6 nd prefix 2001:24::/64
38  ipv6 ospf 1 area 0
39 !
40 interface FastEthernet2/0
41  no ip address
42  duplex auto
43  speed auto
44  ipv6 address 2001:34::4/64
45  ipv6 enable
46  ipv6 nd prefix 2001:34::/64
47  ipv6 ospf 1 area 0
48 !
49 router ospf 1
```

```
50  log-adjacency-changes
51 !
52 ip http server
53 no ip http secure-server
54 !
55 ipv6 router ospf 1
56  router-id 4.4.4.4
57  log-adjacency-changes
58  maximum-paths 1
59 !
60 control-plane
61 !
62 line con 0
63 line aux 0
64 line vty 0 4
65  login
66 !
67 end
```

Exemplo de código B.5: Configurações do R5.

```
 1 !
 2 version 12.4
 3 service timestamps debug datetime msec
 4 service timestamps log datetime msec
 5 no service password-encryption
 6 !
 7 hostname Servidor1
 8 !
 9 boot-start-marker
10 boot-end-marker
11 !
12 no aaa new-model
13 !
14 resource policy
15 !
16 memory-size iomem 5
17 !
```

```
18 ip cef
19 !
20 ipv6 unicast-routing
21 !
22 interface FastEthernet0/0
23  no ip address
24  duplex auto
25  speed auto
26  ipv6 address 2001:22::0/128 anycast
27  ipv6 address autoconfig
28  ipv6 ospf 1 area 2
29 !
30 ip http server
31 no ip http secure-server
32 !
33 ipv6 router ospf 1
34  router-id 5.5.5.5
35  log-adjacency-changes
36 !
37 control-plane
38 !
39 line con 0
40 line aux 0
41 line vty 0 4
42  login
43 !
44 end
```

Exemplo de código B.6: Configurações do R6.

```
1 !
2 version 12.4
3 service timestamps debug datetime msec
4 service timestamps log datetime msec
5 no service password-encryption
6 !
7 hostname Servidor2
8 !
```

```
 9 boot-start-marker
10 boot-end-marker
11 !
12 no aaa new-model
13 !
14 resource policy
15 !
16 memory-size iomem 5
17 !
18 ip cef
19 !
20 ipv6 unicast-routing
21 !
22 interface FastEthernet0/0
23  no ip address
24  duplex auto
25  speed auto
26  ipv6 address  2001:22::0/128 anycast
27  ipv6 address autoconfig
28  ipv6 ospf 1 area 3
29 !
30 ip http server
31 no ip http secure-server
32 !
33 ipv6 router ospf 1
34  router-id 6.6.6.6
35  log-adjacency-changes
36 !
37 control-plane
38 !
39 line con 0
40 line aux 0
41 line vty 0 4
42  login
43 !
44 end
```

C

Scripts de testes utilizadas

Exemplo de código C.1: Cenário elaborado para explicar o funcionamento do protocolo *anycast*

```
1 # This script illustrates how the new anycast Protocol works.
2 #
3 # Written by Hugo Ferreira
4 # hugoluisferreira@me.com
5 #
6
7 # create a simulator instance with multicast features on
8 set ns [new Simulator -multicast on]
9
10 # define trace files...
11 set pimTraceFile [open pimTrace.tr w]
12 $ns trace-all $pimTraceFile
13 set pimNamFile [open pimTrace.nam w]
14 $ns namtrace-all $pimNamFile
15
16 # cbr packets colors
17 $ns color 1 red
18 $ns color 2 green
19 # prune/graft packets
20 $ns color 30 purple
21 $ns color 31 blue
22
23 # create nodes...
24 set iPIMNodes 11
25 for {set i 0} {$i < $iPIMNodes} {incr i} {
26   set pimNode($i) [$ns node]
27   lappend pimNodeList $pimNode($i)
28 }
29
```

```
30 # change shape and color of RP and sources
31 $pimNode(0) shape square;  # Rendezvous Point 0
32 $pimNode(0) color purple;
33 $pimNode(6) shape hexagon; # Servidor 1
34 $pimNode(6) color red;
35 $pimNode(7) shape hexagon; # Servidor 2
36 $pimNode(7) color red;
37 $pimNode(8) shape hexagon; # Servidor 3
38 $pimNode(8) color red;
39 $pimNode(9) shape box; #Client 1
40 $pimNode(9) color green;
41 $pimNode(10) shape box; #Client 2
42 $pimNode(10) color green;
43
44 #Labelling
45 $pimNode(0) label "RP";
46 $pimNode(6) label "SA 1";
47 $pimNode(7) label "SA 2";
48 $pimNode(8) label "SA 3";
49 $pimNode(9) label "C 1";
50 $pimNode(10) label "C 2";
51
52
53 # topology
54 $ns duplex-link $pimNode(0) $pimNode(1) 1.5Mb 10ms DropTail
55 $ns duplex-link $pimNode(0) $pimNode(4) 1.5Mb 10ms DropTail
56 $ns duplex-link $pimNode(1) $pimNode(2) 1.5Mb 10ms DropTail
57 $ns duplex-link $pimNode(1) $pimNode(3) 1.5Mb 10ms DropTail
58 $ns duplex-link $pimNode(2) $pimNode(3) 1.5Mb 10ms DropTail
59 $ns duplex-link $pimNode(2) $pimNode(6) 1.5Mb 10ms DropTail
60 $ns duplex-link $pimNode(2) $pimNode(9) 1.5Mb 10ms DropTail
61 $ns duplex-link $pimNode(3) $pimNode(7) 1.5Mb 10ms DropTail
62 $ns duplex-link $pimNode(3) $pimNode(8) 1.5Mb 10ms DropTail
63 $ns duplex-link $pimNode(4) $pimNode(5) 1.5Mb 10ms DropTail
64 $ns duplex-link $pimNode(5) $pimNode(10) 1.5Mb 10ms DropTail
65
66 # topology layout
67 $ns duplex-link-op $pimNode(0) $pimNode(1) orient left-down
68 $ns duplex-link-op $pimNode(0) $pimNode(4) orient right
69 $ns duplex-link-op $pimNode(1) $pimNode(2) orient 240deg
```

```
70 $ns duplex-link-op $pimNode(1) $pimNode(3) orient 300deg
71 $ns duplex-link-op $pimNode(2) $pimNode(3) orient right
72 $ns duplex-link-op $pimNode(2) $pimNode(6) orient down
73 $ns duplex-link-op $pimNode(2) $pimNode(9) orient up-left
74 $ns duplex-link-op $pimNode(3) $pimNode(7) orient 250deg
75 $ns duplex-link-op $pimNode(3) $pimNode(8) orient 290deg
76 $ns duplex-link-op $pimNode(4) $pimNode(5) orient right-down
77 $ns duplex-link-op $pimNode(5) $pimNode(10) orient right-down
78
79 # Alloc multicast address
80 set pimGroup0 [Node allocaddr]
81
82 # Setup clientes (1 e 2)
83 # Cliente 6
84 set udp9 [new Agent/UDP]
85 $udp9 set dst_addr_ $pimGroup0
86 $udp9 set dst_port_ 0
87 $udp9 set class_ 1
88 $udp9 set flag_ 0
89 $ns attach-agent $pimNode(9) $udp9
90 set cbr9 [new Application/Traffic/CBR]
91 $cbr9 attach-agent $udp9
92 $cbr9 set rate_ 1
93 $cbr9 set node_ $pimNode(9)
94 # Cliente 7
95 set udp10 [new Agent/UDP]
96 $udp10 set dst_addr_ $pimGroup0
97 $udp10 set dst_port_ 0
98 $udp10 set class_ 2
99 $udp10 set flag_ 0
100 $ns attach-agent $pimNode(10) $udp10
101 set cbr10 [new Application/Traffic/CBR]
102 $cbr10 attach-agent $udp10
103 $cbr10 set rate_ 1
104 $cbr10 set node_ $pimNode(10)
105
106 # Setup Servidores (1, 2 e 3)
107 # Servidor 1
108 set udp1 [new Agent/UDP]
109 $ns attach-agent $pimNode(6) $udp1
```

```
110 set rcvr1 [new Agent/LossMonitor]
111 $rcvr1 set echo_reply_ 1
112 $ns attach-agent $pimNode(6) $rcvr1
113 # Servidor 2
114 set udp2 [new Agent/UDP]
115 $ns attach-agent $pimNode(7) $udp2
116 set rcvr2 [new Agent/LossMonitor]
117 $rcvr2 set echo_reply_ 1
118 $ns attach-agent $pimNode(7) $rcvr2
119 # Servidor 3
120 set udp3 [new Agent/UDP]
121 $ns attach-agent $pimNode(8) $udp3
122 set rcvr3 [new Agent/LossMonitor]
123 $rcvr3 set echo_reply_ 1
124 $ns attach-agent $pimNode(8) $rcvr3
125
126 # define unicast and multicast protocols to use...
127 $ns rtproto DV
128 PIM set RP_($pimGroup0) $pimNode(0)
129 $ns mrtproto PIM
130
131 # Inicio do escalonamento dos eventos
132 # 1' evento:: Os rcvrs ligam-se ao RP
133   $ns at 0.00 "$ns set-animation-rate 10ms"
134   $ns at 0.00 "$ns trace-annotate \"Starting simulations. \""
135
136   $ns at 0.10 "$ns set-animation-rate 0.2ms"
137   $ns at 0.10 "$ns trace-annotate \"Server on Node 6 joins the group...\""
138   $ns at 0.10 "$ns trace-annotate \"Node 6 sends Join towards group RP with
        metric 1 .\""
139   $ns at 0.10 "$pimNode(6) join-group $rcvr1 $pimGroup0 1"
140
141   $ns at 0.10 "$ns trace-annotate \"Receiver on Node 7 joins the group...\""
142   $ns at 0.10 "$ns trace-annotate \"Node 7 sends Join towards group RP with
        metric 1 .\""
143   $ns at 0.10 "$pimNode(7) join-group $rcvr2 $pimGroup0 1"
144
145   $ns at 0.10 "$ns set-animation-rate 0.2ms"
146   $ns at 0.10 "$ns trace-annotate \"Receiver on Node 8 joins the group...\""
```

```
147  $ns at 0.10 "$ns trace-annotate \"Node 8 sends Join towards group RP with
       metric 1.\""
148  $ns at 0.10 "$pimNode(8) join-group $rcvr3 $pimGroup0 1"
149
150 #  2' evento :: cbr9 çcomea a fazer pedidos.
151 #  Os rcvrs fazem automaticamente o join ao cliente
152  $ns at 0.50 "$ns set-animation-rate 0.2ms"
153  $ns at 0.50 "$ns trace-annotate \"Source at Node 9 starts sending... \""
154  $ns at 0.50 "$ns trace-annotate \"Node 9 forwards CBR packets to Group.\""
155  $ns at 0.50 "$cbr9 start"
156
157  $ns at 0.565 "$ns set-animation-rate 0.2ms"
158  $ns at 0.565 "$ns trace-annotate \"Node 6 decides to join source 9 tree...\
       ""
159  $ns at 0.565 "$ns trace-annotate \"Node 6 sends Join towards Node 9.\""
160  $ns at 0.565 "$pimNode(6) join-group $rcvr1 $pimGroup0 1 $pimNode(9)"
161
162  $ns at 0.565 "$ns set-animation-rate 0.2ms"
163  $ns at 0.565 "$ns trace-annotate \"Node 7 decides to join source 9 tree...\
       ""
164  $ns at 0.565 "$ns trace-annotate \"Node 7 sends Join towards Node 9.\""
165  $ns at 0.565 "$pimNode(7) join-group $rcvr2 $pimGroup0 1 $pimNode(9)"
166
167  $ns at 0.565 "$ns set-animation-rate 0.2ms"
168  $ns at 0.565 "$ns trace-annotate \"Node 8 decides to join source 9 tree...\
       ""
169  $ns at 0.565 "$ns trace-annotate \"Node 8 sends Join towards Node 9.\""
170  $ns at 0.565 "$pimNode(8) join-group $rcvr3 $pimGroup0 1 $pimNode(9)"
171
172 #  3' evento :: cbr9 çcomea a fazer pedidos.
173 #  Os rcvrs fazem automaticamente o join ao cliente
174  $ns at 1.0 "$ns set-animation-rate 0.2ms"
175  $ns at 1.0 "$ns trace-annotate \"Source at Node 10 starts sending... \""
176  $ns at 1.0 "$ns trace-annotate \"Node 10 forwards CBR packets to Group.\""
177  $ns at 1.0 "$cbr10 start"
178
179  $ns at 1.065 "$ns set-animation-rate 0.2ms"
180  $ns at 1.065 "$ns trace-annotate \"Node 6 decides to join source 10 tree...
       \""
181  $ns at 1.065 "$ns trace-annotate \"Node 6 sends Join towards Node 10.\""
```

```
182   $ns at 1.065 "$pimNode(6) join-group $rcvr1 $pimGroup0 2 $pimNode(10)"
183
184   $ns at 1.065 "$ns set-animation-rate 0.2ms"
185   $ns at 1.065 "$ns trace-annotate \"Node 7 decides to join source 10 tree...
        \""
186   $ns at 1.065 "$ns trace-annotate \"Node 7 sends Join towards Node 10.\""
187   $ns at 1.065 "$pimNode(7) join-group $rcvr2 $pimGroup0 1 $pimNode(10)"
188
189   $ns at 1.065 "$ns set-animation-rate 0.2ms"
190   $ns at 1.065 "$ns trace-annotate \"Node 8 decides to join source 10 tree...
        \""
191   $ns at 1.065 "$ns trace-annotate \"Node 8 sends Join towards Node 10.\""
192   $ns at 1.065 "$pimNode(8) join-group $rcvr3 $pimGroup0 1 $pimNode(10)"
193
194 #  4' evento :: rcvr1 abandona o grupo.
195   $ns at 1.50 "$ns set-animation-rate 0.2ms"
196   $ns at 1.50 "$ns trace-annotate \"Node 6 leaves group...\""
197   $ns at 1.50 "$pimNode(6) leave-group $rcvr1 $pimGroup0"
198
199 #  5' evento :: çãActualizao das émtricas de todos os servidores.
200   $ns at 2.00 "$pimNode(7) update-group $rcvr2 $pimGroup0 2 $pimNode(9)"
201
202 #  6' evento :: Falha de link entre dois óns [2 e 3].
203   $ns rtmodel-at 2.50 down $pimNode(2) $pimNode(3)
204
205 #  7' evento :: Link do 7'evento recupera. Outra falha de link entre dois óns
        [2 e 3].
206   $ns rtmodel-at 3.0 up $pimNode(2) $pimNode(3)
207
208 #  8' evento :: cbr9 e cbr10 param de fazer pedidos.
209   $ns at 3.50 "$ns set-animation-rate 0.2ms"
210   $ns at 3.50 "$ns trace-annotate \"Source at Node 9 stops sending... \""
211   $ns at 3.50 "$ns trace-annotate \"Node 9 forwards CBR packets to Group.\""
212   $ns at 3.50 "$cbr9 stop"
213
214   $ns at 3.50 "$ns set-animation-rate 0.2ms"
215   $ns at 3.50 "$ns trace-annotate \"Source at Node 10 stops sending... \""
216   $ns at 3.50 "$ns trace-annotate \"Node 10 forwards CBR packets to Group.\""
217   $ns at 3.50 "$cbr10 stop"
218
```

```
219 # 9' evento :: Servidor 2 e 3 abandonam o grupo.
220   $ns at 3.80 "$ns set-animation-rate 0.2ms"
221   $ns at 3.80 "$ns trace-annotate \"Node 7 leaves group...\""
222   $ns at 3.80 "$pimNode(7) leave-group $rcvr2 $pimGroup0"
223
224   $ns at 3.80 "$ns set-animation-rate 0.2ms"
225   $ns at 3.80 "$ns trace-annotate \"Node 8 leaves group...\""
226   $ns at 3.80 "$pimNode(8) leave-group $rcvr3 $pimGroup0"
227
228 # 10' evento :: Fim da çãsimuao.
229 $ns at 4.00 "$ns trace-annotate \"Simulation finished... \""
230 $ns at 4.00 "finish"
231
232 proc finish {} {
233   global ns pimTraceFile pimNamFile
234   $ns flush-trace
235
236   close $pimTraceFile
237   close $pimNamFile
238
239   puts "running nam pimTrace.nam..."
240   exec nam pimTrace.nam &
241
242   exit 0
243 }
244
245 $ns run
```

Exemplo de código C.2: Cenário aleatório para apurar a exequibilidade do protocolo *anycast*

```
1 #
  ------------------------------------------------------------------------------

2 global ns rng nnodes nlinks rp s1 s2 s3 s4 c1 c2 c3
3 global x1 x2
4 set ns [new Simulator -multicast on]
5
6 # ºN de óns
```

```
7 set nnodes 18
8
9 # ºN de links
10 set nlinks 30
11
12 # Gera ºn óaleatrio. O seed server para evitar que o únmero gerado seja igual
      em todas as çõsimulaes.
13 set rng [new RNG]
14 $rng seed 0
15
16 # Define o únmero ámximo que pode ser gerado para cada ón.
17 set num [expr $nnodes - 1]
18
19 set numLinks [expr $nlinks - 1]
20
21 set numMetrica [expr 100 - 1]
22
23 set s1 -1
24 set s2 -1
25 set s3 -1
26 set s4 -1
27 set c1 -1
28 set c2 -1
29 set c3 -1
30
31 proc compare-random { new n1 n2 n3 n4 n5 n6 n7 } {
32    if { $new == $n1 || $new == $n2 || $new == $n3 || $new == $n4 || $new ==
         $n5 || $new == $n6 || $new == $n7 } {
33       set new -1
34    }
35    return $new
36 }
37
38
39 # RP
40 # define o valor do ón que áser o RP
41 set rp [expr [$rng integer $num]]
42
43 # Servidor
44 # define o valor do ón que áser s1
```

```
45 while { $s1 == -1 } {
46   #set rng [new RNG]
47   #$rng seed 0
48    set s1 [expr [$rng integer $num]]
49    set s1 [compare-random $s1 $rp $s2 $s3 $s4 $c1 $c2 $c3]
50 }
51 # define o valor do ón que áser s2
52 while { $s2 == -1 } {
53   #set rng [new RNG]
54   #$rng seed 0
55    set s2 [expr [$rng integer $num]]
56    set s2 [compare-random $s2 $rp $s1 $s3 $s4 $c1 $c2 $c3]
57 }
58 # define o valor do ón que áser s3
59 while { $s3 == -1 } {
60   #set rng [new RNG]
61   #$rng seed 0
62    set s3 [expr [$rng integer $num]]
63    set s3 [compare-random $s3 $rp $s1 $s2 $s4 $c1 $c2 $c3]
64 }
65 # define o valor do ón que áser s4
66 while { $s4 == -1 } {
67   #set rng [new RNG]
68   #$rng seed 0
69    set s4 [expr [$rng integer $num]]
70    set s4 [compare-random $s4 $rp $s1 $s2 $s3 $c1 $c2 $c3]
71 }
72
73 #Cliente
74 # define o valor do ón que áser c1
75 while { $c1 == -1 } {
76   #set rng [new RNG]
77   #$rng seed 0
78    set c1 [expr [$rng integer $num]]
79    set c1 [compare-random $c1 $rp $s1 $s2 $s3 $s4 $c2 $c3]
80 }
81 # define o valor do ón que áser c2
82 while { $c2 == -1 } {
83   #set rng [new RNG]
84   #$rng seed 0
```

```
85    set c2 [expr [$rng integer $num]]
86    set c2 [compare-random $c2 $rp $s1 $s2 $s3 $s4 $c1 $c3]
87 }
88 # define o valor do ón que áser c3
89 while { $c3 == -1 } {
90    #set rng [new RNG]
91    #$rng seed 0
92    set c3 [expr [$rng integer $num]]
93    set c3 [compare-random $c3 $rp $s1 $s2 $s3 $s4 $c1 $c2]
94 }
95
96 # define trace files...
97 set pimTraceFile [open pimTrace.tr w]
98 $ns trace-all $pimTraceFile
99 set pimNamFile [open pimTrace.nam w]
100 $ns namtrace-all $pimNamFile
101
102 # cbr packets colors
103 $ns color 1 red
104 $ns color 2 green
105 $ns color 3 yellow
106 # prune/graft packets
107 $ns color 30 purple
108 $ns color 31 blue
109
110 # create nodes...
111 set iPIMNodes $nnodes
112 for {set i 0} {$i < $iPIMNodes} {incr i} {
113    set pimNode($i) [$ns node]
114    lappend pimNodeList $pimNode($i)
115 }
116
117 # topology
118 $ns duplex-link $pimNode(0) $pimNode(1) 1.5Mb 10ms DropTail
119 $ns duplex-link $pimNode(0) $pimNode(3) 1.5Mb 10ms DropTail
120 $ns duplex-link $pimNode(0) $pimNode(6) 1.5Mb 10ms DropTail
121 $ns duplex-link $pimNode(1) $pimNode(4) 1.5Mb 10ms DropTail
122 $ns duplex-link $pimNode(2) $pimNode(3) 1.5Mb 10ms DropTail
123 $ns duplex-link $pimNode(2) $pimNode(5) 1.5Mb 10ms DropTail
124 $ns duplex-link $pimNode(3) $pimNode(4) 1.5Mb 10ms DropTail
```

```
125 $ns duplex-link $pimNode(3) $pimNode(5) 1.5Mb 10ms DropTail
126 $ns duplex-link $pimNode(4) $pimNode(8) 1.5Mb 10ms DropTail
127 $ns duplex-link $pimNode(4) $pimNode(11) 1.5Mb 10ms DropTail
128 $ns duplex-link $pimNode(5) $pimNode(6) 1.5Mb 10ms DropTail
129 $ns duplex-link $pimNode(5) $pimNode(7) 1.5Mb 10ms DropTail
130 $ns duplex-link $pimNode(5) $pimNode(15) 1.5Mb 10ms DropTail
131 $ns duplex-link $pimNode(6) $pimNode(7) 1.5Mb 10ms DropTail
132 $ns duplex-link $pimNode(6) $pimNode(8) 1.5Mb 10ms DropTail
133 $ns duplex-link $pimNode(6) $pimNode(10) 1.5Mb 10ms DropTail
134 $ns duplex-link $pimNode(6) $pimNode(15) 1.5Mb 10ms DropTail
135 $ns duplex-link $pimNode(7) $pimNode(8) 1.5Mb 10ms DropTail
136 $ns duplex-link $pimNode(8) $pimNode(9) 1.5Mb 10ms DropTail
137 $ns duplex-link $pimNode(8) $pimNode(11) 1.5Mb 10ms DropTail
138 $ns duplex-link $pimNode(9) $pimNode(12) 1.5Mb 10ms DropTail
139 $ns duplex-link $pimNode(10) $pimNode(13) 1.5Mb 10ms DropTail
140 $ns duplex-link $pimNode(11) $pimNode(12) 1.5Mb 10ms DropTail
141 $ns duplex-link $pimNode(11) $pimNode(16) 1.5Mb 10ms DropTail
142 $ns duplex-link $pimNode(12) $pimNode(14) 1.5Mb 10ms DropTail
143 $ns duplex-link $pimNode(13) $pimNode(15) 1.5Mb 10ms DropTail
144 $ns duplex-link $pimNode(14) $pimNode(16) 1.5Mb 10ms DropTail
145 $ns duplex-link $pimNode(15) $pimNode(16) 1.5Mb 10ms DropTail
146 $ns duplex-link $pimNode(15) $pimNode(17) 1.5Mb 10ms DropTail
147 $ns duplex-link $pimNode(16) $pimNode(17) 1.5Mb 10ms DropTail
148
149 set rndLinks [expr [$rng integer $numLinks]]
150
151 switch $rndLinks {
152
153     0   {
154       set x1 0
155       set x2 1
156     }
157   1   {
158       set x1 0
159       set x2 3
160     }
161   2   {
162       set x1 0
163       set x2 6
164     }
```

```
165     3     {
166           set x1 1
167           set x2 4
168       }
169     4     {
170           set x1 2
171           set x2 3
172       }
173     5     {
174           set x1 2
175           set x2 5
176       }
177     6     {
178           set x1 3
179           set x2 4
180       }
181     7     {
182           set x1 3
183           set x2 5
184       }
185     8     {
186           set x1 4
187           set x2 8
188       }
189     9     {
190           set x1 4
191           set x2 11
192       }
193     10    {
194           set x1 5
195           set x2 6
196       }
197     11    {
198           set x1 5
199           set x2 7
200       }
201     12    {
202           set x1 5
203           set x2 15
204       }
```

```
205    13   {
206         set x1 6
207         set x2 7
208       }
209    14   {
210         set x1 6
211         set x2 8
212       }
213    15   {
214         set x1 6
215         set x2 10
216       }
217    16   {
218         set x1 6
219         set x2 15
220       }
221    17   {
222         set x1 7
223         set x2 8
224       }
225    18   {
226         set x1 8
227         set x2 9
228       }
229    19   {
230         set x1 8
231         set x2 11
232       }
233    20   {
234         set x1 9
235         set x2 12
236       }
237    21   {
238         set x1 10
239         set x2 13
240       }
241    22   {
242         set x1 11
243         set x2 12
244       }
```

```
245    23   {
246         set x1 11
247         set x2 16
248       }
249    24   {
250         set x1 12
251         set x2 14
252       }
253    25   {
254         set x1 13
255         set x2 15
256       }
257    26   {
258         set x1 14
259         set x2 16
260       }
261    27   {
262         set x1 15
263         set x2 16
264       }
265    28   {
266         set x1 15
267         set x2 17
268       }
269    29   {
270         set x1 16
271         set x2 17
272       }
273
274       default {
275           set x1 0
276           set x2 1
277         }
278
279 }
280
281 $pimNode($rp) shape box;
282 $pimNode($rp) color yellow;
283 $pimNode($s1) shape hexagon;
284 $pimNode($s1) color red;
```

```
285 $pimNode($s2) shape hexagon;
286 $pimNode($s2) color red;
287 $pimNode($s3) shape hexagon;
288 $pimNode($s3) color red;
289 $pimNode($s4) shape box;
290 $pimNode($s4) color red;
291 $pimNode($c1) shape box;
292 $pimNode($c1) color green;
293 $pimNode($c2) shape box;
294 $pimNode($c2) color green;
295 $pimNode($c3) shape box;
296 $pimNode($c3) color green;
297
298 # Alloc multicast address
299 set pimGroup0 [Node allocaddr]
300
301 # Setup clients (6,7,8)
302 # Cliente 6
303 set udp6 [new Agent/UDP]
304 $udp6 set dst_addr_ $pimGroup0
305 $udp6 set dst_port_ 0
306 $udp6 set class_ 1
307 $udp6 set flag_ 0
308 $ns attach-agent $pimNode($c1) $udp6
309 set cbr6 [new Application/Traffic/CBR]
310 $cbr6 attach-agent $udp6
311 $cbr6 set rate_ 1
312 $cbr6 set node_ $pimNode($c1)
313
314 # Cliente 7
315 set udp7 [new Agent/UDP]
316 $udp7 set dst_addr_ $pimGroup0
317 $udp7 set dst_port_ 0
318 $udp7 set class_ 2
319 $udp7 set flag_ 0
320 $ns attach-agent $pimNode($c2) $udp7
321 set cbr7 [new Application/Traffic/CBR]
322 $cbr7 attach-agent $udp7
323 $cbr7 set rate_ 1
324 $cbr7 set node_ $pimNode($c2)
```

```
325 # Cliente 8
326 set udp8 [new Agent/UDP]
327 $udp8 set dst_addr_ $pimGroup0
328 $udp8 set dst_port_ 0
329 $udp8 set class_ 3
330 $udp8 set flag_ 0
331 $ns attach-agent $pimNode($c3) $udp8
332 set cbr8 [new Application/Traffic/CBR]
333 $cbr8 attach-agent $udp8
334 $cbr8 set rate_ 1
335 $cbr8 set node_ $pimNode($c3)
336
337
338 # Setup receivers (1, 2, 3, 4)
339 # Servidor 1
340 set udp1 [new Agent/UDP]
341 $ns attach-agent $pimNode($s1) $udp1
342 set rcvr1 [new Agent/LossMonitor]
343 $rcvr1 set echo_reply_ 1
344 $ns attach-agent $pimNode($s1) $rcvr1
345 set metricaS1 [expr [$rng integer $numMetrica]]
346 set metricaS1_v1 [expr [$rng integer $numMetrica]]
347 set metricaS1_v2 [expr [$rng integer $numMetrica]]
348 set metricaS1_v3 [expr [$rng integer $numMetrica]]
349 set metricaS1_v4 [expr [$rng integer $numMetrica]]
350 # Servidor 2
351 set udp2 [new Agent/UDP]
352 $ns attach-agent $pimNode($s2) $udp2
353 set rcvr2 [new Agent/LossMonitor]
354 $rcvr2 set echo_reply_ 1
355 $ns attach-agent $pimNode($s2) $rcvr2
356 set metricaS2 [expr [$rng integer $numMetrica]]
357 set metricaS2_v1 [expr [$rng integer $numMetrica]]
358 set metricaS2_v2 [expr [$rng integer $numMetrica]]
359 set metricaS2_v3 [expr [$rng integer $numMetrica]]
360 set metricaS2_v4 [expr [$rng integer $numMetrica]]
361 # Servidor 3
362 set udp3 [new Agent/UDP]
363 $ns attach-agent $pimNode($s3) $udp3
364 set rcvr3 [new Agent/LossMonitor]
```

```
365 $rcvr3 set echo_reply_ 1
366 $ns attach-agent $pimNode($s3) $rcvr3
367 set metricaS3 [expr [$rng integer $numMetrica]]
368 set metricaS3_v1 [expr [$rng integer $numMetrica]]
369 set metricaS3_v2 [expr [$rng integer $numMetrica]]
370 set metricaS3_v3 [expr [$rng integer $numMetrica]]
371 set metricaS3_v4 [expr [$rng integer $numMetrica]]
372 # Servidor 4
373 set udp4 [new Agent/UDP]
374 $ns attach-agent $pimNode($s4) $udp4
375 set rcvr4 [new Agent/LossMonitor]
376 $rcvr4 set echo_reply_ 1
377 $ns attach-agent $pimNode($s4) $rcvr4
378 set metricaS4 [expr [$rng integer $numMetrica]]
379 set metricaS4_v1 [expr [$rng integer $numMetrica]]
380 set metricaS4_v2 [expr [$rng integer $numMetrica]]
381 set metricaS4_v3 [expr [$rng integer $numMetrica]]
382 set metricaS4_v4 [expr [$rng integer $numMetrica]]
383
384 # define unicast and multicast protocols to use...
385 $ns rtproto DV
386 PIM set RP_($pimGroup0) $pimNode($rp)
387 $ns mrtproto PIM
388
389 # Inicio do escalonamento dos eventos
390 # 1' evento:: Os rcvrs ligam-se ao RP
391   $ns at 0.00 "$ns set-animation-rate 10ms"
392   $ns at 0.00 "$ns trace-annotate \"Starting simulations. \""
393
394   $ns at 0.10 "$ns set-animation-rate 0.2ms"
395   $ns at 0.10 "$ns trace-annotate \"Server on Node $s1 joins the group...\""
396   $ns at 0.10 "$ns trace-annotate \"Node $s1 sends Join towards group RP with
          metric $metricaS1 .\""
397   $ns at 0.10 "$pimNode($s1) join-group $rcvr1 $pimGroup0 $metricaS1"
398
399   $ns at 0.10 "$ns trace-annotate \"Receiver on Node $s2 joins the group...\"
        "
400   $ns at 0.10 "$ns trace-annotate \"Node $s2 sends Join towards group RP with
          metric $metricaS2 .\""
401   $ns at 0.10 "$pimNode($s2) join-group $rcvr2 $pimGroup0 $metricaS2"
```

```
402
403   $ns at 0.10 "$ns set-animation-rate 0.2ms"
404   $ns at 0.10 "$ns trace-annotate \"Receiver on Node $s3 joins the group...\"
      "
405   $ns at 0.10 "$ns trace-annotate \"Node $s3 sends Join towards group RP with
       metric $metricaS3.\""
406   $ns at 0.10 "$pimNode($s3) join-group $rcvr3 $pimGroup0 $metricaS3"
407
408   $ns at 0.10 "$ns set-animation-rate 0.2ms"
409   $ns at 0.10 "$ns trace-annotate \"Receiver on Node $s4 joins the group...\"
      "
410   $ns at 0.10 "$ns trace-annotate \"Node $s4 sends Join towards group RP with
       metric $metricaS4.\""
411   $ns at 0.10 "$pimNode($s4) join-group $rcvr4 $pimGroup0 $metricaS4"
412
413 #  2' evento :: cbr6 çcomea a fazer pedidos.
414 #  Os rcvrs fazem automaticamente o join ao cliente
415    $ns at 2.00 "$ns set-animation-rate 0.2ms"
416   $ns at 2.00 "$ns trace-annotate \"Source at Node $c1 starts sending... \""
417   $ns at 2.00 "$ns trace-annotate \"Node $c1 forwards CBR packets to Group.\"
      "
418   $ns at 2.00 "$cbr6 start"
419
420   $ns at 2.00 "$ns set-animation-rate 0.2ms"
421   $ns at 2.00 "$ns trace-annotate \"Node $s1 decides to join source ($c1)
      tree...\""
422   $ns at 2.00 "$ns trace-annotate \"Node $s1 sends Join towards Node $c1.\""
423   $ns at 2.00 "$pimNode($s1) join-group $rcvr1 $pimGroup0 $metricaS1_v1
      $pimNode($c1)"
424
425   $ns at 2.00 "$ns set-animation-rate 0.2ms"
426   $ns at 2.00 "$ns trace-annotate \"Node $s2 decides to join source ($c1)
      tree...\""
427   $ns at 2.00 "$ns trace-annotate \"Node $s2 sends Join towards Node $c1.\""
428   $ns at 2.00 "$pimNode($s2) join-group $rcvr2 $pimGroup0 $metricaS2_v1
      $pimNode($c1)"
429
430   $ns at 2.00 "$ns set-animation-rate 0.2ms"
431   $ns at 2.00 "$ns trace-annotate \"Node $s3 decides to join source ($c1)
      tree...\""
```

```
432  $ns at 2.00 "$ns trace-annotate \"Node $s3 sends Join towards Node $c1.\""
433  $ns at 2.00 "$pimNode($s3) join-group $rcvr3 $pimGroup0 $metricaS3_v1
       $pimNode($c1)"
434
435  $ns at 2.00 "$ns set-animation-rate 0.2ms"
436  $ns at 2.00 "$ns trace-annotate \"Node $s4 decides to join source ($c1)
       tree...\""
437  $ns at 2.00 "$ns trace-annotate \"Node $s4 sends Join towards Node $c1.\""
438  $ns at 2.00 "$pimNode($s4) join-group $rcvr4 $pimGroup0 $metricaS4_v1
       $pimNode($c1)"
439
440 #  3' evento :: çãActualizao das émtricas de todos os servidores.
441  $ns at 4.00 "$pimNode($s1) update-group $rcvr1 $pimGroup0 $metricaS1_v2
       $pimNode($c1)"
442  $ns at 4.00 "$pimNode($s2) update-group $rcvr2 $pimGroup0 $metricaS2_v2
       $pimNode($c1)"
443  $ns at 4.00 "$pimNode($s3) update-group $rcvr3 $pimGroup0 $metricaS3_v2
       $pimNode($c1)"
444  $ns at 4.00 "$pimNode($s4) update-group $rcvr4 $pimGroup0 $metricaS4_v2
       $pimNode($c1)"
445
446 #  4' evento :: cbr7 çcomea a fazer pedidos.
447  $ns at 6.00 "$ns set-animation-rate 0.2ms"
448  $ns at 6.00 "$ns trace-annotate \"Source at Node $c1 starts sending... \""
449  $ns at 6.00 "$ns trace-annotate \"Node $c1 forwards CBR packets to Group.\"
       "
450  $ns at 6.00 "$cbr7 start"
451
452  $ns at 6.00 "$ns set-animation-rate 0.2ms"
453  $ns at 6.00 "$ns trace-annotate \"Node $s1 decides to join source ($c1)
       tree...\""
454  $ns at 6.00 "$ns trace-annotate \"Node $s1 sends Join towards Node $c1.\""
455  $ns at 6.00 "$pimNode($s1) join-group $rcvr1 $pimGroup0 $metricaS1_v3
       $pimNode($c2)"
456
457  $ns at 6.00 "$ns set-animation-rate 0.2ms"
458  $ns at 6.00 "$ns trace-annotate \"Node $s2 decides to join source ($c1)
       tree...\""
459  $ns at 6.00 "$ns trace-annotate \"Node $s2 sends Join towards Node $c1.\""
```

```
460   $ns at 6.00 "$pimNode($s2) join-group $rcvr2 $pimGroup0 $metricaS2_v3
          $pimNode($c2)"
461
462   $ns at 6.00 "$ns set-animation-rate 0.2ms"
463   $ns at 6.00 "$ns trace-annotate \"Node $s3 decides to join source ($c1)
          tree...\""
464   $ns at 6.00 "$ns trace-annotate \"Node $s3 sends Join towards Node $c1.\""
465   $ns at 6.00 "$pimNode($s3) join-group $rcvr3 $pimGroup0 $metricaS3_v3
          $pimNode($c2)"
466
467   $ns at 6.00 "$ns set-animation-rate 0.2ms"
468   $ns at 6.00 "$ns trace-annotate \"Node $s4 decides to join source ($c1)
          tree...\""
469   $ns at 6.00 "$ns trace-annotate \"Node $s4 sends Join towards Node $c1.\""
470   $ns at 6.00 "$pimNode($s4) join-group $rcvr4 $pimGroup0 $metricaS4_v3
          $pimNode($c2)"
471
472 #  5' evento :: rcvr1 abandona o grupo.
473   $ns at 8.00 "$ns set-animation-rate 0.2ms"
474   $ns at 8.00 "$ns trace-annotate \"Node $s1 leaves group...\""
475   $ns at 8.00 "$pimNode($s1) leave-group $rcvr1 $pimGroup0"
476
477 #  6' evento :: cbr6 finaliza o seu pedido e o cbr8 çcomea a fazer pedidos.
478   $ns at 10.00 "$ns set-animation-rate 0.2ms"
479   $ns at 10.00 "$ns trace-annotate \"Source at Node $c1 stops sending... \""
480   $ns at 10.00 "$ns trace-annotate \"Node $c1 forwards CBR packets to Group.\
          ""
481   $ns at 10.00 "$cbr6 stop"
482
483   $ns at 11.00 "$ns set-animation-rate 0.2ms"
484   $ns at 11.00 "$ns trace-annotate \"Node $s2 leaves source $c1 only...\""
485   $ns at 11.00 "$pimNode($s2) leave-group $rcvr2 $pimGroup0 $pimNode($c1)"
486
487   $ns at 11.00 "$ns set-animation-rate 0.2ms"
488   $ns at 11.00 "$ns trace-annotate \"Node $s3 leaves source $c1 only...\""
489   $ns at 11.00 "$pimNode($s3) leave-group $rcvr3 $pimGroup0 $pimNode($c1)"
490
491   $ns at 11.00 "$ns set-animation-rate 0.2ms"
492   $ns at 11.00 "$ns trace-annotate \"Node $s4 leaves source $c1 only...\""
493   $ns at 11.00 "$pimNode($s4) leave-group $rcvr4 $pimGroup0 $pimNode($c1)"
```

```
494
495
496   $ns at 10.00 "$ns set-animation-rate 0.2ms"
497   $ns at 10.00 "$ns trace-annotate \"Source at Node $c3 starts sending... \""
498   $ns at 10.00 "$ns trace-annotate \"Node $c3 forwards CBR packets to Group.\
        ""
499   $ns at 10.00 "$cbr8 start"
500
501   $ns at 10.00 "$ns set-animation-rate 0.2ms"
502   $ns at 10.00 "$ns trace-annotate \"Node $s2 decides to join source ($c3)
        tree...\""
503   $ns at 10.00 "$ns trace-annotate \"Node $s2 sends Join towards Node $c3.\""
504   $ns at 10.00 "$pimNode($s2) join-group $rcvr2 $pimGroup0 $metricaS2_v4
        $pimNode($c3)"
505
506   $ns at 10.00 "$ns set-animation-rate 0.2ms"
507   $ns at 10.00 "$ns trace-annotate \"Node $s3 decides to join source ($c3)
        tree...\""
508   $ns at 10.00 "$ns trace-annotate \"Node $s3 sends Join towards Node $c3.\""
509   $ns at 10.00 "$pimNode($s3) join-group $rcvr3 $pimGroup0 $metricaS3_v4
        $pimNode($c3)"
510
511   $ns at 10.00 "$ns set-animation-rate 0.2ms"
512   $ns at 10.00 "$ns trace-annotate \"Node $s4 decides to join source ($c3)
        tree...\""
513   $ns at 10.00 "$ns trace-annotate \"Node $s4 sends Join towards Node $c3.\""
514   $ns at 10.00 "$pimNode($s4) join-group $rcvr4 $pimGroup0 $metricaS4_v4
        $pimNode($c3)"
515
516 #  7' evento :: Falha de link entre dois óns [x1 e x2]
517   $ns rtmodel-at 12.0 down $pimNode($x1) $pimNode($x2)
518
519 #  8' evento :: Link do 7'evento recupera. Outra falha de link entre dois óns
        [x1 e x2].
520   $ns rtmodel-at 14.0 up $pimNode($x1) $pimNode($x2)
521
522 #  9' evento :: rcvr2 e rcvr3 abandonam o grupo..
523   $ns at 16.00 "$ns set-animation-rate 0.2ms"
524   $ns at 16.00 "$ns trace-annotate \"Node $s2 leaves group...\""
525   $ns at 16.00 "$pimNode($s2) leave-group $rcvr2 $pimGroup0"
```

```
526
527   $ns at 16.00 "$ns set-animation-rate 0.2ms"
528   $ns at 16.00 "$ns trace-annotate \"Node $s3 leaves group...\""
529   $ns at 16.00 "$pimNode($s3) leave-group $rcvr3 $pimGroup0"
530
531 # 10' evento :: cbr7 e cbr 8 param de fazer pedidos.
532   $ns at 18.00 "$ns set-animation-rate 0.2ms"
533   $ns at 18.00 "$ns trace-annotate \"Source at Node $c2 stops sending... \""
534   $ns at 18.00 "$ns trace-annotate \"Node $c2 forwards CBR packets to Group.\
        \""
535   $ns at 18.00 "$cbr7 stop"
536
537   $ns at 18.00 "$ns set-animation-rate 0.2ms"
538   $ns at 18.00 "$ns trace-annotate \"Source at Node $c3 stops sending... \""
539   $ns at 18.00 "$ns trace-annotate \"Node $c3 forwards CBR packets to Group.\
        \""
540   $ns at 18.00 "$cbr8 stop"
541
542   $ns at 19.00 "$ns set-animation-rate 0.2ms"
543   $ns at 19.00 "$ns trace-annotate \"Node $s4 leaves source $c2 only...\""
544   $ns at 19.00 "$pimNode($s4) leave-group $rcvr4 $pimGroup0 $pimNode($c2)"
545
546   $ns at 19.00 "$ns set-animation-rate 0.2ms"
547   $ns at 19.00 "$ns trace-annotate \"Node $s4 leaves source $c3 only...\""
548   $ns at 19.00 "$pimNode($s4) leave-group $rcvr4 $pimGroup0 $pimNode($c3)"
549
550 # 11' evento :: rcvr4 abandona o grupo.
551   $ns at 19.50 "$ns set-animation-rate 0.2ms"
552   $ns at 19.50 "$ns trace-annotate \"Node $s4 leaves group...\""
553   $ns at 19.50 "$pimNode($s4) leave-group $rcvr4 $pimGroup0"
554
555 # 12' evento :: Fim da çãsimuao.
556   $ns at 20.00 "$ns trace-annotate \"Simulation finished... \""
557   $ns at 20.00 "finish"
558
559 proc finish {} {
560   global ns pimTraceFile pimNamFile
561   $ns flush-trace
562   close $pimTraceFile
563   close $pimNamFile
```

```
564     exit 0
565 }
566 $ns run
```