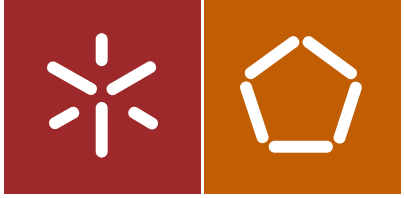
Universidade do Minho
Escola de Engenharia

Ana Margarida Xavier Ferreira Fernandes

Comunicação em Grupo: Multicast Aplicacional versus Multicast IP

outubro de 2015

Comunicação em Grupo: Multicast Aplicacional versus Multicast IP
Ana Margarida Xavier Ferreira Fernandes
UMinho | 2015



Universidade do Minho
Escola de Engenharia

Ana Margarida Xavier Ferreira Fernandes

Comunicação em Grupo: Multicast Aplicacional versus Multicast IP

Dissertação de Mestrado
Ciclo de Estudos Integrados Conducentes ao Grau de
Mestre em Engenharia de Telecomunicações e Informática

Trabalho efectuado sob a orientação de
Professora Doutora Maria João Nicolau
Professor Doutor Alexandre Santos

outubro de 2015

AGRADECIMENTOS

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

Começo por agradecer aos meus orientadores, Professora Doutora Maria João Nicolau e Professor Doutor Alexandre Santos, pelo seu apoio, dedicação e paciência demonstrados, até mesmo nos momentos mais difíceis.

Gostaria de agradecer aos meus amigos, Sara, Silvia, Vera, Nuno e Renato, pela amizade e pelas palavras de incentivo, que sempre me animaram.

Aos meus pais, irmão e restante família, pela inestimável confiança e compreensão demonstradas nesta fase da minha via. Ao meu avô pelas palavras de sabedoria e incentivo.

Por último, ao Hugo, pelo seu apoio incondicional, companheirismo e força para continuar a superar todas as adversidades. Obrigada por acreditares sempre nas minhas capacidades e vibrares com as minhas vitórias.

RESUMO

Nos últimos anos, o *Internet Protocol Television (IPTv)* tem vindo a aumentar a sua popularidade junto dos principais *Internet Service Providers (ISPs)*, que o utilizam de forma a modernizar as suas redes de partilha de conteúdos audiovisuais. Este tipo de aplicações utilizam comunicação em grupo, sobre redes IP (*Internet Protocol*), que requerem muita largura de banda e por isso, a escolha da melhor tecnologia para as suportar é de extrema importância. O *multicast* é a principal tecnologia para permitir uma comunicação eficiente para este tipo de rede, devido ao facto de ter sido desenhado para comunicações de um-para-muitos.

No *multicast* implementado ao nível da camada de rede, o *multicast* IP, os *routers* são responsáveis pela subscrição dos utilizadores nos diferentes grupos e de fazer chegar o tráfego para os nós interessados. Apesar da crença que o *multicast* IP é a arquitetura mais eficiente para distribuir informação para diferentes grupos, a sua lenta implementação levou a que alguns investigadores questionassem se a camada de rede é, necessariamente, a melhor camada para implementar a funcionalidade de *multicast*. Assim, surgiu o *multicast* na camada de aplicação (*Application Layer Multicast* - ALM), como uma técnica alternativa para o *multicast*. Nesta abordagem, a funcionalidade *multicast* é implementada nos sistemas finais, em vez de nos *routers* da rede.

A presente dissertação começa por efetuar um levantamento dos principais protocolos do paradigma *multicast*, ao nível da camada de rede e da camada aplicacional. Depois estabelece plataformas de simulação para *multicast* aplicacional e de nível de rede. Por fim, analisa o desempenho de uma solução *multicast* de cada camada, num cenário de teste, com recurso à simulação, de modo a contribuir para a longa discussão sobre qual a solução mais adequada, para ser implementada em larga escala.

Palavras-Chave

IPTv

Multicast

Multicast IP

ALM

ABSTRACT

In the past few years, Internet Protocol Television (IPTv) popularity has been growing among the Internet Service Providers, in order to modernize its networks to deliver audiovisual content. This type of application use group communication over IP networks, that require high bandwidth and so, choosing the best technology to support them, is extremely important. Multicast is the main technology to allow an efficient communication for this type of network, due to being designed for communications one-to-many.

In IP multicast (multicast implemented at the network level), routers are meant to make/remove subscription from users to different groups and replicate traffic to interested nodes. Despite the belief that IP multicast is the most efficient architecture for data distribution to multiple groups, their slow implementation led some researchers to question whether the network layer is necessarily the best layer for implementing multicast functionality. Therefore, application layer multicast (ALM) has emerged as an alternative technique for multicasting. In this approach, the multicast functionality is implemented at the end-systems instead of the network routers.

This dissertation begins by presenting a survey of multicast protocols at the network level and application level. Then sets simulation platforms for application layer multicast and IP multicast. Finally, it analyzes the performance of a multicast solution on both layers, using the simulation of a test scenario, in order to contribute to the long discussion about the most appropriate solution to implement multicast on a large scale network.

Keywords

IPTv

Multicast

IP Multicast

ALM

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÓNIMOS

ALM	Application Layer Multicast
ASM	Any Source Multicast
CBT	Core Based Tree
DVMRP	Distance Vector Multicast Routing Protocol
DR	Designated Router
GUI	Graphical User Interface
IANA	Internet Assigned Numbers Authority
IGMP	Internet Group Management Protocol
IP	Internet Protocol
IPV4	Internet Protocol version 4
IPV6	Internet Protocol version 6
IPTV	Internet Protocol Television
ISP	Internet Service Provider
LSA	Link State Advertisements
MOSPF	Multicast Open Shortest Path First
NED	Network Description
NS-2	Network Simulator 2
NS-3	Network Simulator 3
OMNeT++	Objective Modular Network Testbed in C++
OPNET	Optimized Network Engineering Tools
OSPF	Open Shortest Path First
P2P	Peer-to-Peer
PIM-DM	Protocol Independent Multicast – Dense Mode
PIM-SM	Protocol Independent Multicast – Sparse Mode

RP	Rendez - Vous Point
RPF	Reverse Path Forwarding
SSM	Source Specific Multicast
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

1 INTRODUÇÃO

1.1 Enquadramento temático

Hoje em dia, como os computadores e tecnologias estão sempre a evoluir, uma grande variedade de serviços de rede estão, também, a ser rapidamente desenvolvidos. Dentro destas tecnologias, os serviços de *Internet Protocol Television* (IPTv) têm-se tornado o recente método de transmissão para entregar vídeo e áudio às casas dos consumidores. O rápido crescimento dos serviços IPTv deve-se ao facto de que permite aos *Internet Service Providers* (ISPs) reforçarem a sua competitividade através de novos serviços (possibilidade num único pacote de serviço ter *digital voice*, TV e *data*) e oferece aos utilizadores maior flexibilidade e interatividade e cria oportunidades para um leque grande de novas aplicações [1].

O IPTv é definido como um serviço de multimédia que codifica transmissões de TV ao vivo, numa série de pacotes IP e fornece-os aos utilizadores, através de redes de banda larga residenciais [1]. Numa transmissão IPTv, uma fonte partilha conteúdo multimédia com um grupo de utilizadores, em larga escala. Na Figura 1.1 encontra-se um exemplo de uma comunicação em grupo, onde os canais de televisão são transmitidos para os utilizadores interessados. De notar que apesar de haver dois utilizadores interessados no canal RTP, este só é transmitido uma vez pelo servidor.

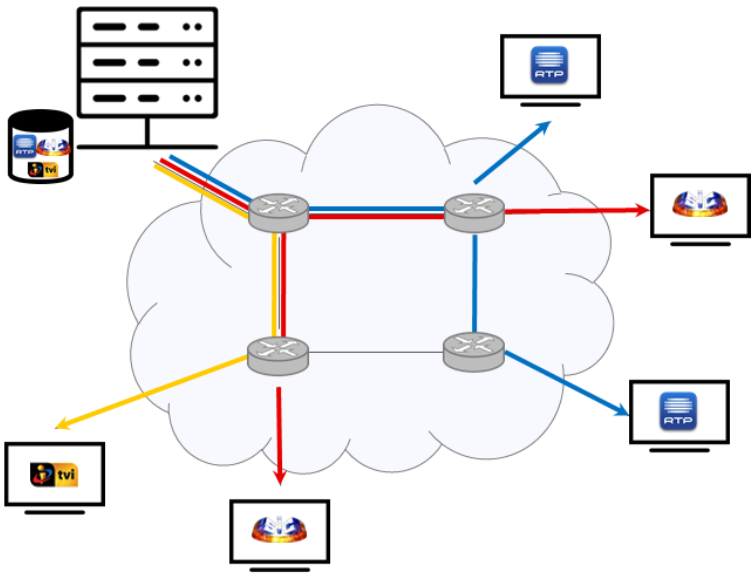


Figura 1.1 - Comunicação em grupo em larga escala

Desde há alguns anos atrás, o mercado para serviços de IPTv tem crescido rapidamente e a tendência é continuar. O gráfico da Figura 1.2 mostra a evolução do número total de assinaturas de IPTv em todo o mundo desde 2009 até 2014 [2]. No fim de 2009 havia aproximadamente 34 milhões de assinaturas e em 2014 este número aumentou para aproximadamente 117 milhões. Este facto demonstra o quão rápido está a ser o crescimento de subscrições de IPTv em todo o mundo, impulsionado pelo facto dos ISPs quererem fornecer o melhor serviço aos consumidores, estando a modernizar as suas redes.

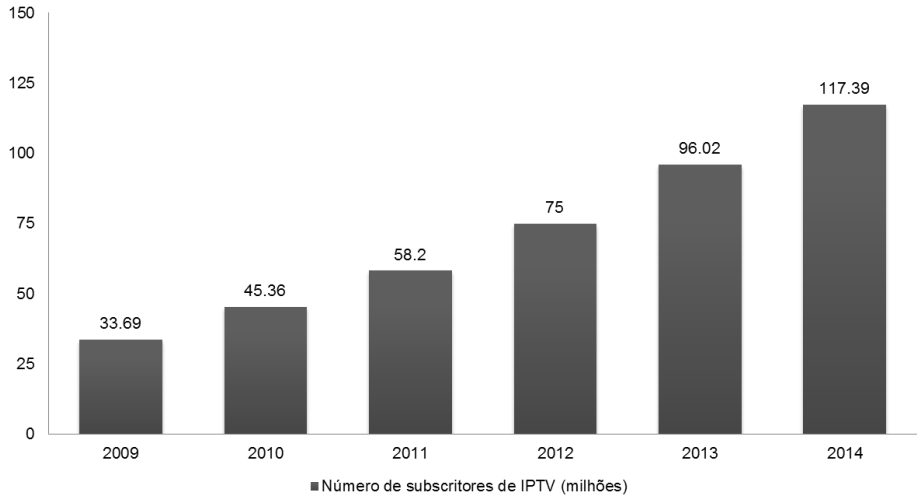


Figura 1.2 - Número de assinaturas de IPTv em todo o mundo desde 2009 até 2014 [2]

Com a crescente popularidade deste tipo de aplicações que implicam a utilização em larga escala da comunicação em grupo, a escolha da melhor arquitetura para suportar este tipo de aplicações tem uma grande importância.

Nas redes IP, o principal mecanismo de encaminhamento é o *unicast*, que se caracteriza por uma ligação ponto-a-ponto (um pacote é enviado para um único dispositivo). No entanto, durante os últimos anos, têm surgido aplicações que requerem a comunicação eficiente entre grupos de dispositivos, aumentando a popularidade da transmissão *multicast* [3]. No *multicast*, a informação é transmitida para um único endereço *multicast* e é recebida por qualquer dispositivo que requer a obtenção dessa informação. O *multicast* pode ser implementado tanto na camada de rede como na camada de aplicação.

No *multicast* na camada de rede (*Multicast* IP) [4], a fonte envia os pacotes de dados apenas uma vez, mesmo que estes tenham que ser entregues a um grande número de utilizadores. Os *routers* da rede têm a função de replicar os pacotes, sempre que os caminhos divergem, de forma a atingir todos os recetores e estes pacotes são enviados apenas uma vez em cada ligação.

No *multicast* da camada de aplicação (*Application Layer Multicasting* - ALM) [5] , a funcionalidade do *multicast* é implementada como um serviço de aplicação em vez de um serviço de rede. Sendo assim, a replicação dos pacotes e o encaminhamento são assegurados pelos *hosts* finais participantes, em vez de ser pelos *routers*, como no caso do *multicast* IP. Os *hosts* finais formam uma rede de *overlay* (sobreposição), onde cada caminho da rede corresponde a um caminho *unicast* direto, entre dois membros do grupo *multicast*.

1.2 Objetivos

O principal objetivo deste trabalho consiste em estabelecer plataformas de simulação para avaliar e comparar a utilização do *multicast* tanto ao nível da camada de rede como da camada de aplicação, para suportar aplicações que exigem comunicação em grupo. Nesse sentido, o trabalho desenvolvido ao longo desta dissertação incidiu na realização dos seguintes objetivos:

- Compreender o funcionamento do paradigma de comunicação multicast para comunicação em grupo.
- Estudar aprofundadamente os principais protocolos de encaminhamento multicast e das principais soluções existentes para multicast no nível aplicacional.
- Escolher um protocolo multicast da camada de rede e da camada de aplicação para simulação.
- Estabelecer plataformas de simulação para multicast aplicacional e de nível da rede.
- Testar e avaliar o protocolo multicast IP e o protocolo ALM, no contexto de uma comunicação em grupo.

1.3 Conteúdo da dissertação

O presente documento está estruturado em cinco capítulos, contendo ainda mais dois anexos, relativos ao trabalho desenvolvido.

O primeiro capítulo contém uma breve introdução ao tema desta dissertação, a definição dos principais objetivos e a descrição da estrutura, adotada na elaboração da dissertação.

O capítulo 2 começa por apresentar uma secção de contextualização à comunicação em grupo e, depois, expõe o estado da arte relativamente ao encaminhamento multicast, na camada de rede e na camada de aplicação, através do estudo das propostas existentes.

No capítulo 3 é realizada uma breve descrição dos simuladores mais utilizados na área das redes e ainda apresenta as funcionalidades do simulador escolhido. Também são descritas as *frameworks* inerentes ao simulador.

No quarto capítulo são apresentadas as simulações e resultados provenientes do cenário avaliado. É apresentada a topologia que serve de base para o cenário de teste e ainda são descritos os parâmetros de avaliação de desempenho.

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

2 COMUNICAÇÃO EM GRUPO: MULTICAST APLICACIONAL E MULTICAST IP

2.1 Introdução

Atualmente, têm surgido, com o aumento do uso da Internet, inúmeras aplicações que envolvem comunicação em grupo, nomeadamente, videoconferência, jogos *online*, ensino à distância e o foco desta dissertação, serviços IPTv. Para a implementação deste tipo de serviço, a tecnologia *multicast* é sempre a tecnologia escolhida ao invés do *unicast* [3].

Quando um pacote é enviado entre dois *hosts*, ou seja, existe um remetente e um destinatário, é utilizado o *unicast*. Para cada cliente é enviado um pacote individual do servidor. No caso do envio de um mesmo pacote para vários utilizadores, com o *unicast*, o mesmo pacote é enviado várias vezes, até que atinja todos os utilizadores (Figura 2.1). Este processo desperdiça largura de banda, portanto, neste caso, o unicast é ineficiente.

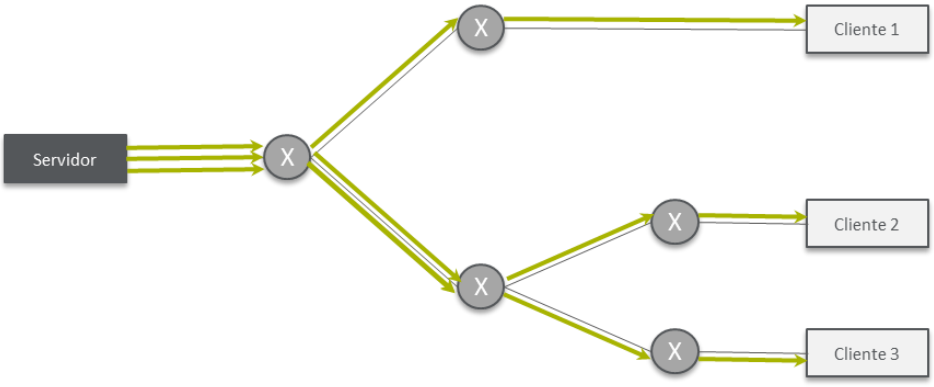


Figura 2.1 - *Streaming unicast*

No *multicast*, o servidor envia um único *stream* de dados para vários utilizadores. Os pacotes de dados são replicados na rede, quando os caminhos divergem, resultando numa entrega de informação muito mais eficiente (Figura 2.2). Sendo assim, a tecnologia *multicast* é mais eficiente pois utiliza uma estrutura de árvore para entregar a informação, com o objetivo de evitar a transmissão de pacotes duplicados na rede. Este facto permite reduzir o congestionamento na rede, aumentando o desempenho global desta. O uso eficiente da rede e a redução da carga nas fontes de tráfego permitem disponibilizar serviços a um maior número de clientes do que o encaminhamento *unicast*, pois necessitam de menos recursos

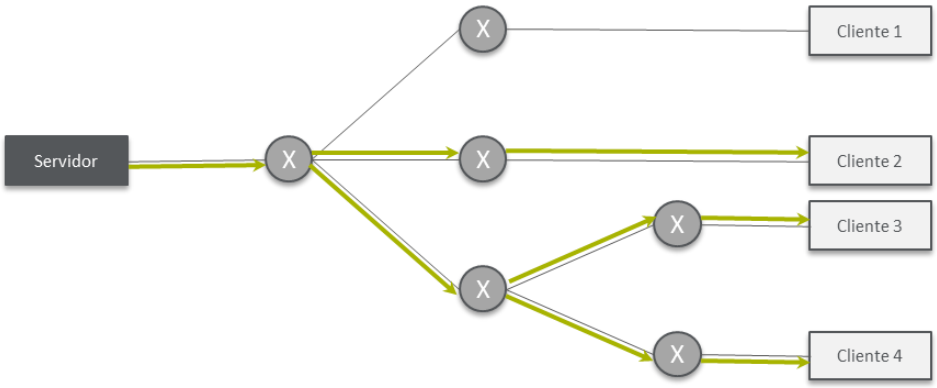


Figura 2.2 - *Streaming multicast*

O *multicast* é baseado no conceito de grupo. Um grupo *multicast* é um grupo arbitrário de recetores que expressam interesse em receber um fluxo de dados em particular. Os *hosts* que estiverem interessados em receber o fluxo de dados de um grupo têm que se juntar a este, fazendo a subscrição no grupo. A constituição dos membros dos grupos é dinâmica, ou seja, os *hosts* podem entrar e sair dos grupos sempre que quiserem.

O *multicast* implementado ao nível da camada de rede, *multicast* IP, pressupõe que os *routers* da rede suportam o protocolo. Estes formam, entre si, uma estrutura em árvore, onde os pacotes são replicados para os nós interessados. Os *routers* são responsáveis por efetuar/anular a subscrição nos diferentes grupos, sempre que possuem utilizadores interessados (estejam conectados direta ou indiretamente). Apesar da sua eficiência, o *multicast* IP está longe de ser largamente implementado devido à falta de um protocolo de encaminhamento *multicast* interdomínio escalável e a necessidade da implementação global de *routers* capazes de *multicast* IP.

Apesar do *multicast*, implementado na camada de rede, ser mais eficiente, a sua lenta implementação e a crescente necessidade de suporte de aplicações com múltiplos utilizadores, deram lugar ao aparecimento do *multicast* ao nível da camada de aplicação. No *multicast* a nível aplicacional, os *hosts* finais é que são responsáveis pela duplicação e encaminhamento dos pacotes de dados, em vez dos *routers* como no *multicast* IP. Esta diferença facilita a sua adoção pois não obriga os *routers* a suportarem o *multicast* IP. Os *hosts* finais formam uma rede de *overlay*, em que cada ligação corresponde a um caminho *unicast* direto entre eles.

Todos os pacotes de dados são enviados como pacotes *unicast* e encaminhados de um membro da rede *overlay* para outro, de acordo com as regras do protocolo *unicast* utilizado.

2.2 *Multicast* na camada de rede

A entrega dos serviços de multimédia aos consumidores pode ser bastante facilitada pelo uso da comunicação *multicast*. Quando existe uma sobreposição de caminhos em direção a vários destinos, apenas uma cópia de um pacote de dados será enviada através de qualquer conexão de rede nesse caminho. Quando os caminhos são diferentes para chegarem aos utilizadores, os pacotes são replicados no *router* e são enviados para as ligações respetivas.

A Figura 2.3 mostra que a transmissão de um pacote de dados na rede é replicada onde for necessário.

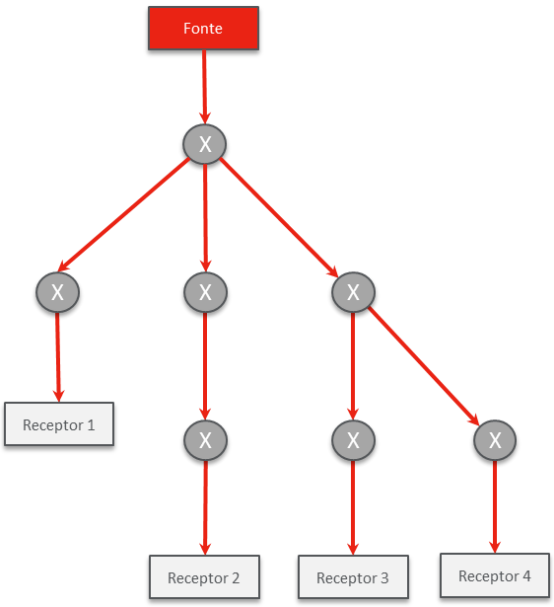


Figura 2.3 - Transmissão de informação no *multicast* IP

Os benefícios do uso do *multicast* são de fácil perceção. O *multicast* reduz a carga do servidor, visto que este só tem que enviar um pacote por ligação, em vez de enviar múltiplos pacotes para os diferentes recetores. Para além desta vantagem, o *multicast* reduz a carga em toda a rede, visto que só uma cópia do pacote de dados está a ser transmitida na rede. Estas características reduzem significativamente o consumo da largura de banda e processamento em toda a rede.

O primeiro modelo *multicast* foi proposto e descrito por Steven Deering [6], no final dos anos 80 e desde o início dos anos 90 que tem sido testado e implementado no *Multicast Backbone* (MBone). Este foi criado pois, na altura, os *routers*, na sua maioria, não tinham o *multicast* ativo, funcionando como uma rede paralela. S. Deering propôs o modelo *multicast* na camada de rede com base na noção de um grupo. Neste sistema, todos os *hosts* que estiverem interessados formam um grupo *multicast* para comunicarem uns com os outros. Cada grupo *multicast* é identificado por um endereço *multicast* especial, endereço de classe D, no caso do IPv4 e, no caso do IPv6, o endereço tem o prefixo FF00::/8. Cada *host*, que quiser receber informação, tem que se juntar a um grupo *multicast* particular, contactando o *router* mais próximo, através do protocolo *Internet Group Management Protocol* (IGMP). Este protocolo é explicado na secção 2.2.3. No caso do IPv6, é utilizado o protocolo *Multicast Listener Discovery* (MLD). Depois de se juntar a um grupo *multicast*, o *host* já consegue receber a informação que é enviada para esse grupo. Atualmente, a arquitetura *multicast* consiste em protocolos de gestão de grupos, protocolos de encaminhamento e protocolos de transporte.

2.2.1 Endereçamento *Multicast*

A comunicação no *multicast* é baseada no conceito de grupo, onde cada *host* utiliza o endereço do grupo para transmitir/receber informação.

Atualmente, existem dois protocolos para comunicações na Internet a funcionar simultaneamente, o *Internet Protocol version* 4 (IPv4) e o *Internet Protocol version* 6 (IPv6) [7]. O IPv6 introduz uma série de melhorias, no entanto, o principal fator para a sua utilização tem sido a falta de endereços IPv4.

A sua lenta adoção faz com que o endereçamento preferencial no *multicast* ainda seja o IPv4 e por isso, o IPv4 é o único protocolo falado neste documento.

Endereços IPv4

Nos endereços IPv4 existem cinco classes de endereços, A, B, C, D e E, em que as três primeiras classes são para identificar endereços IPv4 *unicast*, a classe D para endereços IPv4 *multicast* e a E encontra-se reservada para uso futuro.

Os endereços IPv4 *multicast* são atribuídos diretamente pela *Internet Assigned Numbers Authority* (IANA) [8]. A gama de endereços desde 224.0.0.0 até 239.255.255.255 representa os endereços IPv4 *multicast* dentro do espaço de endereçamento IPv4. Um endereço IPv4 *multicast* (Figura 2.4) é composto por um conjunto de 4 bits mais significativos que define a gama de endereços IPv4 *multicast* e por um conjunto de 28 bits menos significativos que identificam o grupo *multicast*.

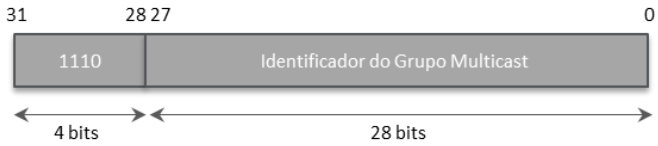


Figura 2.4 - Formato do endereço IPV4 *multicast*

A gama de endereços IPv4 *multicast* é dividida noutras gamas de acordo com o alcance dos endereços IPv4 *multicast*, sendo este alcance os limites onde os grupos *multicast* são válidos.

A Tabela 2.1 descreve as gamas de endereços IPv4 *multicast*, de acordo com [8].

Tabela 2.1 - Gama de endereços IPv4 *multicast* [adaptado de [8]]

Gama de Endereços			Designação
224.0.0.0	-	224.0.0.255	*Local Network Control Block*
224.0.1.0	-	224.0.1.255	*Internetwork Control Block*
224.0.2.0	-	224.0.255.255	*AD-HOC Block I*
224.1.0.0	-	224.1.255.255	*RESERVED*
224.2.0.0	-	224.2.255.255	*SDP/SAP Block*
224.3.0.0	-	224.4.255.255	*AD-HOC Block II*
224.5.0.0	-	244.255.255.255	*RESERVED*
225.0.0.0	-	231.255.255.255	*RESERVED*
232.0.0.0	-	232.255.255.255	*Source –Specific Multicast Block*
233.0.0.0	-	233.251.255.255	*GLOP Block*
233.252.0.0	-	233.255.255.255	*AD-HOC Block III*
234.0.0.0	-	238.255.255.255	*RESERVED*
239.0.0.0	-	239.255.255.255	*Administratively Scoped Block*

2.2.2 Modelos de Serviço *Multicast* IP

Um modelo de serviço multicast define a forma como os nós podem usufruir do multicast IP. Existem dois modelos atuais: ASM e SSM.

O modelo *Any Source Multicast* (ASM) [6] é conhecido como o modelo *multicast* mais tradicional (muitos-para-muitos). É um modelo aberto onde qualquer *host* pode juntar-se ou sair de um dado grupo *multicast*, quando o pretender. Neste modelo, a fonte envia pacotes para o endereço de grupo e estes são entregues a todos os membros do grupo. A fonte não tem que ser necessariamente um membro deste grupo. Este modelo suporta tanto as árvores centradas na fonte como as árvores partilhadas. Estes dois conceitos são explicados na secção 2.2.4.

O modelo ASM contém alguns problemas como colisões de endereços pois um endereço de um grupo *multicast* pode possivelmente ser utilizado por outro grupo *multicast*. Sendo difícil garantir que um endereço *multicast* é único. Outro problema está no facto dos recetores não controlarem as fontes que estão a emitir tráfego, podendo ser afetados por ataques de *flooding*.

O novo modelo de serviço *multicast*, *Source Specific Multicast* (SSM) [9], veio resolver os problemas do modelo ASM. Neste modelo, o tráfego é entregue através de uma só fonte para os recetores do grupo (um-para-muitos), desta forma, um *host* identifica um canal

com uma só fonte e vários recetores (S, G). Aqui o S é o endereço da fonte e G é o endereço do recetor. Quando um recetor quiser receber tráfego *multicast* da fonte (S), tem que subscrever o canal (S,G). Depois a fonte (S) envia o tráfego para os recetores através de uma árvore centrada na fonte. Para cada grupo a fonte é diferente, o que resolve o problema de colisão de endereços do modelo ASM. No SSM, não é permitido que nenhuma outra fonte, além de S, envie tráfego no mesmo canal (S,G). Desta forma, este modelo simplifica o encaminhamento *multicast* interdomínio pois não são necessárias árvores partilhadas.

2.2.3 Protocolos de gestão de grupos

Na arquitetura *multicast*, os protocolos de gestão de grupos são utilizados pelos *hosts* para reportarem, aos *routers multicast* da sub-rede, informação dos membros dos grupos, quando se juntam ou abandonam um grupo. O protocolo IGMP é utilizado no *multicast* como o protocolo de gestão de grupos. O protocolo IGMP tem três versões, IGMPv1, IGMPv2 e IGMPv3.

IGMP Versão 1

Nesta versão [6], os membros só se podem juntar aos grupos *multicast*, não havendo mecanismo para sair deles. De forma a conhecer quais os membros que deixaram de estar interessados no tráfego de um grupo, utiliza-se um mecanismo baseado em *timeout*. Como este protocolo não suporta a capacidade dos *hosts* saírem de um grupo, não é o mais indicado para ser usado em aplicações de IPTv.

IGMP Versão 2

Na versão 2 do protocolo IGMP [10] existe a oportunidade de um membro sair de um grupo, o que não existia na versão 1 deste protocolo. Esta é a principal característica em aplicações de IPTv, onde a largura de banda é fundamental e o encaminhamento de grupos *multicast* inativos, através de uma conexão, pode levar a situações de congestão e consequentemente pode degradar a qualidade de vídeo.

O protocolo IGMPv2 é desenhado para suportar redes *Any Source Multicast* (ASM). Numa rede ASM, o host IGMP especifica o grupo *multicast* que se deseja juntar e fica à escuta de todo o tráfego desse grupo, independentemente de quem está a envia-lo.

IGMP Versão 3

A principal melhoria desta versão [11] reside no facto que esta suporta o modelo *Single Source Model* (SSM). Com este modelo, o host especifica o endereço de origem através do qual está interessado receber tráfego. Esta característica é uma importante melhoria na segurança pois previne que os clientes recebam tráfego gerado por outros *hosts* na rede.

2.2.4 Algoritmos de encaminhamento *multicast*

Os algoritmos de encaminhamento *multicast* têm como função calcular os caminhos, para a transmissão de informação, com base na rede, sendo esta um grafo e com base num conjunto de requisitos, como o número de saltos, menor custo, etc.

Estes protocolos devem ter em atenção vários objetivos como a minimização da carga introduzida na rede (evitar ciclos, replicações, concentrações de tráfego nalguns caminhos, etc), suporte para comunicações fiáveis, seleção das rotas ótimas, minimização da quantidade de informação de estado que é armazenada nos *routers* (de forma a ser escalável), além de deverem garantir que os dados transmitidos cheguem apenas aos membros de um grupo.

Os algoritmos de encaminhamento constroem diferentes tipos de árvores de difusão: árvore centrada na fonte e árvore partilhada. Na árvore centrada na fonte, as fontes utilizam os caminhos mais curtos para atingirem os seus recetores. Na árvore partilhada, as fontes enviam primeiro a informação para o *router* core da rede e depois este envia para os recetores.

Árvore centrada na fonte

Na árvore centrada na fonte (Figura 2.5), a fonte age como a raiz da árvore, em que os ramos desta são os caminhos para os recetores. Desta forma, cada transmissão de informação para um grupo é uma árvore separada.

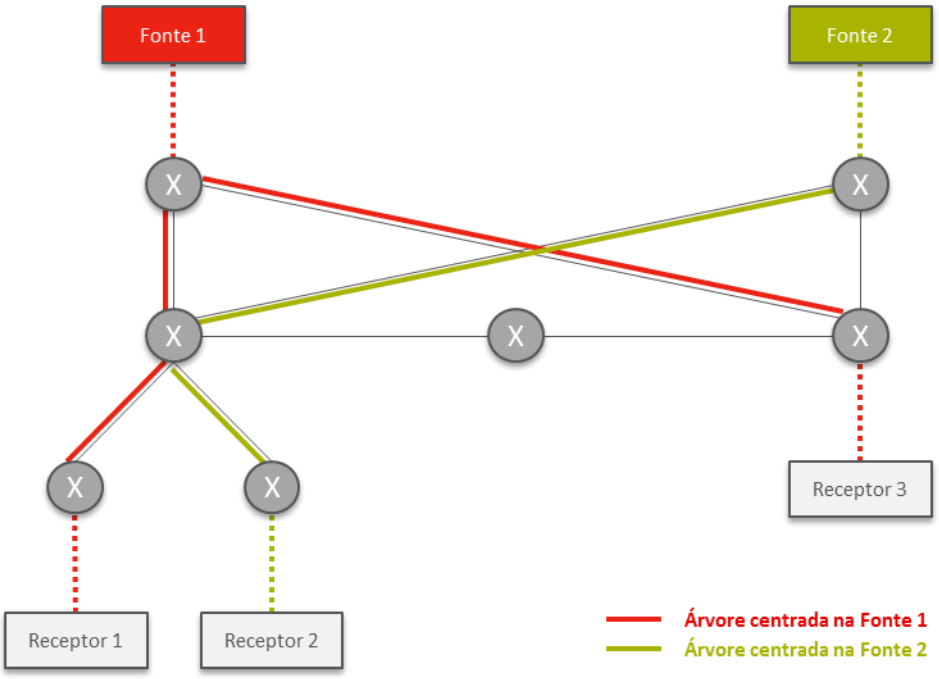


Figura 2.5 - Árvore centrada na fonte

Dentro deste tipo de árvores existem três tipos de estratégias de construção de árvores: *flooding*, árvores de caminhos inversos e árvores de caminhos mais curtos.

De acordo com a estratégia de *flooding*, um *router*, ao receber um datagrama *multicast*, apenas tem de verificar se é a primeira vez que o recebe. Se sim, encaminha o datagrama para todas as interfaces menos para a interface por onde o recebeu. Se não, o datagrama é descartado. Apesar de serem muito simples e robustos, os algoritmos, baseados neste tipo de estratégia, consomem muitos recursos de comunicação e memória e por isso tornam-se pouco escaláveis.

Na estratégia das árvores de caminhos inversos, quando um router recebe um datagrama *multicast*, verifica se este foi recebido através da interface utilizada para atingir o originador através do melhor caminho e, em caso afirmativo, propaga-o através de todas as interfaces, exceto aquela por onde chegou o datagrama. Se acontecer o contrário, o *router* descarta-o. Este mecanismo é conhecido como *Reverse Path Forwarding* (RPF). Depois do *flooding* com verificação RPF, são enviadas mensagens de *prune* pelos encaminhadores sem membros do grupo *multicast* ligados. As mensagens de *prune* permitem assim eliminar a

transmissão de pacotes nos ramos da árvore que não alimentam nenhum membro. Este processo denomina-se *flood and prune* e resulta numa árvore com os melhores caminhos de volta para a fonte.

Na estratégia das árvores de caminhos mais curtos, as árvores têm que ser construídas, obrigatoriamente, no mesmo sentido que o tráfego. Para isso, os routers têm que possuir conhecimento completo da topologia de rede e da constituição dos grupos, de forma a estabelecerem os melhores caminhos, desde cada uma das fontes, até aos diferentes destinatários. Como estas árvores utilizam os caminhos mais curtos para minimizar o atraso na rede, estas estruturas são apropriadas para áreas onde os membros dos grupos estão densamente distribuídos.

Árvore partilhada

Neste tipo de árvore (Figura 2.6), só existe uma raiz para todas as fontes no grupo e esta é colocada num lugar escolhido na rede. Esta raiz é chamada de *RendezVous Point* (RP). Neste tipo de árvore, o tráfego é enviado das fontes para o RP e, depois, é transmitido, através da árvore partilhada, para os recetores. Portanto, as árvores partilhadas têm atrasos maiores pois requerem que os pacotes sejam enviados primeiro para o RP e só depois é que são distribuídos. No entanto, as árvores partilhadas utilizam melhor os recursos pois aumenta a concentração do tráfego e, portanto, reduz a informação a ser guardada nas tabelas de encaminhamento dos *routers*.

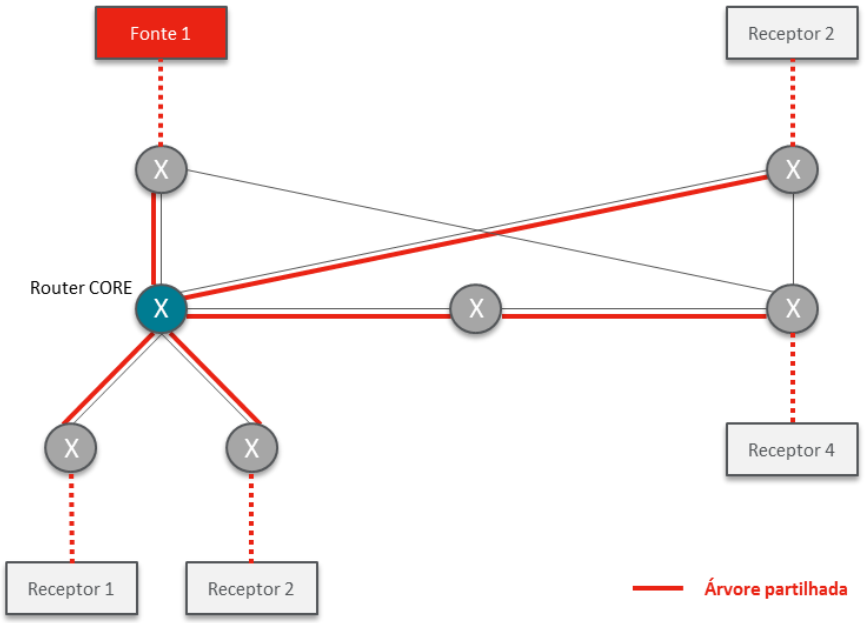


Figura 2.6 - Árvore partilhada

Dentro deste tipo de árvore existem dois tipos de abordagens: árvores de custo mínimo (Steiner) e árvores centradas num nó pré-definido.

A árvore de Steiner [12] é uma árvore que alcança todos os membros de um grupo e que minimiza o custo total da árvore. Relativamente a este último facto, esta árvore é diferente da árvore de caminhos mais curtos, que tem como objetivo minimizar o custo individual de cada um dos seus ramos e não o custo total da árvore. Na prática, este tipo de árvore não é utilizada, pois exige um conhecimento completo de toda a topologia, incluída a constituição do grupo multicast e é necessário recalcular a árvore sempre que um elemento entra/sai de um grupo.

Na abordagem de árvores centradas num nó pré-definido, os algoritmos estabelecem uma única árvore partilhada por todas as fontes e destinatários de um grupo multicast. Nesta estratégia existe um nó pré-definido (*core router* ou *Rendez-vous Point*) que age como o centro ou raiz da árvore. Quando um membro pretende juntar-se a um grupo, envia um pedido a esse nó pré-definido e todos os *routers* no caminho, ao receberem esse pedido, atualizam as suas tabelas de encaminhamento, de forma a construírem um novo ramo, que liga o novo membro à árvore.

2.2.5 Protocolos de encaminhamento multicast

Hoje em dia, existem muitos protocolos *multicast* disponíveis, em que alguns utilizam árvores centradas na fonte como o *Distance Vector Multicast Routing Protocol* (DVMRP), o *Multicast Open Shortest Path First* (MOSPF) e o *Protocol Independent Multicast – Dense Mode* (PIM-DM) e outros que utilizam as árvores partilhadas como o *Protocol Independent Multicast – Sparse Mode* (PIM-SM), o *Core Based Tree* (CBT). O PIM-SM possui a capacidade de alternar de uma árvore partilhada para uma árvore centrada na fonte, sempre que seja vantajoso.

Distance Vector Multicast Routing Protocol (DVMRP)

O protocolo DVMRP [13] utiliza o algoritmo de vetores de distância para construir a árvore *multicast*. Cada entrada no vetor contém a distância para determinada fonte. Esta distância é medida em número de saltos. Utiliza a estratégia de *flood and prune* com verificação RPF.

Este tipo de protocolo não é o ideal para comunicações em larga escala pois o mecanismo de *flooding* do conteúdo multimédia introduz uma sobrecarga em nós que não estão "interessados" na informação.

Multictast Open Shortest Path First (MOSPF)

O protocolo MOSPF [14] é uma extensão ao protocolo de encaminhamento *unicast* OPSF versão 2, para suportar encaminhamento *multicast*. No MOSPF, cada *router* tem sempre informação sobre a rede toda e os membros de cada grupo. Um *router* MOSPF distribui esta informação ao inundar (*flooding*) a rede com anúncios do estado das suas ligações (*Link State Advertisements - LSA*), periodicamente e, desta forma, os routers MOSPF ficam a saber a que routers estão ligados os membros dos diferentes grupos *multicast*. Com base na chegada do primeiro datagrama para um grupo, cada *router* constrói a árvore de caminhos mais curtos, cuja raiz é a fonte do datagrama.

Apesar da vantagem deste protocolo não inundar a rede com conteúdo multimédia originado por uma fonte, necessita de transmitir para todos os membros da rede o estado das

suas ligações e a constituição dos grupos *multicast*. Numa rede dinâmica, como o tipo da aplicação IPTV, onde os membros estão constantemente a entrar/sair do grupo, este protocolo origina uma carga extra na rede, pois os *routers* necessitam de atualizar a sua base de dados com a constituição de todos os grupos multicast, além da base de dados topológica, em cada alteração da rede.

Core Based Tree (CBT)

Neste protocolo [15], é designado um *router core* como a raiz da árvore partilhada para cada grupo. Os recetores interessados efetuam a subscrição na árvore centrada no *router core*. Quando uma fonte envia um datagrama para o grupo, o datagrama primeiro atinge o *router* core e depois, este envia-o para os seus vizinhos na árvore *multicast*. Após receber um datagrama, o *router* envia para as suas interfaces na árvore partilhada, exceto aquela por onde recebeu o datagrama.

O CBT reduz o nível de *overhead* na rede, pois não utiliza o mecanismo de *flooding*. No entanto, introduz um ponto único de falha na rede e aumenta o tráfego nos caminhos partilhados à volta do *router core*.

Protocol-Independent Multicast- Dense Mode (PIM-DM)

O protocolo PIM-DM [16] é semelhante ao DVMRP, pois também utiliza o mecanismo *flood and prune*, com verificação RPF. No entanto, distingue-se no facto, de que não utiliza um protocolo de encaminhamento unicast específico, enquanto o DVMRP utiliza o seu próprio protocolo de encaminhamento unicast. Este protocolo é apropriado para redes onde os recetores estão distribuídos de forma densa e próximos da fonte

Protocol-Independent Multicast- Sparse Mode (PIM-SM)

Este protocolo [17] foi criado para ser utilizado em redes de grandes dimensões e por isso a escalabilidade é uma das suas principais preocupações. Ao contrário de outros protocolos, não depende de um protocolo unicast específico, tendo por isso, na sua designação, a expressão *protocol independent* (independente do protocolo).

O PIM-SM constrói dois tipos de árvores: árvore partilhada e árvore centrada na fonte. Começa por construir uma árvore partilhada com raiz num ponto pré-definido, designado por Rendez-Vous Point (RP). Quando um host pretende enviar dados, envia-os por unicast para o RP. Por outro lado, quando um host pretende receber dados, o encaminhador mais próximo (Designated Router - DR) deve registar-se junto do RP, para os poder receber. O fluxo de dados flui assim da origem para o RP e deste para os diferentes destinos. Com base no débito gerado pelas fontes, a árvore partilhada dá lugar às árvores centradas nas fontes.

Uma árvore partilhada é mantida por uma entrada (*,G), que é criada quando um *router* recebe uma mensagem de *join* para um grupo G, que ainda não se encontra na tabela de encaminhamento. Depois da criação da entrada, o DR envia uma mensagem de *join* ao próximo router, na direção do RP.

Cada router, no caminho até ao RP, ao receber uma mensagem de *join*, cria ou atualiza uma entrada (*,G), na sua tabela de encaminhamento. Sempre que a entrada (*,G) é criada de novo, a mensagem de *join* é reenviada ao próximo nó, no caminho mais curto, até ao RP.

Quando uma fonte deseja enviar tráfego para um grupo, "encapsula" cada pacote de dados numa mensagem de registo e envia-a por unicast até ao RP. O RP "desencapsula" a mensagem e reenvia o pacote de dados, através da árvore partilhada, a todos os membros do grupo.

Um *router* DR, que tenha membros de um grupo diretamente ligados, começa por se juntar a uma árvore de distribuição partilhada e centrada num RP e, quando começar a receber dados das fontes, pode comutar para árvores centradas nestas. Quando decide comutar de árvore, o DR coloca uma entrada (S,G) (S – endereço da fonte; G – endereço do grupo) na sua tabela de encaminhamento. Uma vez criada a entrada, o DR envia uma mensagem *join* ao próximo *router*, em direção à fonte. Todos os *routers*, no trajeto até à fonte, criam ou atualizam as entradas (S,G).

Um *router*, ao receber o primeiro pacote através da árvore centrada na fonte, faz *prune* daquela fonte na árvore partilhada, ou seja, abandona esta árvore, para garantir que não recebe pacotes duplicados.

Na Figura 2.7 apresenta-se um exemplo com as duas árvores que são criadas no PIM-SM. Inicialmente é criada a árvore partilhada e depois o recetor 2 comuta para a árvore centrada na fonte.

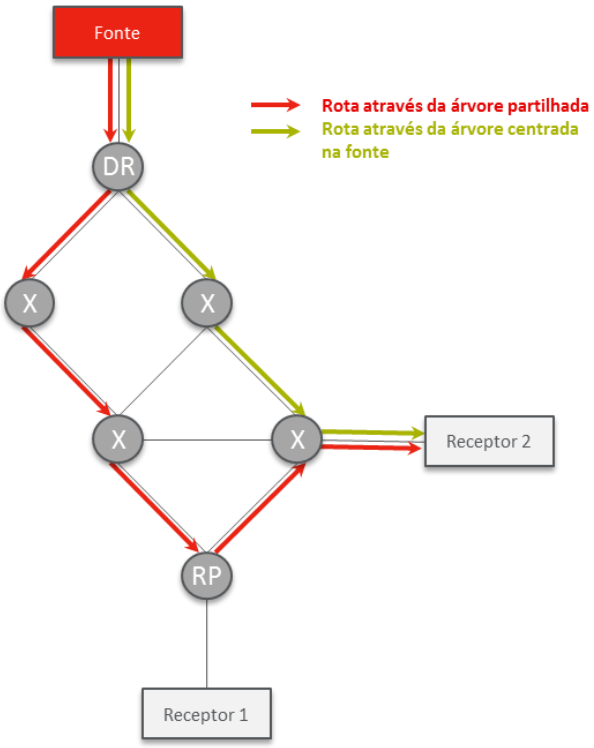


Figura 2.7 - Árvores criadas no PIM-SM

2.3 *Multicast* na camada de aplicação

Apesar do *multicast* da camada de aplicação ser posterior ao multicast IP, este tem despertado muito interesse nos investigadores. No *multicast* ao nível da aplicação, os processos inerentes ao *multicast*, como a formação dos grupos, a replicação dos pacotes e o encaminhamento *multicast*, são implementados nos *hosts* finais, ao contrário do que acontece ao nível da camada de rede. Este conceito consiste, simplesmente, na implementação da funcionalidade *multicast* como um serviço de aplicação em vez de um serviço de rede. Enquanto o *Multicast* IP é implementado por nós da camada de rede (*routers*) e evita várias cópias do mesmo pacote, na mesma ligação, através da construção de árvores eficientes, o ALM é implementado por nós da camada da aplicação (sistemas finais ou *proxies*), o que pode resultar em múltiplas cópias do mesmo pacote, na mesma ligação, assim como na construção de árvores pouco eficientes. Apesar destas desvantagens em relação ao *Multicast* IP, o ALM é mais fácil e rápido de implementar e tem a possibilidade de se adaptar a uma determinada aplicação [5].

A Figura 2.8 a) representa um exemplo de configuração ALM para o mesmo grupo de recetores e fonte do cenário de *multicast* IP, representado na Figura 2.8 b). No ALM, a árvore *multicast* é construída na camada de aplicação. Utilizando somente a capacidade de *unicast* da rede, a fonte envia dois pacotes, um para *Recetor 1* e outro para *Recetor 2* e cada um envia o pacote para o *Recetor 4* e *Recetor 3*, respetivamente.

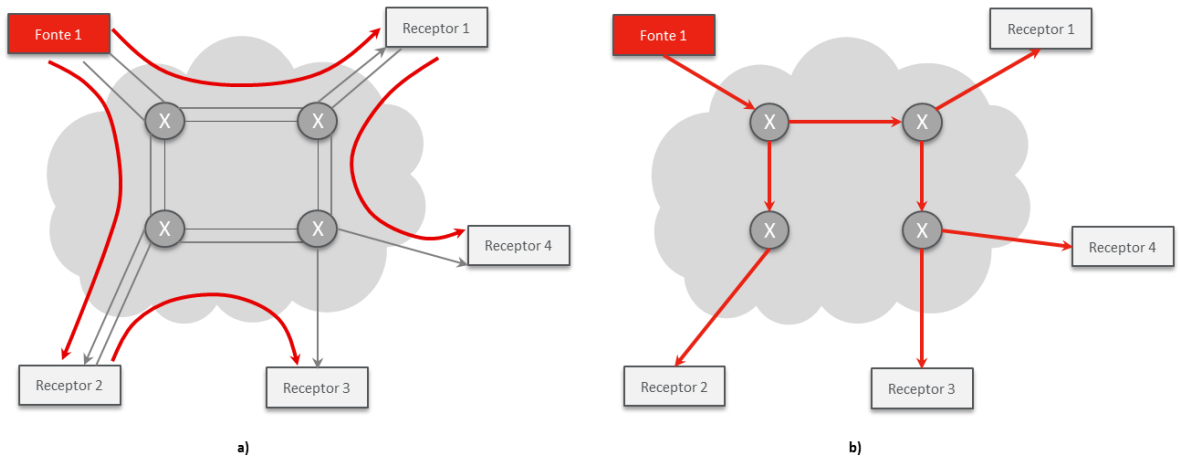


Figura 2.8 - Encaminhamento multicast a) na camada aplicacional b) na camada de rede

Desde a introdução do conceito de ALM, têm surgido inúmeros protocolos ALM com uma grande variedade de abordagens e características [18]. Estes diferem no domínio da aplicação, no nível de implementação do protocolo e na forma como o protocolo organiza os membros num grupo de comunicação, numa sessão *multicast*. De seguida, serão explicadas estas características e exemplos de protocolos que encaixam nestas diferentes categorias.

2.3.1 Domínio da aplicação

O domínio da aplicação [18] é definido por três grandes características - o número de utilizadores que um protocolo suporta, o tipo de informação que o protocolo entrega e as métricas que o protocolo tenta otimizar. Os diferentes tipos de domínios podem ser caracterizados pelos seguintes grupos:

- ***Streaming* de áudio/vídeo:** normalmente envolve uma única fonte que distribui informação para um número grande de recetores. A métrica principal é a largura de banda e a latência. Exemplos de protocolos ALM desta categoria são o Bayeux [19], NICE [20] e OMNI [21].
- **Conferência de áudio/vídeo:** esta categoria envolve grupos de tamanho pequeno a médio que interagem em sessões de conferência múltiplas. A diferença entre esta e a categoria anterior é o tamanho dos grupos *multicast*, sendo estes mais pequenos. O Narada [22] é um exemplo de protocolo ALM deste tipo.
- **Serviço de *multicast* genérico:** os protocolos desta categoria tentam criar um serviço de *multicast* genérico baseado em algumas métricas que afetam várias aplicações. Exemplos: CAN *Multicast* [23], Scribe [24] e YoiD [25].
- **Transferência de ficheiros:** transferência e distribuição fiável de ficheiros, normalmente grandes (exemplo: bases de dados distribuídas e partilha de ficheiros). A única métrica é a largura de banda. O exemplo de protocolo ALM que se encaixa nesta categoria é o OverCast [26].

Como se pode observar, as diferentes classes de aplicações têm diferentes conjuntos de requisitos em relação a fiabilidade, latência, largura de banda e escalabilidade. Estes requisitos por sua vez determinam o mecanismo de gestão dos grupos de *multicast*.

2.3.2 Nível de implementação

Um dos principais fatores que determina as características de um protocolo ALM é o nível em que o protocolo é desenvolvido, se no nível da infraestrutura ou no nível do sistema final [18]. O nível da infraestrutura, também conhecido como *proxy-based*, requer o desenvolvimento de servidores/*proxies* dedicados, onde estes se organizam numa rede de *overlay* e asseguram um serviço *multicast* transparente ao utilizador final. A Figura 2.9 apresenta um exemplo para este tipo de nível de implementação com duas proxies implementadas na rede. O nó 1 envia tráfego para a Proxy 1, que reencaminha para o nó 3 e para a Proxy 2. Por fim, a Proxy 2 reencaminha o tráfego para o nó 2 e nó 4.

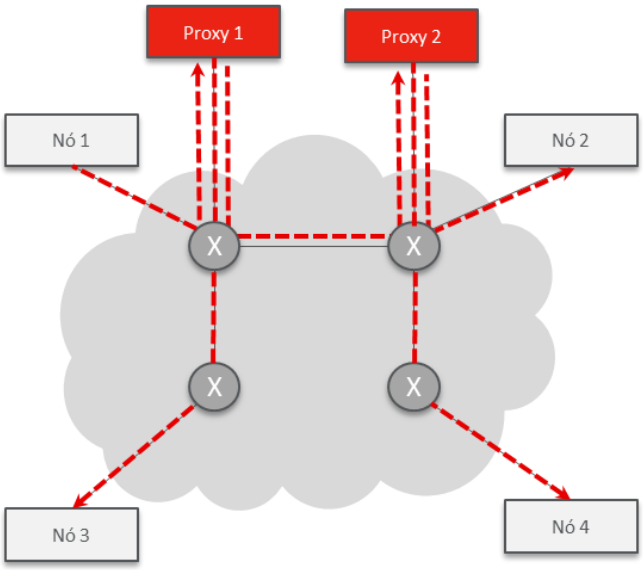


Figura 2.9 - Nível de implementação proxy-based

Por outro lado, os protocolos ao nível do sistema final (*end-system*) assumem somente um serviço *unicast* da infraestrutura e esperam que o sistema final participe ao fornecer a lógica *multicast*, pois este tem responsabilidade no encaminhamento (Figura 2.10).

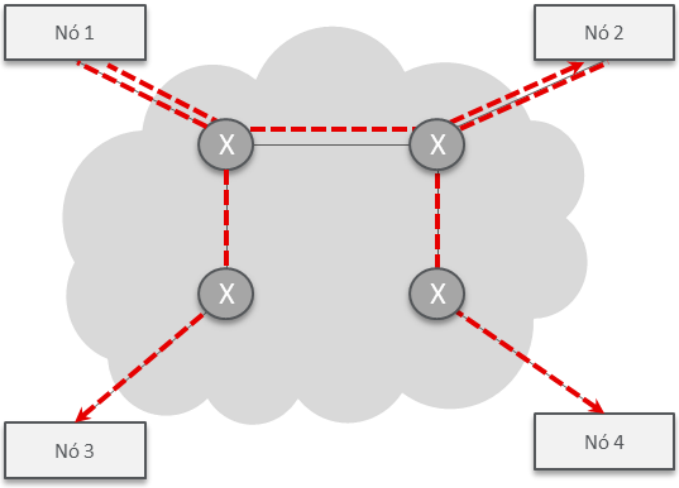


Figura 2.10 - Nível de implementação end-system

Os protocolos ALM *proxy-based* constroem uma rede *overlay*, onde os seus *proxies* são responsáveis pelo encaminhamento do conteúdo, aumentando assim a sua eficácia. Como estes *proxies* fazem parte do core da rede, este tipo de protocolos consegue disponibilizar maior largura de banda aos nós proxy (comparando com a largura de banda disponível no sistema final), consegue fornecer um ciclo de vida mais longo aos nós *overlay* (comparando com a natureza passageira dos sistemas finais), aliviam os sistemas finais de qualquer responsabilidade de encaminhamento e, portanto, reduzem a complexidade da aplicação, pois o *multicast* está disponível de forma transparente para os sistemas finais. A maior desvantagem desta abordagem é a necessidade de desenvolvimento de *proxies* dedicadas, ficando sujeita ao custo de aquisição e manutenção. Este tipo de abordagem também pode ser menos adaptável e otimizada para aplicações específicas, visto que, tipicamente, fornece um serviço de *multicast* genérico em vez de um serviço específico, para uma particular classe de aplicações.

Protocolos ALM ao nível do sistema final desfrutam de mais flexibilidade, adaptabilidade para aplicações específicas e desenvolvimento imediato, mas apresentam limitações de escalabilidade (para um grande número de utilizadores ou sessões simultâneas). Têm que se preocupar com largura de banda limitada e com o facto dos sistemas finais terem a responsabilidade de encaminhamento (e, por isso, aumentam a complexidade do desenvolvimento do software de aplicação e necessidade de processamento dos mesmos).

A Figura 2.11 apresenta os diferentes protocolos que encaixam nas duas abordagens.

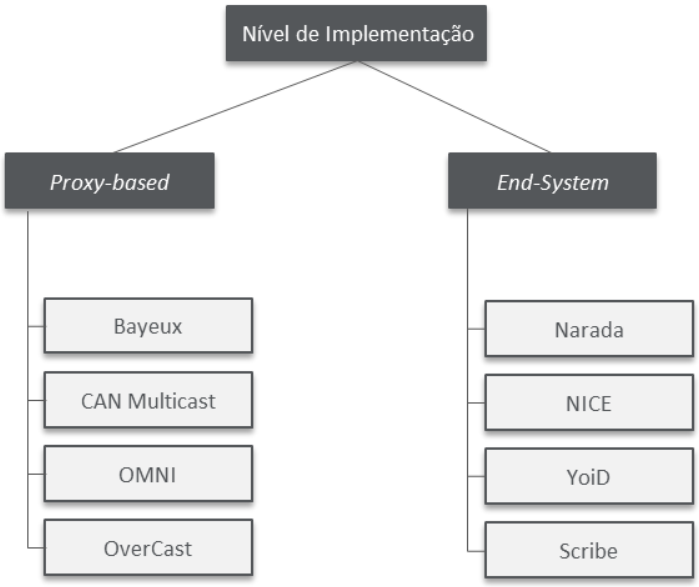


Figura 2.11 - Exemplos de protocolos para cada nível de implementação

2.3.3 Algoritmo de construção de topologias

A questão, de como os membros se agrupam para uma sessão de comunicação multicast, tem grande influência no desempenho do protocolo, ou seja, como é que os membros se organizam na topologia de rede *overlay*. A construção da topologia consiste em duas fases, a fase de agrupamento e a fase de manutenção da topologia. A fase de agrupamento refere-se ao processo em que o novo nó se junta à topologia. A fase de manutenção gere a conetividade da topologia. Esta topologia pode ser construída de forma centralizada ou distribuída [18].

Na abordagem centralizada, a criação e a manutenção da topologia são efetuadas por um nó central controlador. Os membros, que chegam à rede, aprendem sobre a identidade do controlador, através do RP, que tem a função de fornecer, ao novo membro, informação sobre os membros já existentes. O controlador tem conhecimento de todos os membros e as métricas de performance entre os membros, para assegurar uma rede de alta qualidade. A estrutura da rede é depois distribuída aos membros, nomeadamente, a identificação dos vizinhos de cada membro. Os membros, quando recebem esta informação, iniciam as conexões para os vizinhos atribuídos. Quando integrados na rede, os membros entram na fase de manutenção onde monitorizam as conexões para os vizinhos respetivos. Qualquer alteração

no estado dos seus vizinhos, obriga os membros a reportar ao controlador, de forma a reparar/reconstruir a rede.

Na abordagem distribuída, a criação e a manutenção da topologia de rede é feita pelos membros de forma distribuída. Quando um membro, pretende juntar-se à rede, requisita a lista de membros do grupo multicast, junto do RP. Da lista fornecida, o membro seleciona quais os membros a que se quer juntar e envia uma mensagem de *join* para cada um deles. Quando um membro, recebe a mensagem, toma a decisão de admissão do novo membro. Esta decisão é baseada na disponibilidade para uma nova ligação. Quando integrado na rede, o membro entra na fase de manutenção desta. Se um dos seus vizinhos falhar/sair, o próprio nó é que é responsável por alterar a sua situação.

Apesar de, intuitivamente, se pensar que a abordagem de encaminhamento distribuído é a que melhor se encaixa nas aplicações de larga escala, para que estas possam gerir de forma mais eficiente a comunicação dos grupos *multicast*, ainda existem incentivos para a abordagem centralizada. Na abordagem distribuída, a carga de se manter a árvore é distribuída, de forma uniforme, entre os nós da rede. Mas a comunicação síncrona entre os membros para aplicações em tempo real é difícil de assegurar, devido ao atraso na tomada de decisões inerentes. A gestão centralizada dos grupos *multicast* é uma escolha sensata para aplicações de menor escala pois é simples e fácil de desenvolver. Naturalmente, há sempre o risco de um único ponto de falha no sistema centralizado. A escolha entre uma destas abordagens está na decisão entre simplicidade e o aspeto prático versus robustez. Como é percetível, quase todos os protocolos ALM adotam uma abordagem distribuída.

2.3.4 Tipos de topologia

Os protocolos ALM criam duas topologias diferentes [5]: topologia de controlo e topologia de distribuição da informação. A topologia de controlo ajuda a manter e a aperfeiçoar a rede. Os membros da topologia mantêm uma malha (*mesh*) conectada entre eles. A topologia de distribuição da informação é utilizada, como o próprio nome indica, para transmissão de conteúdo multimédia, assumindo uma forma de árvore. Os protocolos ALM diferenciam-se entre si na forma como estabelecem estas topologias. Os protocolos que constroem, em primeiro lugar, a topologia de controlo, utilizam a abordagem *mesh-first*,

enquanto, os protocolos que constroem, em primeiro lugar, a topologia de transmissão de informação, utilizam a abordagem *tree-first*.

Na abordagem *mesh-first*, normalmente, a fonte é escolhida como a raiz e é implementado um protocolo de encaminhamento na malha para que se crie uma árvore. Como este tipo de topologia é criada no início, a sua formação é conhecida por todos os seus membros. Em contrapartida, a topologia de árvore resultante, para a transmissão de informação, não é conhecida. Portanto a qualidade da árvore depende da qualidade da malha escolhida. A figura Figura 2.12 apresenta uma estrutura de topologia em mesh, com uma árvore partilhada (e portanto, bidirecional) incorporada.

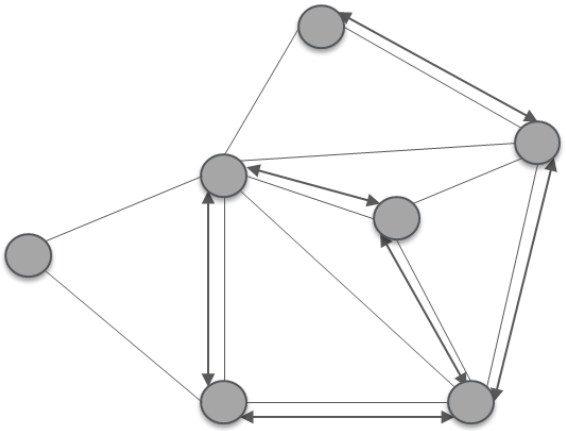


Figura 2.12 - Abordagem mesh-first

Em relação à abordagem *tree-first*, a árvore é construída diretamente sem qualquer malha (Figura 2.13a)). Os membros selecionam explicitamente os seus pais, dos membros conhecidos da árvore. Isto requer um algoritmo que detete e evite *loops* e assegura que a estrutura é de facto uma árvore. Uma das razões para se utilizar a abordagem *tree-first* em vez da *mesh-first* é o facto da primeira fornecer controlo direto sobre a árvore. Este controlo é importante por diferentes razões, nomeadamente, na seleção do melhor vizinho pai que tem mais recursos, ou respondendo aos membros que estão em estado de falha, com o mínimo de impacto para a árvore. Outra vantagem desta abordagem é o facto das ações dos membros da rede serem independentes. Este facto torna o protocolo mais simples, pois a comunicação introduz pouca sobrecarga. No entanto, quando um membro muda de pai, arrasta os seus descendentes consigo (Figura 2.13b)). Isto é desejável, no sentido em que os descendentes não precisam de mudar os seus vizinhos, de facto, eles estão, na realidade, inconscientes da mudança. No entanto, este facto pode resultar em árvores desiguais, o que as torna menos

eficientes do que as árvores corretamente formadas. A vantagem da abordagem *mesh-first* torna-se evidente aqui, pois proporciona mais liberdade para aperfeiçoar a árvore. É possível manipular a topologia da árvore para uma dimensão significativa, ao selecionar os vizinhos da malha e mudar as métricas. Portanto, a abordagem *mesh-first* é mais robusta e compreensiva às partições das árvores e é mais adequada para aplicações com múltiplas fontes, com o custo de maior sobrecarga no controlo.

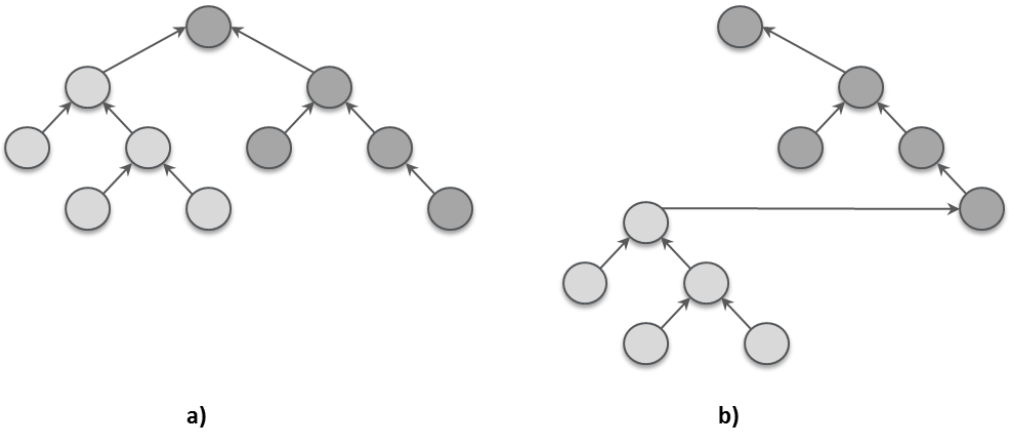


Figura 2.13 - Topologia de árvore a) no estado inicial b) árvore assimétrica

A Figura 2.14 apresenta os diferentes protocolos que encaixam nas duas abordagens.

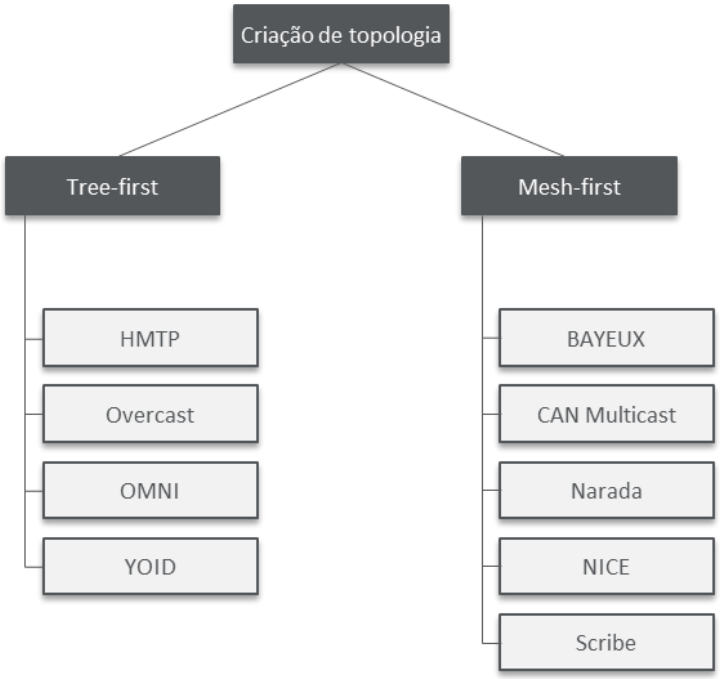


Figura 2.14 - Exemplos de protocolos para cada tipo de criação de topologia

2.3.5 Scribe

O Scribe [24] é uma infraestrutura *multicast* do nível da aplicação para topologias de grande escala. Este protocolo suporta múltiplos grupos, com múltiplos membros e fontes por grupo. É construído em cima do Pastry, um substrato *overlay peer-to-peer* (P2P) de encaminhamento e localização e tira vantagem das propriedades de localização, auto-organização e fiabilidade deste. O Pastry é utilizado para criar e gerir os grupos *multicast* e para construir árvores *multicast* eficientes para a disseminação de informação para cada grupo.

O Scribe é um sistema distribuído de publicação/subscrição que utiliza o Pastry para a gestão de encaminhamento e procura de *hosts*, na sua rede *underlay*.

Pastry

O Pastry [27] é um substrato P2P de encaminhamento que oferece boas propriedades na localização de serviços. Forma uma rede *overlay* robusta e auto-organizável.

Cada nó, na rede Pastry, tem um identificador único de 128 bits (*nodeId*). O *nodeId* é utilizado para indicar a posição do nó num espaço de *nodeIds* circular, que varia entre 0 e $2^{128} - 1$. O *nodeId* é atribuído de forma aleatória, quando o nó se junta ao sistema. É assumido que os *nodeIds* são gerados de tal forma, que o conjunto de *nodeIds* resultante é distribuído, de forma uniforme, no espaço de *nodeIds* de 128bits. Uma forma de gerar estes IDs é através da *hash* criptográfica das chaves públicas dos nós ou dos seus endereços IP.

O Pastry utiliza correspondência de prefixos para encaminhar as mensagens. Assumindo uma rede Pastry com N nós, cada nó Pastry mantém uma tabela de encaminhamento com $[log_2^b N]$ linhas e $2^b - 1$ colunas (b é um parâmetro de configuração com o valor típico 4). As entradas na linha *n* partilham os primeiros *n* dígitos com o presente nó. Cada nó também mantém um conjunto de folhas *l*, que contém os endereços IP dos nós, com os *l/2 nodeIds* numericamente mais perto e os *l/2 nodeIds* numericamente menos perto do *nodeId* do presente nó.

Dada uma mensagem com a sua chave, o nó, em primeiro lugar, verifica o seu conjunto de folhas. Se existir um nó, cujo *nodeId* é mais próximo da chave, a mensagem é encaminhada diretamente para o nó. Se não encontrar nenhuma correspondência no seu conjunto de folhas, o nó verifica a tabela de encaminhamento e a mensagem é encaminhada para um nó que partilha um prefixo comum com a chave em, pelo menos, mais um dígito. Desta forma, com $[\log 2^b N\ ]$ passos, a mensagem consegue atingir o seu nó destino.

A figura mostra um cenário de exemplo. A tabela do lado esquerdo representa a tabela de encaminhamento do nó Pastry com o *nodeId* 147ahf e o diagrama do lado direito representa o encaminhamento de uma mensagem, com a chave d36c4e, do nó 147ahf. O nó 147ahf, através da sua tabela de encaminhamento, encontra o nó d12b23, que partilha um prefixo com um dígito em comum com a chave. Depois o nó d12b23 verifica a sua tabela de encaminhamento e encontra o nó d3241c, que partilha um prefixo com dois dígitos em comum com a chave. Este processo continua até que a chave seja coberta pelo nó d368b3.

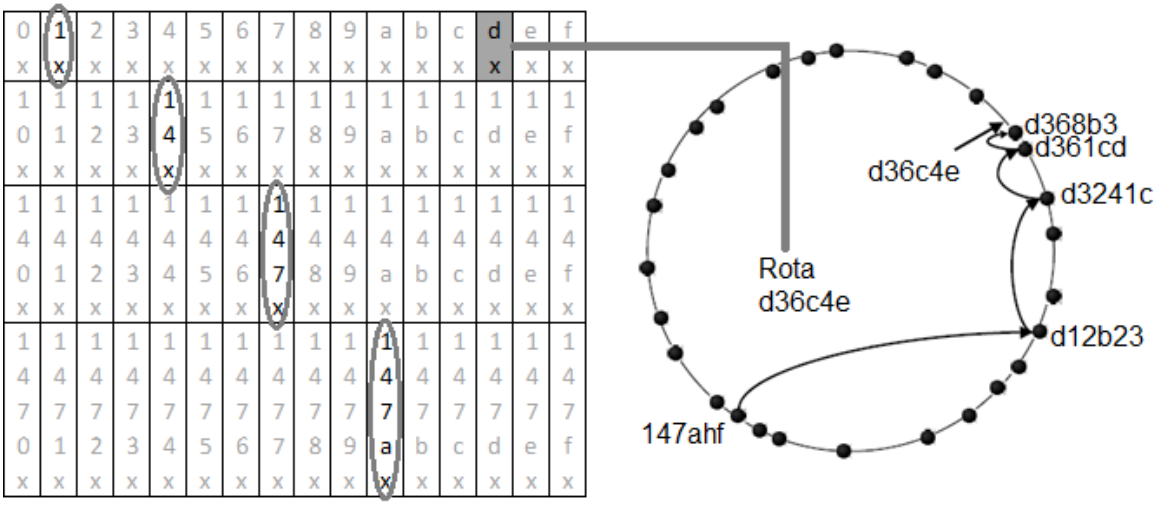


Figura 2.15 - Cenário de exemplo no Pastry [baseado em [24]]

Um novo nó, para se juntar a uma rede Pastry, tem que conhecer um nó já existente da rede. O novo nó inicializa o seu estado ao contactar o nó existente através de uma mensagem de *join*, com o seu *nodeId* como a chave da mensagem. A mensagem é encaminhada para outro nó existente, com o *nodeId* mais perto, numericamente, do *nodeId* do novo nó. Depois, todos os nós, que se encontram no caminho de encaminhamento, enviam as suas tabelas de estado para o novo nó X e este cria as suas próprias tabelas de estado, com base na nova

informação. Finalmente, o novo nó informa qualquer nó que precise de saber da sua existência.

A manutenção das tabelas de encaminhamento é conseguida através de trocas periódicas de mensagens *keep-alive* entre os nós vizinhos. Quando um nó não responde a essas mensagens durante um período de tempo, presume-se que este falhou. Detetada a falha do nó, todos os membros do conjunto de folhas desse nó são informados e estes atualizam os seus conjuntos de folhas.

Scribe – gestão dos grupos

Cada grupo tem um único *groupId*. O nó Scribe, com o *nodeId* numericamente mais perto do *groupId,* age como o RP para o grupo associado. O RP é a raiz da árvore *multicast* criada para o grupo.

Para criar um grupo, o nó Scribe pede ao Pastry para encaminhar uma mensagem *create,* utilizando o groupId como a chave desta. O Pastry entrega esta mensagem ao nó com o nodeId numericamente mais perto do groupId. De seguida, o Scribe adiciona o grupo à lista dos grupos já conhecidos. Este nó Scribe torna-se o RP do grupo. O groupId é a hash do nome textual do grupo, concatenado com o nome do criador do grupo.

Scribe – gestão dos membros dos grupos

O Scribe cria uma árvore *multicast,* cuja raiz é o RP, para disseminar as mensagens *multicast* no grupo. A árvore *multicast* é criada utilizando um mecanismo similar ao *Reverse Path Forwarding*. A árvore é formada ao juntar as rotas Pastry de cada membro do grupo para o RP. As operações de adesão dos membros nos grupos são geridas de uma forma descentralizada para suportar conjuntos de membros grandes e dinâmicos.

Cada membro do grupo mantém uma tabela de filhos para este, que contém uma entrada (endereço IP e nodeId) para cada filho, da árvore *multicast*.

Quando um nó Scribe se quer juntar a um grupo *multicast*, pede ao Pastry para encaminhar uma mensagem *join,* com o groupId como chave. Esta mensagem é encaminhada,

através do Pastry, em direção ao RP. Cada nó, ao longo da rota, verifica se esse nó já faz parte dessa árvore *multicast*, se sim, aceita esse nó como um filho, adicionando-o à sua tabela de filhos. Se não, cria uma entrada para o grupo e adiciona o nó origem como um filho na tabela de filhos associada. Depois, o nó envia a mensagem de *join* para o próximo nó até que atinja o RP (o novo ramo, composto pelo nó que se quer juntar ao grupo e o pai deste, junta-se à árvore *multicast*).

Na Figura 2.16, podemos observar a criação da árvore de um grupo *multicast* no Scribe. O nó 1250 envia uma mensagem *join* em direção ao nó responsável, do grupo *multicast* 7876. O nó 7532 recebe a mensagem, cria uma tabela filhos para esse grupo *multicast* e adiciona uma entrada com a informação do nó 1250. Depois o nó 7532 encaminha a mensagem em direção à raiz da árvore do grupo *multicast*.

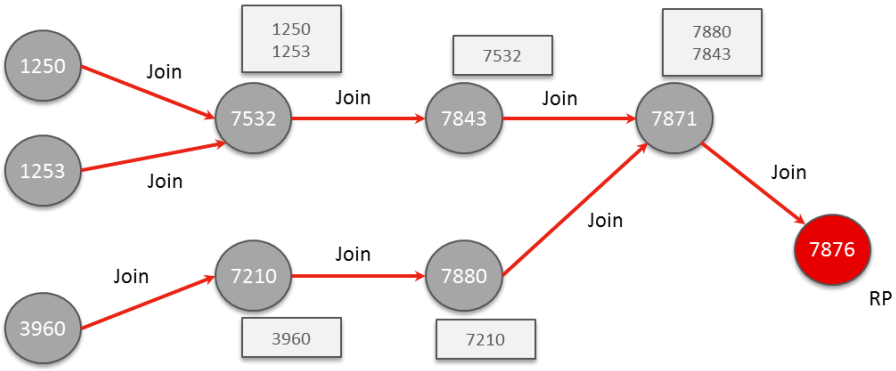


Figura 2.16 - Criação da árvore do grupo *multicast*

Quando um nó *Multicast* deseja sair de um grupo, tem que verificar a sua tabela filhos e se não tiver entradas, envia uma mensagem *leave* para o seu pai na árvore *multicast*. Quando o pai recebe a mensagem, atualiza a tabela filhos, removendo a entrada do nó que quer sair. Se a tabela de encaminhamento estiver agora vazia, envia a mensagem *leave* para o seu pai. Se não estiver, não encaminha essa mensagem.

As propriedades das rotas Pastry garantem que este mecanismo forma uma árvore. Não existem *loops* porque o nodeId do nó seguinte, em cada salto de uma rota Pastry, corresponde a um prefixo mais longo do groupId do que o do nó anterior, ou corresponde a um prefixo com o mesmo comprimento e é numericamente mais perto, ou é o nodeId da raiz.

O mecanismo de gestão dos membros dos grupos é eficiente para grupos com muitos membros. A lista dos membros de um grupo é distribuída pelos nós na árvore *multicast*. As propriedades de aleatoriedade do Pastry asseguram que a árvore é bem equilibrada e a carga do encaminhamento é uniformemente equilibrada nos nós. Este equilíbrio permite ao Scribe suportar números grandes de grupos e membros por grupo. Os pedidos de *join* são tratados localmente de uma forma distribuída. Em particular, o RP não trata de todos os pedidos de *join*.

As propriedades de localização do Pastry asseguram que a árvore *multicast* possa ser utilizada para disseminar as mensagens, de forma eficiente. O atraso para encaminhar a mensagem, desde o RP a cada membro do grupo, é pequeno, devido à propriedade de rotas curtas. A carga imposta na rede física também é pequena porque a maior parte das mensagens são enviadas pelos nós que estão perto das folhas e a distância atravessada na rede, por estas mensagens, é pequena.

Scribe – propagação das mensagens *multicast*

As fontes *multicast* utilizam o Pastry para localizar o RP de um grupo e obterem o endereço IP deste. Depois guardam o endereço, para enviarem mensagens *multicast* para o grupo, sem utilizarem outra vez o Pastry.

As mensagens *multicast* são propagadas desde o RP ao longo da árvore, de forma óbvia. Existe uma única árvore *multicast* para cada grupo e todas as fontes *multicast* utilizam o mecanismo, descrito acima, para enviarem mensagens para o grupo. Isto facto permite ao RP realizar controlo de acesso.

Na Figura 2.17 podemos observar a propagação da informação no Scribe. Qualquer nó pode enviar informação para o RP do grupo *multicast*. Depois o RP começará a propagação da informação *multicast* ao longo da árvore *multicast*.

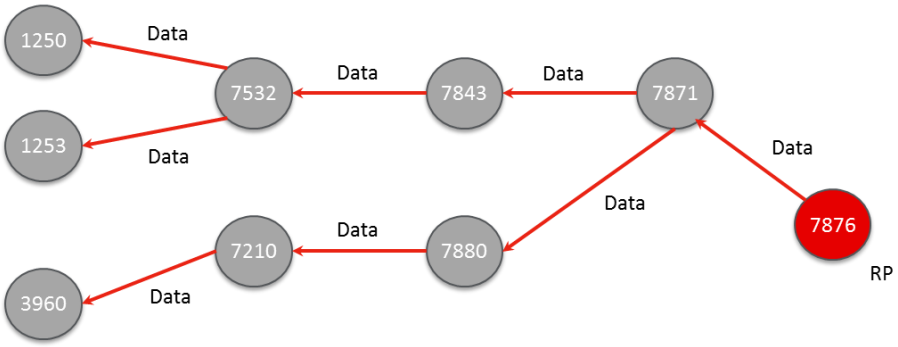


Figura 2.17 - Transmissão da informação

Scribe – fiabilidade

Aplicações que utilizam grupos multicast têm que ter diversos requisitos de fiabilidade. Alguns grupos requerem entrega de mensagens de forma fiável e ordeira, enquanto outros requerem somente entrega *best-effort*. Portanto, o Scribe fornece uma entrega de mensagens *best-effort* mas oferece uma *framework* para as aplicações implementarem fortes garantias de fiabilidade.

O Scribe utiliza o protocolo *Transmission Control Protocol* (TCP) para disseminar as mensagens de forma fiável, dos pais para os seus filhos, na árvore multicast e para controlo de fluxo, e utiliza o Pastry para reparar a árvore multicast, quando um encaminhador falha.

De forma a controlar o estado dos nós na árvore, cada nó não-folha envia, periodicamente, uma mensagem *heartbeat* para os seus filhos. As mensagens multicast servem como um sinal implícito de *heartbeat*, evitando a necessidade de mensagens explícitas deste tipo. Um filho suspeita que o seu pai está em estado de falha, quando não recebe mensagens deste. Quando deteta a falha do seu pai, o nó "chama" o Pastry, para este encaminhar uma mensagem de join para o indentificador do grupo. O Pastry encaminha a mensagem para o novo pai e portanto repara a árvore multicast.

O Scribe também tolera a falha das raízes das árvores multicast (RPs). O estado associado ao RP, que identifica o criador do grupo e tem uma lista de controlo de acesso, é replicado para os nós mais próximos, no espaço do nodeId. Estes nós encontram-se no conjunto de folhas da raiz. Se a raiz falhar, os seus filhos detetam a falha e juntam-se novamente ao grupo, através do Pastry. Este encaminha as mensagens de join para uma nova

raiz (o nó com o nodeId mais próximo numericamente do groupId), que obtém o papel de RP. Os remetentes multicast, da mesma forma, descobrem o novo RP, através de encaminhamento Pastry.

3 CONCEÇÃO DO AMBIENTE DE SIMULAÇÃO

No presente capítulo será apresentada uma breve descrição dos simuladores mais utilizados, atualmente, na área de simulação de redes. Depois, será explicado, com mais detalhe, o simulador escolhido para a comparação entre multicast do nível da camada de rede e o multicast do nível da camada de aplicação, utilizando o protocolo PIM-SM e a aplicação Scribe. Este capítulo também contém uma breve explicação da instalação do simulador.

3.1 Plataformas de Simulação

Na área de investigação de redes é dispendioso implementar um *testbed* físico que contenha computadores, *routers* e ligações de dados para validar um determinado protocolo ou algoritmo de rede. Para tal, os simuladores de rede são uma boa alternativa pois estabelecem um bom compromisso entre o custo operacional e a veracidade dos resultados obtidos. Estes permitem desenvolver, simular, testar e analisar a performance de diferentes protocolos.

Existem dois tipos de simuladores de redes, os comerciais e os *open-source*. Os comerciais normalmente têm um custo associado, não disponibilizando o código fonte do seu *software* aos utilizadores em geral. Todos os utilizadores têm que pagar a licença para o uso do *software* ou para encomendar pacotes em específico. Neste tipo de simuladores, a documentação está completa e atualizada e é mantida por *staff* especializado da empresa. Nos simuladores *open-source*, o código está disponível aos utilizadores, onde qualquer pessoa pode contribuir para o simulador. Este tipo de simulador consegue ser bastante flexível e reflete os desenvolvimentos mais recentes de novas tecnologias, de uma forma mais rápida que os simuladores de redes comerciais.

Atualmente existem inúmeros simuladores de redes com características diferentes mas os mais os mais utilizados, na área de investigação académica de redes de comunicação, são o OPNET, NS-2, NS-3 e OMNeT++.

O OPNET (*Optimized Network Enginnering Tools*) [28] é desenvolvido pela empresa *OPNET Techonologies Inc*.. É um produto comercial, no entanto, tem uma licença gratuita para fins académicos. O ambiente de *software* é especializado para investigação e

desenvolvimento de redes. Devido ao facto de ser um fornecedor de *software* comercial, o OPNET oferece, aos utilizadores, um suporte gráfico poderoso e uma biblioteca de protocolos extensa, baseada na linguagem de programação *C*. Este simulador baseia-se em eventos discretos e utiliza uma estrutura hierárquica para organizar as redes.

O NS-2 (*Network Simulator* 2) [29] é um dos simuladores de redes *open-source* mais populares. É um simulador orientado a objetos, baseado em eventos discretos e foi desenvolvido, originalmente, na Universidade *California-Berkely*. Fornece um suporte substancial para a simulação de protocolos de várias camadas da pilha protocolar, sobre redes com e sem fios. Utiliza a linguagem de programação *C++* para implementar os protocolos e a linguagem *oTCL* para os utilizadores controlarem o cenário de simulação e agendarem os eventos.

O NS-3 [30] é uma evolução natural ao NS-2, sendo também um simulador *open-source*, baseado em eventos discretos. Está licenciado sob a licença *GNU GPLv2* e está disponível para investigação e desenvolvimento. O core do NS-3 é escrito em *C++* e a interface de *script* em *Python*. Este simulador suporta a incorporação de mais *softwares* de redes *open-source* e reduz a necessidade de reescrever modelos para a simulação. Apesar do NS-3 ser desenhado para substituir o NS-2, existem duas principais razões para a sua escassa utilização. O NS-2, como referido anteriormente, possui uma grande popularidade, contendo uma grande base de protocolos desenvolvidos especificamente para o simulador. Como o NS-3 foi desenhado como um novo simulador, não existe compatibilidade com as versões anteriores do NS-2, obrigando a que os protocolos sejam lentamente migrados para a nova arquitetura.

O OMNeT++ (*Objective Modular Network Testbed in C++*) [31], tal como o NS-2 e NS-3, é um simulador *open-source* e é baseado em eventos discretos. Foi desenvolvido por András Varga de *Technical University of Budapest* e em vez de ser construído para ser um simulador especializado, o OMNeT++ foi desenhado para ser o mais genérico possível. O OMNeT++ pode ser descrito, segundo as palavras do seu criador, não como um simulador mas sim uma plataforma de simulação. A arquitetura do simulador é baseada em módulos e são programados em *C++*. Este pode ser encarado como uma base aplicacional, que permite aos investigadores desenvolver diferentes plataformas baseadas nessa mesma base (Inet, Oversim, etc). Estes modelos são desenvolvidos independentemente do OMNeT++ e têm os seus próprios ciclos de lançamento.

Sendo o principal objetivo da dissertação fazer uma comparação entre duas camadas da pilha protocolar diferentes, existiu um grande motivo para a escolha do simulador, este tem que suportar um protocolo da camada de rede e da aplicacional.

O simulador escolhido foi o OMNeT++ pois contém os dois protocolos simulados e na próxima secção é explicado a estrutura do simulador e as *frameworks* inerentes a este.

3.2 OMNeT++

O OMNeT++ [32] foi desenhado para suportar simulação de redes em larga escala. Este objetivo é conseguido através das seguintes características do simulador:

- Para permitir simulação em larga escala, os modelos de simulação são hierárquicos e construídos a partir de componentes reutilizáveis;
- O *software* facilita a visualização e *debugging* dos modelos de simulação, de forma a reduzir o tempo de análise, que, normalmente, ocupa maior parte do tempo dos projetos de simulação;
- O *software* é modular, personalizável e permite embutir as simulações em aplicações maiores;
- Fornece um ambiente de desenvolvimento integrado que facilita o desenvolvimento do modelo de simulação e análise de resultados.

Estrutura do modelo

Um modelo OMNeT++ (Figura 3.1) consiste em módulos que comunicam entre si, através de mensagens. Os módulos ativos são chamados de *simple modules* e são escritos em C++. Estes módulos podem ser agrupados em *compound modules* com diferentes níveis (o número de níveis hierárquicos não tem limite). Os módulos simples possuem interfaces de entrada e de saída (denominada de *gate*), utilizando conexões para comunicarem entre si,

As conexões são criadas dentro de um único nível do modelo hierárquico pois não são permitidas conexões entre diferentes níveis, para não comprometer com a reutilização do modelo. Devido a esta estrutura hierárquica, as mensagens são propagadas através de uma

cadeia de conexões, que começam e acabam nos *simple modules*. Propriedades como atraso de propagação, *data rate* e *bit error date* podem ser atribuídas às conexões.

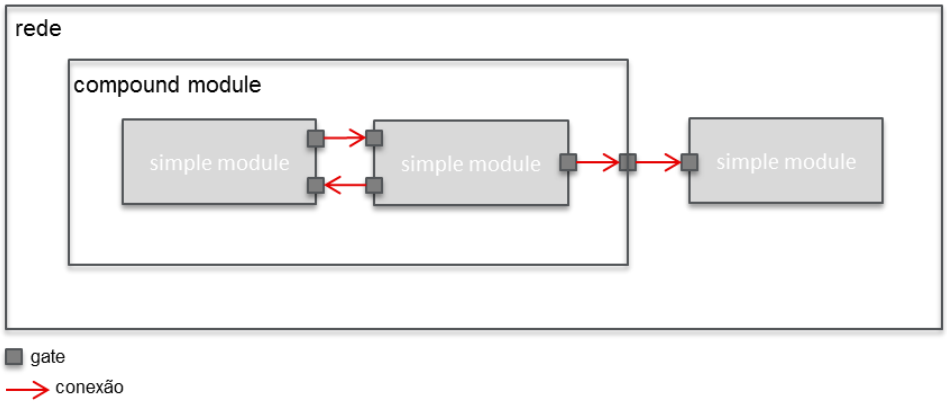


Figura 3.1 - Estrutura de um modelo no OMNeT++

Os módulos normalmente possuem parâmetros associados, onde estes podem apresentar diferentes tipos – *booleanos*, numéricos ou *strings*. Os valores destes parâmetros permitem influenciar o funcionamento das topologias – onde se encontra a fonte, os receptores, entre outros.

Linguagem NED

O utilizador define a estrutura de um modelo através da linguagem de descrição de topologia NED. Um ficheiro NED contém declarações de *simple modules*, definições de *compound modules* e da rede. As declarações do *simple module* descrevem a interface do módulo, gates ou parâmetros. As definições do *compound module* consistem na declaração da interface externa do módulo (gates e parâmetros) e na definição dos submódulos e as suas conexões. Simplificando, um ficheiro NED descreve os seus componentes (*routers* e *hosts*), as suas ligações por interface, a posição onde são apresentados no GUI, o protocolo a ser utilizado.

Ficheiros INI

Num cenário de simulação genérico, normalmente, um utilizador quer saber como é que a simulação se comporta com diferentes parâmetros de entrada. Estas variáveis não se encaixam no código nem nos ficheiros NED, pois podem mudar entre execuções da simulação. Os ficheiros INI permitem definir uma sequência de eventos e os seus valores, influenciando os parâmetros dos ficheiros NED (Figura 3.2). O resultado da simulação pode ser exportado para diferentes formatos (*log* de eventos, escalar ou vectorial), para um posterior processamento por ferramentas de avaliação disponíveis pelo OMNet++ ou por terceiros.

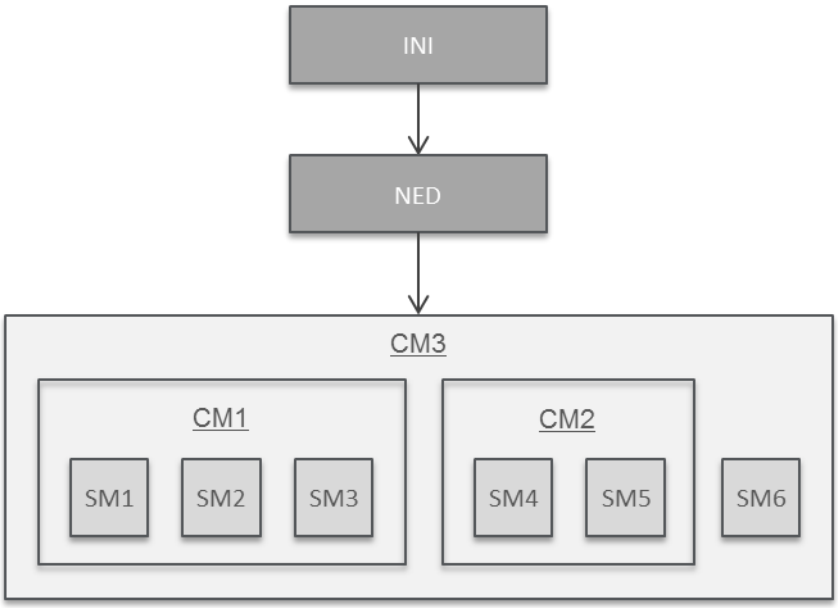


Figura 3.2 - Interação dos diferentes componentes no OMNeT++

Interface gráfica

O OMNeT++ utiliza como interface GUI (*Graphical User Interface*) o *Tkenv*. Nesta interface são implementas as funcionalidades de analise paa o simulador, sendo utilizados para três métodos:

- *Automatic animation* – capacidade de apresentar o fluxo das mensagens nas topologias de rede e as alterações de estado nos nós.
- *Module output windows* – janelas gráficas pré-definidas na simulação (ficheiros INI) para apresentar parâmetros de análise específicos.

- *Object inspectors* – permitem analisar o modelo de simulação com mais detalhe, pois podem ser utilizados para apresentar o estado ou conteúdo de um objeto (através de um histograma), assim como, manualmente, influenciar os seus parâmetros.

3.2.1 INET

O INET [33] é uma *framework* de modelos de simulação de redes desenvolvida sobre o OMNeT++. Disponibiliza um amplo conjunto de protocolos de rede, agentes e outros modelos para os investigadores que trabalham com redes de comunicação.

O INET contém modelos para protocolos da camada de ligação com e sem fios (Ethernet, PPP, IEEE 802.11, etc), para a camada de rede (IPv4, IPv6, OSPF, BGP, etc), para a camada de transporte (TCP, UDP, etc), suporte para mobilidade, protocolos MANET, DiffServ, MPLS, vários modelos de aplicação e muitos outros protocolos e componentes.

O INET é construído sob o conceito de módulos que comunicam através de trocas de mensagem. Os agentes e protocolos de rede são representados por componentes, que podem ser combinados para formarem *hosts*, *routers*, *switches*, e outros dispositivos de rede. Componentes novos podem ser programados pelo utilizador, e componentes já existentes podem ser facilmente compreendidos e modificados.

O INET beneficia da infraestrutura fornecida pelo OMNeT++, utilizando os serviços fornecidos pela biblioteca e *kernel* de simulação do OMNeT++ (Figura 3.3). Os modelos podem ser desenvolvidos, parametrizados, executados, e os seus resultados avaliados através do IDE de simulação do OMNeT++.

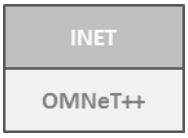


Figura 3.3 - Plataforma INET

O INET beneficia da abordagem modular do OMNeT++, permitindo que outras plataformas o utilizem para desenvolver conteúdos sobre os seus modelos.

3.2.2 Extensão ANSA

Apesar do INET suportar multicast, somente umas funcionalidades básicas se encontram disponíveis para simulação. Diversos protocolos de multicast não estão disponíveis, nomeadamente o PIM-SM, que é um dos protocolos objecto de estudo nesta dissertação. Para colmatar esta lacuna surge o projeto ANSA (*Automated Network Simulation and Analysis*), que, entre outros, adicionou o PIM-SM ao INET (Figura 3.4).

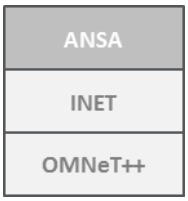


Figura 3.4 - Plataforma ANSA

O projeto ANSA [34] decorre na Universidade de Tecnologia de Brno, na República Checa e é dedicado ao desenvolvimento de várias ferramentas de *software* que conseguem criar modelos de simulação, baseados em redes reais e permitem analisar e verificar configurações da rede em específico. O principal objetivo de investigação deste grupo é suportar transferências de multicast nativo, juntamente com encaminhamento multicast dinâmico, utilizando o PIM. Esta extensão contém vários modelos de simulação novos, entre eles, o modelo ANSARouter é o que implementa os protocolos de gestão de grupos multicast e os protocolos de encaminhamento multicast. Dentro destes protocolos de encaminhamento, a extensão implementa o PIM-SM.

Na Figura 3.5 encontra-se a simulação do protocolo PIM-SM, no simulador OMNeT++. Esta simulação é um exemplo criado pelo projeto ANSA que contém uma fonte de tráfego multicast e dois recetores. Pode-se também verificar na imagem, a interface Tkenv da simulação, onde se encontram todos os *logs* desta.

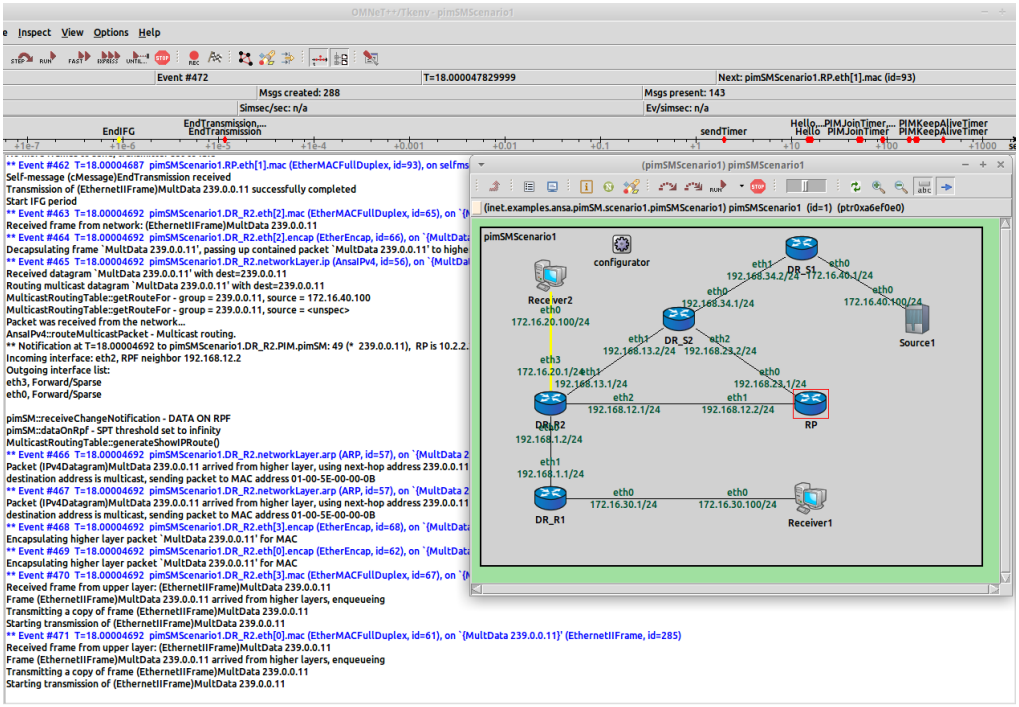


Figura 3.5 - Exemplo de simulação do protocolo PIM-SM

3.2.3 OverSim

O OverSim [35] é uma plataforma de simulação de redes *overlay* flexível, baseado no OMNeT++. É utilizado para modelar protocolos P2P em cima da *framework* INET, numa versão mais antiga (Figura 3.6). Devido à sua arquitetura modular, o OverSim pode ser utilizado numa grande variedade de cenários. Todos os módulos no OverSim podem ser facilmente modificados ou substituídos.

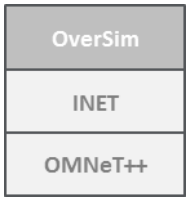


Figura 3.6 - Plataforma OverSim

O Oversim utiliza a interface gráfica do OMNeT++, portanto, consegue apresentar a topologia *overlay* e *underlay*, o interior de todos os nós da simulação e todas as mensagens da rede em detalhe. Esta característica facilita o *debugging* de novos protocolos e permite uma melhor compreensão dos protocolos existentes. Para avaliações, o OverSim contém módulos

que recolhem e processam estatísticas e, juntamente com scripts de processamento dos ficheiros output da simulação, a informação recolhida pode ser facilmente utilizada para a criação de gráficos.

O OverSim foi desenhado como uma *framework* de simulação modular (Figura 3.7). É constituído pelos protocolos da camada de aplicação que comunicam, através de uma API Common, com os protocolos Overlay e estes comunicam com a rede *underlay*, através de uma interface UDP.

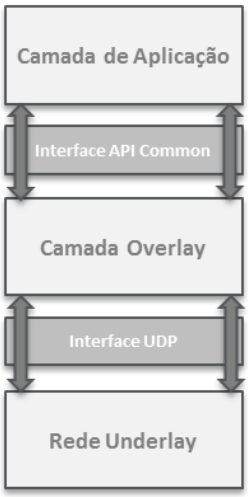


Figura 3.7 - Arquitetura modular do OverSim

O OverSim fornece diferentes modelos para a abstração *underlay*, que diferem em complexidade e exatidão:

- *Simple*: O *Simple Underlay* é o modelo *underlay default* para o OverSim. Neste modelo os pacotes de dados são enviados, diretamente, de um nó overlay para outro, utilizando uma tabela de encaminhamento global. Combina um *overhead* computacional baixo com um nível de exatidão alto, o que o torna um bom modelo de simulação para redes de *overlay* grandes.
- INET: para simulações de redes de acesso heterogéneas, *backbone routers* e mobilidade terminal, o OverSim fornece um modelo *underlay*, baseado na *framework*

INET. Aqui, a *stack* IP é completamente modelada e os *routers* podem fazer parte da rede de simulação overlay.

- *SingleHost*: Este modelo fornece suporte para redes reais no OverSim. Funciona como um *middleware* para suportar a implementação de protocolos *overlay*, desenvolvidos para o OverSim, em redes reais.

Como já foi referido, todos os modelos de rede *underlay* têm uma interface UDP para os protocolos *overlay*. Portanto, o uso de um modelo de rede *underlay* diferente é completamente transparente para a rede *overlay*.

O OverSim fornece várias implementações de protocolos overlay e de aplicações. Para o interesse desta dissertação contém a aplicação Scribe que é construída em cima do protocolo overlay Pastry. A Figura 3.8 contém a simulação do protocolo Scribe, que utiliza o modelo Simple Underlay.

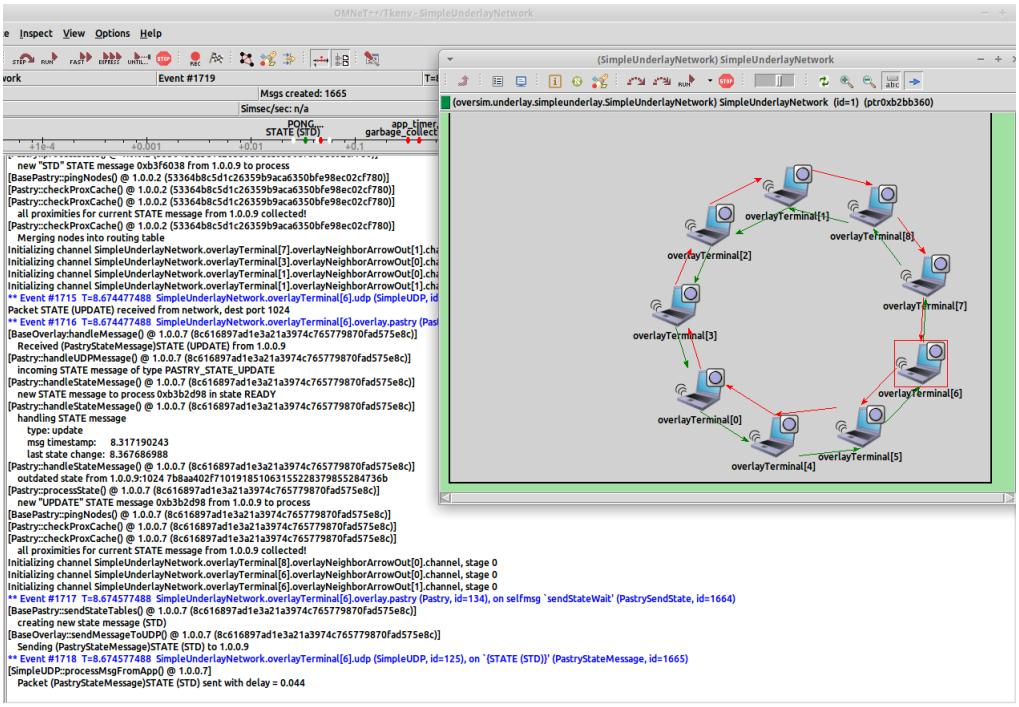


Figura 3.8 - Exemplo de simulação da aplicação Scribe

3.3 Preparação do ambiente de testes

Para estabelecer a plataforma para a simulação do protocolo PIM-SM e da aplicação Scribe foi utilizado o simulador OMNeT++ a correr num terminal com o sistema operativo Linux, com Ubuntu 14.04.

À data de estabelecimento da plataforma de simulação, a versão mais recente do OMNet++ era a 4.6 e do INET era a 2.6.0. No entanto, tanto a extensão ANSA (que suporta o PIM-SM) como o Oversim (que suporta o Scribe) foram desenvolvidos para versões mais antigas. A extensão ANSA utiliza o OMNeT++ 4.4 e o INET 2.2. O OverSim utiliza o OMNeT++ 4.2.2 e o INET 1.99.2.

De forma a ter as duas simulações na mesma versão do OMNeT++, tentou-se instalar a framework OverSim no OMNeT++ 4.4, com o INET 2.2 com a extensão ANSA. Tal não foi possível pois o OverSim foi desenhado com base no INET 1.99.2 e as diferenças desta versão para a versão 2.2 são enormes. Foi equacionado efetuar a alteração do código, mas rapidamente se desistiu da ideia pois desviaria do objetivo da dissertação. Deste modo, foram estabelecidos dois ambientes de simulação distintos, um para o ANSA e outro para o OverSim.

De forma a instalar a versão do OMNeT++ 4.2.2 em versões recentes do sistema operativo Linux, foi necessário modificar alguns ficheiros do OMNeT++. Esta modificação está apresentada no anexo A.

4 SIMULAÇÃO E AVALIAÇÃO DOS PROTOCOLOS

Este capítulo tem como foco a construção dos ambientes de simulação e a análise dos resultados da comparação entre o PIM-SM e o Scribe. De início são explicadas as métricas de avaliação dos dois protocolos, depois é definida uma topologia única para as duas simulações e todos os passos essenciais para as suas criações. Por fim, tendo em conta as métricas de avaliação definidas, são apresentados os resultados da simulação.

4.1 Parâmetros de avaliação de desempenho

Atendendo ao objetivo da dissertação, que se propõe a comparar um protocolo da camada de rede com um protocolo da camada de aplicação para tráfego multicast, as seguintes métricas foram definidas para espelhar as suas diferenças:

- Número de réplicas - número de cópias do pacote multicast original até entregar aos recetores.
- End-to-end delay - tempo que demora desde que a fonte envia o tráfego multicast, até que os recetores o recebam.
- Start-up delay - tempo que demora desde que um recetor faz um pedido *join* a um grupo multicast, até receber o tráfego desse grupo.

4.2 Construção dos cenários de teste

De forma a efetuar uma comparação justa entre o PIM-SM e o Scribe, o primeiro passo foi definir uma arquitetura de simulação comum aos dois protocolos.

A Figura 4.1 representa a topologia desenhada para a simular uma rede de um ISP. É possível identificar uma rede *backbone* constituída pelos *routers* R2, R3, R4, R6 e R7.

A fonte encontra-se ligada a um *router* R1, que por sua vez está ligado ao *backbone*, sendo considerada como parte da rede CORE do ISP. Os *routers* de acesso R5, R8 e R9 estão ligados ao *backbone*, servindo de ligação para os recetores. Todas as ligações foram definidas com 100Mbps de largura de banda, com um atraso de 10ms.

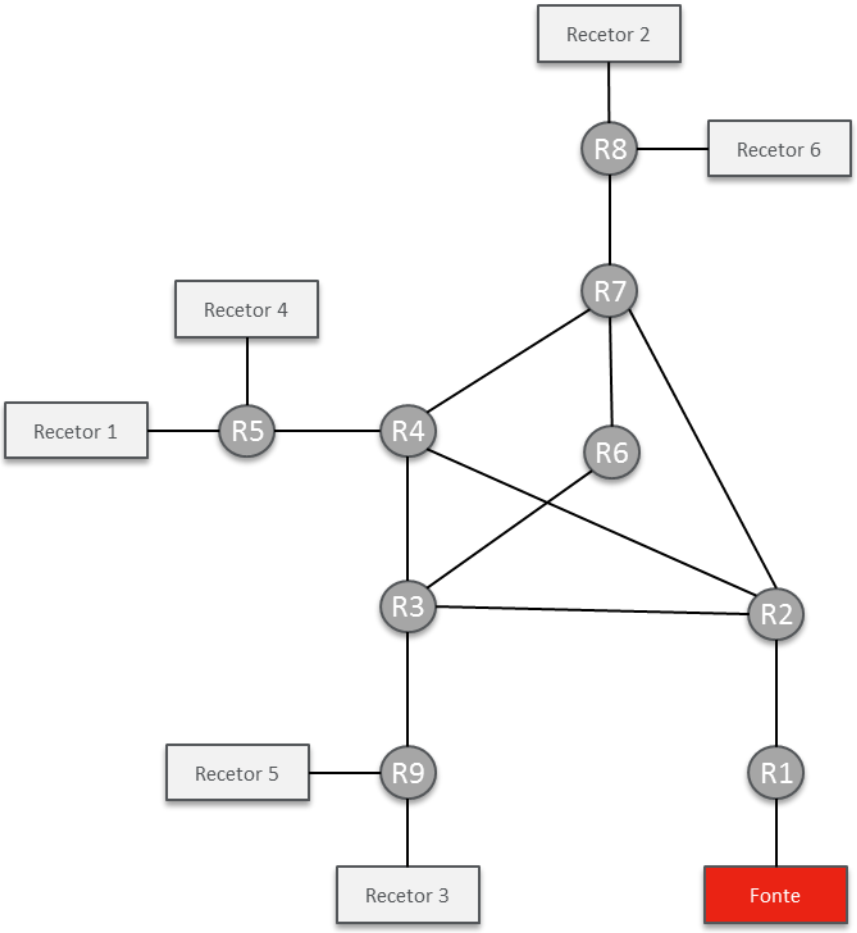


Figura 4.1 - Topologia de simulação

Como explicado no capítulo anterior, o PIM-SM e o Scribe foram desenvolvidos em versões diferentes do OMNeT++/Inet e por equipas diferentes, sendo por isso necessário adaptá-los, de forma a construir a topologia acima descrita.

4.2.1 Construção da topologia do PIM-SM

A implementação disponível para o PIM-SM encontra-se desenvolvida segundo as *guidelines* do OMNeT++, ou seja, a configuração da topologia pode ser facilmente adaptada, alterando simplesmente os ficheiros INI e NED. Isto simplifica a simulação, pois não obriga a conhecer todos os pormenores dos módulos envolvidos neste protocolo. Em particular, para esta simulação, foi necessário alterar os seguintes ficheiros:

- PIM-SM.ned – contém todos os elementos da simulação, os módulos necessários para a simulação, os tipos dos nós da topologia e as suas ligações.
- configPIM.xml – define os mecanismos de encaminhamento dos nós da rede (unicast e multicast).
- PIM-SM.ini – especifica a configuração da rede, os parâmetros a serem apresentados na parte gráfica (nomes, ids, etc) e os eventos da simulação.

PIM-SM.ned

Inicialmente, como se pode verificar na Figura 4.2, são importados todos os módulos necessários à simulação, onde se destacam o PIMRouter (router com a capacidade para processar tráfego multicast), o ANSAStandardHost (sistemas terminais, fonte ou recetores), inet_ethernetline (ligações físicas entre os nós) e o IPv4NetworkConfigurator para a definição de endereços IPv4 e do *routing* estático.

```
import inet.ansa.networklayer.pim.PIMRouter;
import inet.ansa.nodes.inet.AnsaHost;
import inet.ansa.nodes.inet.ANSAStandardHost;
import inet.nodes.ethernet.inet_ethernetline;
import inet.world.scenario.ScenarioManager;
import inet.networklayer.autorouting.ipv4.IPv4NetworkConfigurator;
```

Figura 4.2 - PIM-SM.ned: módulos importados

De seguida, na figura Figura 4.3, são definidos os diferentes nós, sendo descriminado o tipo de nó (PIMRouter ou ANSAStandardHost), a sua localização na topologia e o número de interfaces.

```
        RP: PIMRouter {
            parameters:
                @display("p=808,464");
            gates:
                ethg[4];
        }
        Fonte: ANSAStandardHost {
            parameters:
                @display("i=device/server;p=808,704");
            gates:
                ethg[1];
        }
        Recetor1: ANSAStandardHost {
            parameters:
                @display("p=57,316");
            gates:
                ethg[1];
        }
```

Figura 4.3 - PIM-SM.ned: definição dos nós

Por fim, na Figura 4.4, são definidas todas as ligações bidirecionais entres os nós. Cada ligação identifica os dois nós envolvidos, as interfaces correspondentes de cada nó e o tipo de ligação.

```
    connections allowunconnected:
        Recetor2.ethg[0] <--> inet_ethernetline <--> DR1.ethg[1];
        DR1.ethg[0] <--> inet_ethernetline <--> BR1.ethg[0];
        BR1.ethg[1] <--> inet_ethernetline <--> BR2.ethg[3];
        BR1.ethg[2] <--> inet_ethernetline <--> BR4.ethg[0];
        BR1.ethg[3] <--> inet_ethernetline <--> RP.ethg[1];
        BR2.ethg[0] <--> inet_ethernetline <--> DR2.ethg[1];
        DR2.ethg[0] <--> inet_ethernetline <--> Recetor1.ethg[0];
        BR2.ethg[1] <--> inet_ethernetline <--> BR3.ethg[0];
        BR2.ethg[2] <--> inet_ethernetline <--> RP.ethg[2];
        BR3.ethg[1] <--> inet_ethernetline <--> BR4.ethg[1];
        BR3.ethg[2] <--> inet_ethernetline <--> RP.ethg[3];
        BR3.ethg[3] <--> inet_ethernetline <--> DR3.ethg[1];
        DR3.ethg[0] <--> inet_ethernetline <--> Recetor3.ethg[0];
        RP.ethg[0] <--> inet_ethernetline <--> DR4.ethg[1];
        DR4.ethg[0] <--> inet_ethernetline <--> Fonte.ethg[0];
        Recetor4.ethg[0] <--> inet_ethernetline <--> DR2.ethg[2];
        Recetor5.ethg[0] <--> inet_ethernetline <--> DR3.ethg[2];
        Recetor6.ethg[0] <--> inet_ethernetline <--> DR1.ethg[2];
```

Figura 4.4 - PIM-SM.ned: definição das interligações dos nós

Para obter o tipo de ligação desejada (100Mbps), foi necessário alterar o ficheiro /src/nodes/ethernet/EtherLink.ned e acrescentar esse tipo de ligação. Pode-se verificar a definição da ligação na Figura 4.5.

```
// Ethernet channel for InetUnderlay
channel inet_ethernetline extends ned.DatarateChannel
{
    parameters:
        delay = 10ms;
        datarate = 100Mbps;

}
```

Figura 4.5 - EtherLink.ned: definição do tipo de ligação entre os nós

Para a ligação foi definido o *datarate* e o atraso de propagação.

ConfigPIM.xml

O ficheiro ConfigPIM.xml contém a informação de encaminhamento de cada nó. Existem essencialmente dois tipos de nós, sistemas terminais e *routers* (PIM routers).

O sistema terminal tem uma configuração mais básica, possuindo uma só interface para o *router* de acesso. Através da Figura 4.6, pode-se verificar que, no ficheiro xml, é possível configurar o seu endereço IP e as suas rotas. Nesta simulação optou-se por só configurar *default routing* para os sistemas terminais, apontando este para o seu *gateway*.

```
<!-- Fonte -->
<Host id="10.9.0.2">
      <Interfaces>
            <Interface name="eth0">
                  <IPAddress>10.9.0.2</IPAddress>
                  <Mask>255.255.255.0</Mask>
            </Interface>
      </Interfaces>
      <Routing>
            <Static>
                  <Route>
                        <NetworkAddress>0.0.0.0</NetworkAddress>
                        <NetworkMask>0.0.0.0</NetworkMask>
                        <NextHopAddress>10.9.0.1</NextHopAddress>
                        <ExitInterface>eth0</ExitInterface>
                  </Route>
            </Static>
      </Routing>

</Host>
```

Figura 4.6 - ConfigPIM.xml: configuração da informação de encaminhamento do nó Fonte

Um *router* PIM estende um router tradicional, adicionado a capacidade de compreender protocolos multicast, neste caso o PIM-SM. Além das rotas de encaminhamento

unicast e dos interfaces como nos sistemas terminais, possui um parâmetro que permite ativar o multicast e definir o endereço do RP. Ao nível do interface é necessário ativar o PIM-SM, para poder participar na rede multicast. Estas configurações são apresentadas na Figura 4.7.

```
<!-- RP -->
<Router id="10.6.0.1">
      <Routing>
            <Multicast enable="1">
                  <Pim>
                        <RPAddress>
                              <IPAddress>10.18.0.1</IPAddress>
                        </RPAddress>
                  </Pim>
            </Multicast>
            <Static>
                  <Route>
                        <NetworkAddress>10.2.0.0</NetworkAddress>
                        <NetworkMask>255.255.255.0</NetworkMask>
                        <NextHopAddress>10.13.0.2</NextHopAddress>

                  </Route>
                  (…)

            </Static>
      </Routing>
      <Interfaces>
            <Interface name="eth0">
                  <IPAddress>10.6.0.1</IPAddress>
                  <Mask>255.255.255.0</Mask>
                  <Pim>
                        <Mode>sparse-mode</Mode>
                  </Pim>

            </Interface>

            (…)
            <Interface name="lo0">
                  <IPAddress>10.18.0.1</IPAddress>
                  <Mask>255.255.255.0</Mask>

            </Interface>
      </Interfaces>

</Router>
```

Figura 4.7 - ConfigPIM.xml: configuração da informação de encaminhamento do nó RP

O *router* RP possui uma característica adicional em relação aos demais. É necessário configurar uma interface *loopback* com o endereço do RP, para este identificar o seu papel.

PIM-SM.ini

O primeiro passo na criação da simulação passa por invocar os ficheiros que servem de base para os diferentes eventos. Através do parâmetro do *network*, definimos qual o ficheiro NED que iremos utilizar. Além do tempo da simulação, é ainda visível, na Figura 4.8, no código abaixo, a invocação do ficheiro com as configurações de encaminhamento definidas anteriormente no *configPIM.xml*.

```
description = "PIM-SM"
network = pimSM
tkenv-plugin-path = ../../../../etc/plugins
sim-time-limit = 20s
# devices settings
**.configFile = "configPIM.xml"
```

Figura 4.8 - PIM-SM.ini: configurações iniciais

De seguida, a Figura 4.9 apresenta a definição dos parâmetros dos componentes da rede, como os seus nomes e identificadores.

```
**.DR1.hostname = "DR1"
**.DR2.hostname = "DR2"
**.DR3.hostname = "DR3"
**.DR4.hostname = "DR4"
**.RP.hostname = "RP"
**.BR1.hostname = "BR1"
**.BR2.hostname = "BR2"
**.BR3.hostname = "BR3"
**.BR4.hostname = "BR4"

**.DR1.deviceId = "10.10.0.1"
**.DR2.deviceId = "10.7.0.1"
**.DR3.deviceId = "10.16.0.1"
**.DR4.deviceId = "10.9.0.1"
**.RP.deviceId = "10.6.0.1"
**.BR1.deviceId = "10.2.0.1"
**.BR2.deviceId = "10.3.0.1"
**.BR3.deviceId = "10.5.0.1"
**.BR4.deviceId = "10.4.0.1"
```

Figura 4.9 - PIM-SM.ini: definição dos componentes da rede

Por fim, a Figura 4.10 contém a configuração dos parâmetros da simulação. Estes parâmetros permitem configurar o tempo a que a fonte começa a enviar o tráfego (*start_time_1*), o intervalo em que é enviado (*interval*) e o endereço de destino do tráfego. Na parte dos recetores, pode-se definir o tempo a que eles enviam os pedidos de join

(*start_time_n)* e o endereço multicast do grupo. Estes eventos são possíveis através do módulo *ipTrafGen*.

```
**.Fonte.ipTrafGen.startTime = start_time_1
**.Fonte.ipTrafGen.packetInterval = interval
**.Fonte.ipTrafGen.numPackets = 100
**.Fonte.ipTrafGen.destAddresses = "239.0.0.11"
**.Fonte.ipTrafGen.packetName = "MultData 239.0.0.11"

**.Recetor1.ipTrafGen.startTime = start_time_2
**.Recetor1.ipTrafGen.destAddresses = "239.0.0.11"
**.Recetor1.ipTrafGen.packetName = "IGMP-239.0.0.11"
```

Figura 4.10 - PIM-SM.ini: definição dos parâmetros da simulação

A Figura 4.11 representa graficamente uma topologia com seis recetores e uma fonte, utilizando o ambiente Tkenv do OMNeT++.

De notar que a localização do RP foi definida de uma forma estratégica, escolhendo *router* que permite uma maior eficiência do protocolo. Sendo assim, o RP é um dos *routers* com maior número de ligações para os routers do *backbone* da rede e tem ligação direta com o DR da fonte.

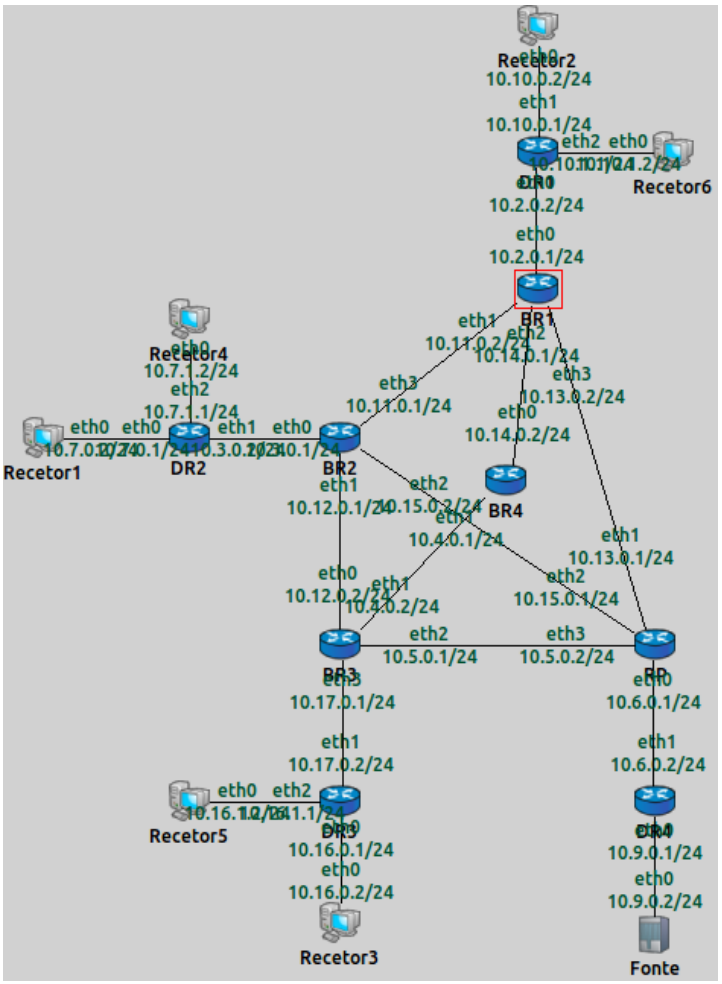


Figura 4.11 - Topologia de simulação do PIM-SM

4.2.2 Construção da topologia do Scribe

O OverSim foi desenhado para permitir simular protocolos da camada da aplicação, normalmente abstraindo (ou simplificando) a camada de rede. Na sua última versão são disponibilizados um conjunto de protocolos *peer-to-peer*, para os mais diferentes fins. Na presente dissertação foi reutilizada a versão disponível do Scribe, introduzindo diversas alterações para permitir a comparação com a versão do PIM-SM.

A primeira alteração surgiu precisamente da já referida simplificação dos nós da camada de rede. A simulação disponível no OverSim utiliza, como base, o módulo *SimpleUnderlay*, que implementa o encaminhamento de forma muito genérica, tendo como principal vantagem não introduzir um *overhead* computacional, o que é muito importante para simulações com muitos nós. A Figura 4.12 é um exemplo de uma simulação baseado neste

modelo *SimpleUnderlay*, ilustrando perfeitamente a abstração da camada de rede, realçando os sistemas finais, que correm o Scribe.

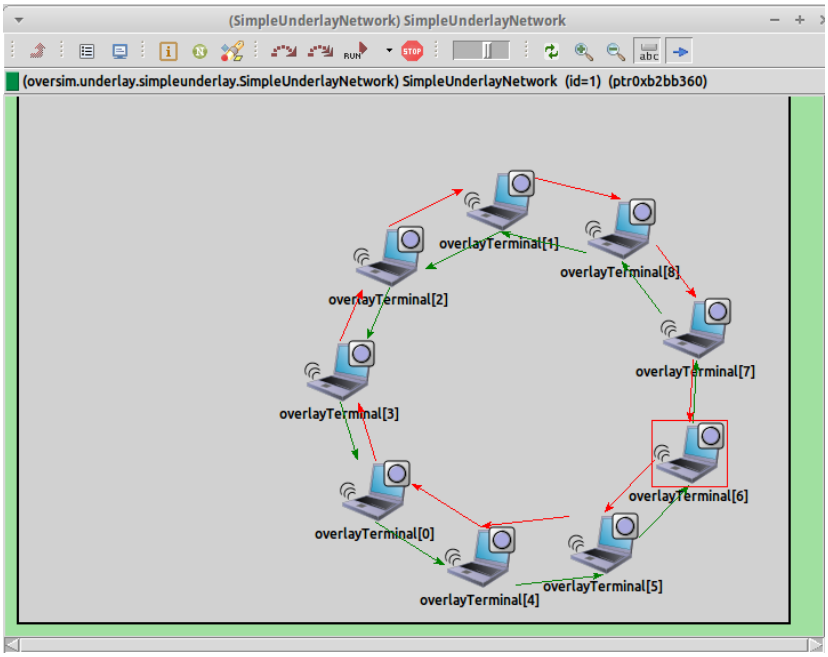


Figura 4.12 - Simulação do Scribe com o módulo *SimpleUnderlay*

InetUderlayNetworkScribe.ned

Para se obter uma comparação, o mais fidedigna possível, foi necessário criar um novo ficheiro, para utilizar o módulo *InetUnderlay*. Este módulo é baseado no *framework* Inet, o mesmo que é utilizado para o PIM-SM. Este módulo contém o *stack* IP completo e ainda permite simular uma rede *Backbone*.

Baseado em modelos disponíveis no OverSim que utilizam o *InetUnderlay*, foi criado um ficheiro *InetUnderlayNetworkScribe.ned* para posteriormente ser invocado pela simulação. Como se pode verificar na Figura 4.13, na simulação é possível influenciar parâmetros como o número de *routers* de *backbone*, o número de routers de acesso, a composição das camadas da aplicação Scribe, entre outros.

```
network InetUnderlayNetworkScribe
{
    parameters:
        string overlayType;    // the overlay used in the simulation (for
overlayBackboneRouters)
        string tier1Type;    // the application on top of the overlay used
in the simulation (for overlayBackboneRouters)
        string tier2Type;    // the module type on top of tier 1 (for
overlayBackboneRouters)
        string tier3Type;    // the module type on top of tier 2 (for
overlayBackboneRouters)
        int backboneRouterNum;    // number of backbone routers in the
network
        int accessRouterNum;    // number of access routers in the network
        int overlayBackboneRouterNum;    // number of backbone routers in
the network that participate the overlay
        int overlayAccessRouterNum;    // number of access routers in the
network that participate the overlay
        int outRouterNum;    // set to 1 if you want to use a TunOutRouter
as connection to a real network
        double connectivity;    // degree of connectivity of backbone
routers (1 = every backbone router is connected to all the others)
```

Figura 4.13 – InetUnderlayNetworkScribe.ned: parâmetros da rede costumizáveis

O ficheiro InetUnderlayNetworkScribe.ned contém ainda as ligações específicas entre os *routers*, bem como o tipo de canal escolhido.

Scribe.ini

Na elaboração da simulação, o primeiro passo é associar o ficheiro *InetunderlayNetworkScribe*, para permitir a posterior criação dos cinco *backbone routers* e dos quatro *access routers*. O número de ligações entre os *backbones* é definido para 0.8, para não existir uma ligação em *full-mesh*, mais uma vez para ficar alinhada com a topologia desejada. A conectividade entre os *access router* e os sistemas terminais são definidos para ligações de 100Mbps. Estas configurações são apresentadas na Figura 4.14.

```
description = Scribe Simulation (INET)
#definition of Underlay Network (customized for Scribe)
network = oversim.underlay.inetunderlay.InetUnderlayNetworkScribe
#Number of backbones routers
InetUnderlayNetworkScribe.backboneRouterNum = 5
#access routers that do not belong to the overlay.
InetUnderlayNetworkScribe.accessRouterNum = 4
#connectivity between Backbone routers
InetUnderlayNetworkScribe*.connectivity = 0.8
#Selecting 100Mbps connection between the terminals and the access routers
InetUnderlayNetworkScribe*.*ccessRouter[*].accessNet.channelTypes =
"oversim.common.inet_ethernetline oversim.common.inet_ethernetline"
InetUnderlayNetworkScribe*.underlayConfigurator.startIPv4 = "10.1.0.1"
InetUnderlayNetworkScribe*.outRouterNum = 0

# Definition of Host type for Scribe
InetUnderlayNetworkScribe.underlayConfigurator.terminalTypes =
"oversim.underlay.inetunderlay.InetOverlayHost"
```

Figura 4.14 – Scribe.ini: configuração da topologia

O Sistema terminal é definido como um *InetOverlayHost*, permitindo fazer parte da rede *overlay* e usufruir do encaminhamento associado ao Inet.

O OverSim utiliza um módulo denominado *Churn Generator* para efetuar a criação de todos os sistemas terminais (*peers*), sendo este responsável por gerir o ciclo de vida destes. Existem basicamente quatro tipos de *Churn Generators*:

- No Churn – é o mais simples, pois não interfere na evolução do *peer* após a sua criação;
- LifetimeChurn – aquando da criação de um nó, é atribuído um tempo de vida (*lifetime)* aleatório e no final desse tempo o nó é removido. Um novo nó será adicionado no seu lugar. Ideal para testar as aplicações *peer-to-peer* tradicionais, onde os nós estão sempre a se conectar /desconectar;
- ParetoChurn – Uma evolução do anterior, especialmente otimizado para simular grandes redes *p2p* de partilha de ficheiros ou jogos online;
- RandomChurn – Como o nome sugere, consoante um valor aleatório, um nó pode ser adicionado, apagado ou migrado.

Na dissertação, como se pode verificar na Figura 4.15, foram criados sete nós do tipo *No Churn*, pois garantem maior controlo sobre o seu período de vida, permitindo comparar com os sistemas finais presentes na simulação do PIM-SM.

```
description = Scribe Simulation (INET)
#definition of Underlay Network (customized for Scribe)
network = oversim.underlay.inetunderlay.InetUnderlayNetworkScribe
#Number of peers (terminals) that you be part of the network - 7
#Selecting a churngenerator of the type NoChurn, the terminal will be
created without lifetime (static).
*.underlayConfigurator.churnGeneratorTypes = "oversim.common.NoChurn
oversim.common.NoChurn oversim.common.NoChurn oversim.common.NoChurn
oversim.common.NoChurn oversim.common.NoChurn oversim.common.NoChurn"
#number of Tiers and the protocols to run on them
**.numTiers = 2
**.overlayType = "oversim.overlay.pastry.PastryModules"
**.tier1Type = "oversim.applications.scribe.MulticastScribe"
**.tier2Type = "oversim.applications.almtest.ALMT
**.tier3Type = "oversim.common.TierDummy"
**.overlay*.pastry.useCommonAPIforward = true
**.overlay*.pastry.enableNewLeafs = true
**.neighborCache.enableNeighborCache = true
*.globalObserver.numGlobalFunctions = 1
*.globalObserver.globalFunctions[0].functionType =
"oversim.applications.almtest.MessageObserverModule"
```

Figura 4.15 – Scribe.ini: configuração do sistema terminal

Uma particularidade desta implementação foi a necessidade de especificar as diferentes *camadas* necessárias por cada sistema terminal. A Figura 4.16 ilustra a representação de um sistema terminal, sendo visível a complexidade associada, pois necessita de um conjunto de aplicações, sobre o encaminhamento do *InetUnderlayHost*.

Figura 4.16 - Sistema terminal Scribe

A aplicação utilizada para os sistemas terminais, quer para a fonte quer para os recetores, foi o *ALMTest*. Esta é disponibilizada no OverSim, sendo esta responsável pelos

pedidos de *join/prune*, envio de pacotes e validação da receção. Trata-se de uma aplicação muito básica que só permite influenciar dois parâmetros:

1- Se o sistema terminal pode fazer *join* a mais que um grupo;
2- Se o sistema terminal pode transmitir pacotes multimédia para o grupo.

Nas diferentes simulações, foi utilizado apenas um único grupo e a fonte para o grupo foi o OverlayTerminal[0]. Estas definições são definidas no código da Figura 4.17.

```
#If joinGroups if true, the terminal will join to more that one group
**[0].tier*.almTest.joinGroups = false
**[1].tier*.almTest.joinGroups = false
**[2].tier*.almTest.joinGroups = false
**[3].tier*.almTest.joinGroups = false
**[4].tier*.almTest.joinGroups = false
**[5].tier*.almTest.joinGroups = false
**[6].tier*.almTest.joinGroups = false
#If sendMessages is true, the terminal will act as a source for the
multicast group
**[0].tier*.almTest.sendMessages = false
**[1].tier*.almTest.sendMessages = true
**[2].tier*.almTest.sendMessages = false
**[3].tier*.almTest.sendMessages = false
**[4].tier*.almTest.sendMessages = false
**[5].tier*.almTest.sendMessages = false
**[6].tier*.almTest.sendMessages = false
```

Figura 4.17 – Scribe.ini: definição das características dos nós

Devido às limitações da aplicação, foi necessário adaptar o código da classe ALMTest.cc, para permitir influenciar o período em que os sistemas terminais se ligam ao grupo e, no caso da fonte, quando começa a emitir.

Quando um sistema terminal é iniciado (criado por um *churn generator*), é invocado o método *handleTimerEvent()*. Este método, apresentado na Figura 4.18, foi totalmente alterado, para deixar de tratar todos os nós do mesmo modo, permitindo especificar parâmetros específicos por nó. Basicamente, cada sistema terminal possui um identificador (*id_peer_n*) ao nível da camada ALMTest, permitindo a distinção dos diferentes *peers*. Quando a função é invocada, o primeiro método a ser validado é a variável global groupNum, que identifica o número de grupo para fazer *join*. Se o valor for igual a zero, significa que ainda não fez *join* a nenhum grupo, procedendo para a próxima validação. Por cada *peer* é definido um tempo para fazer *join* (time_peer_n). Quando o tempo da simulação é igual a esse

valor, é invocada a função *joingroup()*, que irá proceder ao envio da mensagem de join. Se o tempo da simulação não for igual ao desejado, é cancelado o evento atual e é agendado um novo evento (*time_peer_n*).

```
if (groupNum == 0){
            if (getId() == id_peer_1 ) {
                if(simTime().dbl() >= time_peer_1){
                    joinGroup( ++groupNum );
                    if(sendMessages){ // it's the source... let him come
back to start publishing messages...
                        cancelEvent(timer);
                        scheduleAt( simTime() + timer_source, timer );
                    }
                } else {
                    cancelEvent(timer);
                    scheduleAt( time_peer_1, timer );
                }
            }
            (...)
                if (getId() == id_peer_n ) {
                if(simTime().dbl() >= time_peer_n){
                    joinGroup( ++groupNum );
                    if(sendMessages){ // it's the source... let him come
back to start publishing messages...
                        cancelEvent(timer);
                        scheduleAt( simTime() + timer_source, timer );
                    }
                } else {
                    cancelEvent(timer);
                    scheduleAt( time_peer_n, timer );
                }
            }
            (...)
                       } else if ( sendMessages ) {
                    sendDataToGroup( intuniform( 1, groupNum ));
                    scheduleAt( simTime() + timer_source, timer );
             }
```

Figura 4.18 – ALMTest.cc: configuração dos eventos dos nós

No momento do *join*, existem dois casos possíveis – para um recetor ou para uma fonte. No caso do recetor a única necessidade é de garantir que ele faz *join* no momento correto. Para a fonte, além de validar que faz *join* no momento desejado, é necessário criar uma recursividade de eventos, para garantir que após um determinado tempo (*timer_source*), a função é novamente invocada. A distinção do tipo de sistema terminal é efetuada através da variável *sendMessages* (booleano).

Para auxiliar na análise dos resultados obtidos, foram ainda alteradas as funções *handleMCast()*, *sendDataToGroup()* e *joinGroup()*, para imprimirem nos ficheiros de *logs* os seguintes valores:

- Tempo da simulação;
- Identificador do nó;
- Fonte;
- Identificador do pacote multicast.

A Figura 4.19 representa graficamente o resultado de uma simulação para seis recetores, de acordo com a topologia previamente definida e comum aos dois protocolos.

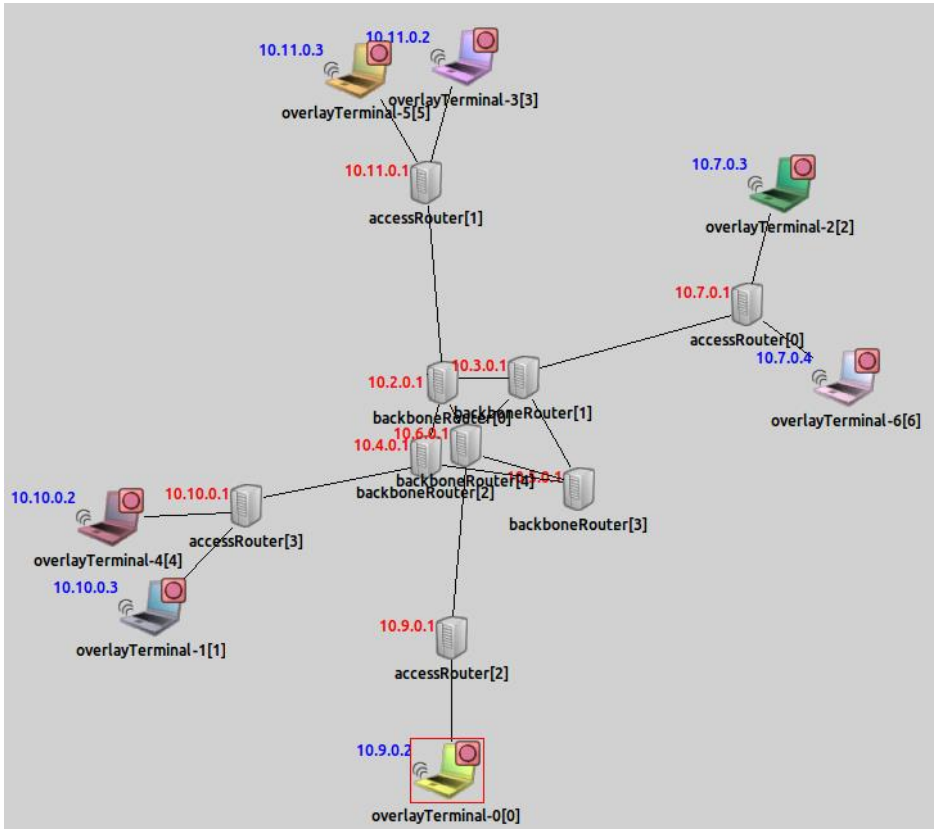


Figura 4.19 - Topologia de simulação do Scribe

4.3 Resultados

Nesta secção são apresentados os resultados da simulação. Estes resultados foram obtidos consoante as métricas de avaliação de desempenho apresentadas.

O OMNeT++ apresenta diversas opções para a análise dos resultados das simulações, sendo os principais formatos o vetorial, escalar ou sobre a forma de *logs* de simulação. Nesta dissertação, para a obtenção dos resultados, foram analisados essencialmente os ficheiros *logs*. Estes ficheiros contêm todos os eventos da simulação e estes são constituídos pelo tempo exato da ocorrência dos eventos, as mensagens trocadas pelos diferentes nós da rede, os endereços de origem e de destino destes, etc. A Figura 4.20 apresenta um exemplo de *logs* de simulação da aplicação Scribe e do protocolo PIM-SM. Estes *logs* representam o envio de tráfego por parte da fonte.

```
** Event #10566  T=3.1  InetUnderlayNetworkScribe.overlayTerminal-
3[3].tier2.almTest (ALMTest, id=327), on selfmsg `app_timer' (cMessage,
id=99) [ALMTest::sendMeeessage()]
** Event #548  T=10.1500584  pimSM.Recetor6.networkLayer.ip (IPv4, id=388),
on `{MultData 239.0.0.11}' (IPv4Datagram, id=393) Received datagram
`MultData 239.0.0.11' with dest=239.0.0.11
```

Figura 4.20 - Exemplo de *logs* de simulação da aplicação Scribe e do protocolo PIM-SM

Além dos *outputs* existentes nas simulações, foram ainda alteradas algumas funções para acrescentar informação que facilita a análise das métricas desejadas.

Sendo o objetivo da dissertação comparar o PIM-SM com o Scribe, foi utilizado uma análise combinatória de todos os casos possíveis para a topologia especificada anteriormente. Foi executado um conjunto de simulações para os diferentes números de recetores (1, 2, 3, 4, 5 e 6), com o intuito de obter a média por número de recetores. A Figura 4.21 apresenta quatro combinações possíveis para o conjunto de simulações com três recetores.

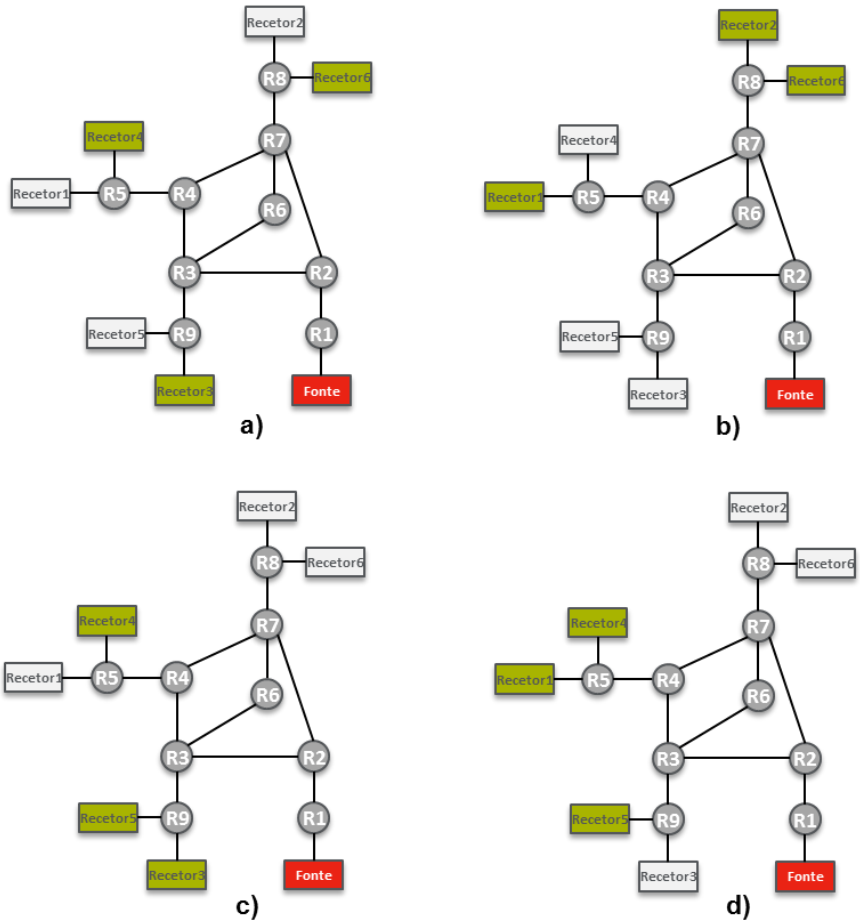


Figura 4.21 - Exemplo de combinações possíveis com três recetores

Em todas as simulações, a fonte inicia o envio de conteúdo antes de existirem recetores no grupo, enviando periodicamente tráfego com intervalo de 0,1 segundos. De seguida, consoante o caso de estudo em questão, o número de recetores desejado efetua *join* ao grupo no mesmo intervalo de tempo.

Através dos eventos disponíveis nos *logs*, os parâmetros de avaliação de desempenho foram obtidos da seguinte forma:

- Número de réplicas – o primeiro passo é identificar o envio do pacote multicast. Nesse evento, existe um identificador do pacote, permitindo nos seguintes eventos identificar o número de réplicas até aos sistemas terminais;
- *Start-up delay* – esta métrica foi calculada com base nos eventos do sistema terminal. Essencialmente é identificado o pedido de *join* ao grupo com o identificador do sistema terminal, sendo de seguida procurado no log a primeira receção de uma

mensagem multicast para esse identificador. O valor final é a diferença do tempo entre a receção do conteúdo multicast e o envio do pedido de *join*;

- *End-to-end delay* – simplesmente é identificado o tempo de envio do pacote multicast pela fonte e validado o tempo que demorou a ser recebido pelos sistemas terminais.

Tendo em conta os parâmetros de avaliação de desempenho apresentados, foram extraídos gráficos com os valores retirados da simulação.

4.3.1 Número de réplicas

O gráfico representado na Figura 4.22 mostra a média de pacotes replicados em relação ao número de recetores. Comparando os dois protocolos, é possível verificar que o PIM-SM apresenta uma taxa de incremento de número de réplicas, por número de recetores, mais baixa que o Scribe, tendendo esse valor para estabilizar. Este resultado é especialmente visível, quando o número de recetores aumenta, pois, no Scribe, o RP é forçado a introduzir um *overhead* no envio dos pacotes para os recetores, onde no PIM-SM essa replicação só é efetuada em caminhos diferentes na árvore.

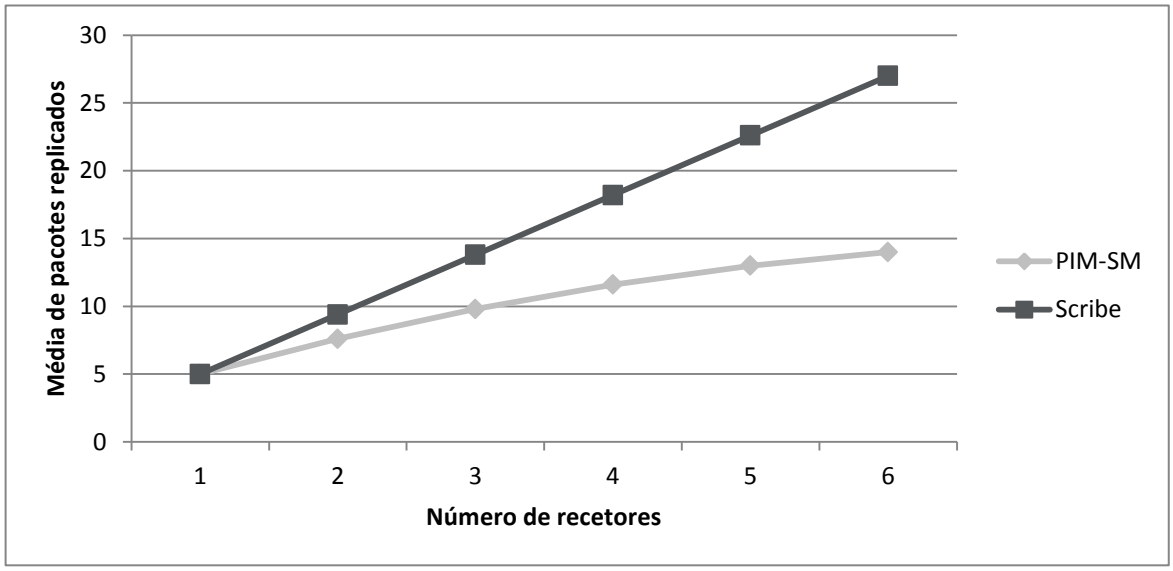


Figura 4.22 - Média de pacotes replicados em função do número de recetores

4.3.2 End-to-end delay

O gráfico da Figura 4.23 ilustra a média do *end-to-end delay* em relação ao número de recetores. Este gráfico espelha claramente a vantagem do PIM-SM, quando o número de recetores aumenta, sendo possível verificar que, no Scribe, a média da diferença do atraso aumenta aproximadamente 50 milissegundos por número de recetor.

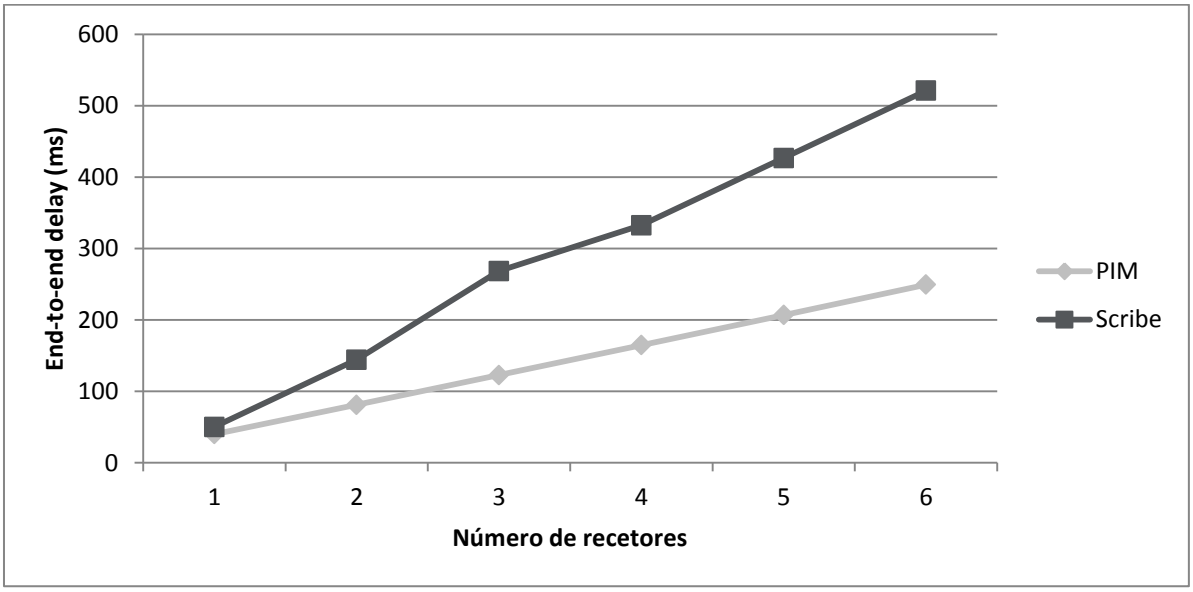


Figura 4.23 - *End-to-end delay* em função do número de recetores

4.3.3 Start-up delay

No gráfico da Figura 4.24, é possível verificar que o tempo que os recetores ficam à espera chega a ser duas vezes superior ao *delay* habitual no Scribe e a três vezes no PIM-SM. Isto é devido ao processo inicial de criação de grupos dos dois protocolos.

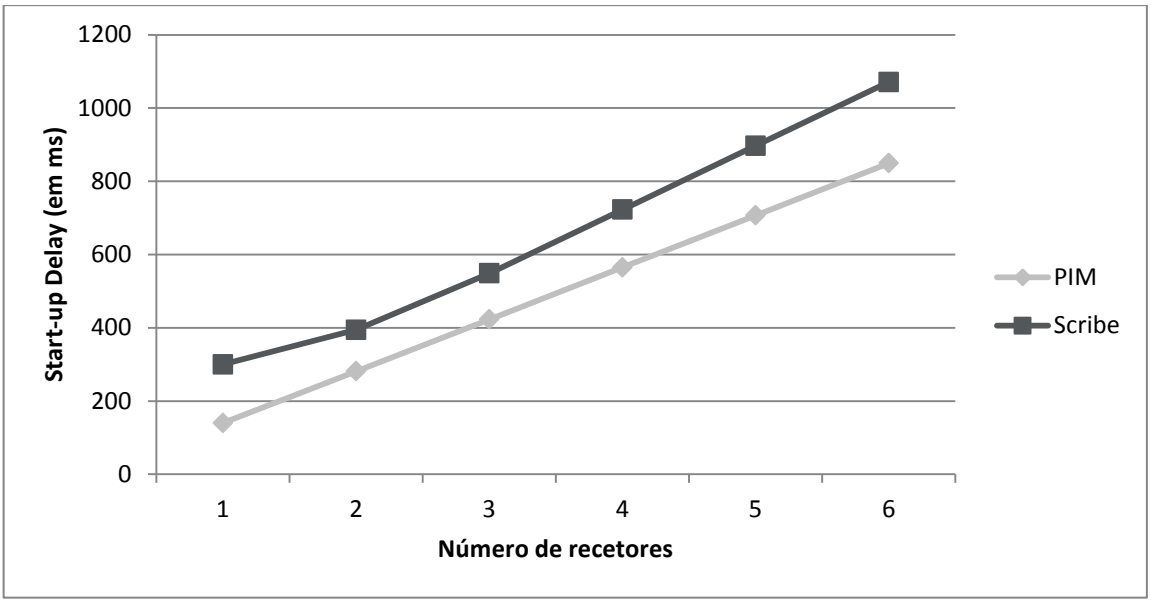


Figura 4.24 - *Start-up delay* em função do número de recetores

5 CONCLUSÃO

A presente dissertação é concluída neste capítulo. Ao longo deste, é realizada uma reflexão sobre os objetivos propostos e expostas as contribuições da dissertação. São ainda apresentadas direções para trabalho futuro, identificando os aspetos que não foram possíveis de alcançar.

5.1 Objetivos alcançados

O trabalho elaborado, ao longo da dissertação, tinha como finalidade estudar, aprofundadamente, o paradigma de comunicação multicast tanto ao nível da camada de rede como ao nível da camada de aplicação, fazendo um levantamento das soluções existentes para o multicast de cada camada. As duas técnicas de multicast são simuladas e avaliadas num cenário de teste, recorrendo ao OMNeT++. Considerando os objetivos inicialmente propostos, conclui-se:

- O paradigma de comunicação multicast para comunicação em grupo é a tecnologia preferida pois o servidor envia um único *stream* de dados para vários utilizadores, evitando a transmissão de pacotes duplicados na rede. Este facto permite reduzir o congestionamento na rede e portanto aumenta o desempenho global desta. O uso eficiente da rede e a redução da carga nas fontes de tráfego permitem disponibilizar serviços a um maior número de clientes.
- Um dos objetivos fundamentais do trabalho passava por realizar um estudo aprofundado das propostas para o encaminhamento multicast, ao nível da camada de rede e da camada de aplicação. Ao nível da camada de aplicação, existem muitas soluções e portanto optou-se por identificar as principais características que as diferem e apresentar, para cada uma, as aplicações ALM correspondentes.
- De forma a fazer uma comparação da performance das duas soluções multicast, escolheu-se um protocolo de cada camada para posterior simulação. A escolha do protocolo PIM-SM, dentro dos protocolos multicast IP, recaiu no facto de ser um protocolo de encaminhamento multicast criado para ser utilizado em redes de grandes dimensões e por isso a escalabilidade é uma das suas principais preocupações. A

escolha da aplicação Scribe deveu-se ao facto de ser uma aplicação descentralizada e facilmente escalável para redes de grandes dimensões, existindo uma implementação do mesmo num simulador.

- Tendo em vista a avaliação de desempenho das duas soluções multicast, para comunicação em grupo em larga escala, utilizou-se o simulador OMNeT++. A escolha do simulador prendeu-se com o facto de ser um simulador completo e conter implementações para as duas soluções multicast, permitindo a simulação e a análise comparativa. As duas soluções estão desenvolvidas para versões diferentes do OMNeT++/INET e por isso foi criada uma topologia de simulação para cada versão. De forma a comparar os dois protocolos, foram definidos parâmetros de avaliação de desempenho, sendo eles, o número de réplicas, *start-up delay* e *end-to-end delay*. Para a simulação foi utilizada uma análise combinatória de todos os casos possíveis para a topologia definida. Foi executado um conjunto de simulações para os diferentes números de recetores, com o intuito de obter a média por número de recetores.

A principal contribuição desta dissertação foi o facto de materializar em valores concretos, as inúmeras comparações teóricas realizadas ao longo dos anos, entre o *multicast* ao nível da rede e aplicacional. Os resultados obtidos apontam o PIM-SM como a solução mais eficiente para comunicação em grupo, explicando o porquê de inúmeros ISPs apostarem neste tipo de implementação.

5.2 Trabalho futuro

Neste tipo de investigação académica, o simulador revela-se a maior dificuldade no estudo mais aprofundando dos protocolos, pois por vezes as melhores implementações de diferentes protocolos não estão na mesma versão do simulador. Este problema é especialmente visível, quando se tratam de protocolos de camadas diferentes.

O trabalho realizado deixa em aberto algumas questões por explorar. Um dos aspetos que precisa de ser modificado é colocar a implementação do PIM-SM e do Scribe na mesma versão do OMNeT++/INET, de forma a só ter uma topologia de simulação. Na fase final da dissertação, surgiu uma nova versão do INET que possui o PIM-SM integrado de raiz, reutilizando o código do projeto ANSA. O passo seguinte será migrar a implementação do

Scribe para esta nova versão, permitindo uma nova comparação. De realçar, que as simulações efetuadas foram novamente executadas na nova versão do INET, não introduzindo diferenças nos resultados obtidos (no anexo B encontra-se o exemplo da configuração utilizada).

De forma a testar a comunicação em grupo, numa maior escala, é necessário acrescentar, na topologia de simulação, mais recetores, com vista a verificar se, quando o número de recetores aumenta drasticamente, o comportamento dos protocolos continua a ser igual. A realização de novos testes para enriquecer a comparação entre os dois protocolos é imperativa, nomeadamente, a definição de novas métricas de avaliação de desempenho.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] A. A. Mahimkar, Z. Ge, A. Shaikh, J. Wang, J. Yates, Y. Zhang, and Q. Zhao, "Towards automated performance diagnosis in a large IPTV network," *Proc. ACM SIGCOMM 2009 Conf. Data Commun. - SIGCOMM '09*, p. 231, 2009.

[2] Statista, "Number of IPTV subscriptions worldwide from 2009 to 2014." [Online]. Available: Http://www.statista.com/statistics/274021/number-of-iptv-subscriptions-worldwide-since-2009/.

[3] A. Maraj and A. Shehu, "The necessity of multicast for IPTV streaming," in *Proceedings of the 10th WSEAS International Conference on TELECOMMUNICATIONS and INFORMATICS*, 2011, pp. 132–136.

[4] L. Lao, J. H. Cui, M. Gerla, and D. Maggiorini, "A comparative study of multicast protocols: Top, bottom, or in the middle?," *Proc. - IEEE INFOCOM*, vol. 4, pp. 2809–2814, 2005.

[5] S. Banerjee and B. Bhattacharjee, "A Comparative Study of Application Layer Multicast Protocols," *Network*, vol. 4, p. 9, 2002.

[6] S. Deering, "Host Extensions for IP Multicasting," *Rfc 1112*, pp. 1–17, 1989.

[7] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," *Rfc 2460*, pp. 1–39, 1998.

[8] M. Cotton, L. Vegoda, and D. Meyer, "IANA Guidelines for IPv4 Multicast Address Assignments," *Rfc 5771*, pp. 1–11, 2010.

[9] H. Holbrook and B. Cain, "Source-Specific Multicast for IP," *Rfc 4607*, pp. 1–19, 2006.

[10] W. Fenner, "Internet Group Management Protocol, Version 2," *Rfc 2236*, pp. 1–24, 1997.

[11] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and a. Thyagarajan, "Internet group management protocol, version 3," *Rfc 3376*, pp. 1–53, 2002.

[12] A. Archer, M. Bateni, M. Hajiaghayi, and H. Karloff, "Improved Approximation Algorithms for Prize-Collecting Steiner Tree and TSP," *SIAM J. Comput.*, vol. 40, no. 2, p. 309, 2011.

[13] D. Waitzman, C. Partridge, and S. Deering, "Distance Vector Multicast Routing Protocol," *Rfc 1075*, pp. 1–24, 1988.

[14] J. Moy, "MOSPF: Analysis and Experience," *Rfc 1585*, pp. 1–13, 1994.

[15] A. Ballardie, "Core Based Trees (CBT) Multicast Routing Architecture," *Rfc 2201*, pp. 1–15, 1997.

[16] A. Adams, J. Nicholas, and W. Siadak, "Protocol Independent Multicast - Dense Mode (PIM-DM) : Protocol Specification (Revised)," *Rfc 3973*, pp. 1–61, 2005.

[17] B. Fenner, M. Handley, H. Holbrook, and I. Kouvelas, "Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised)," *Rfc 4601*, pp. 1–112, 2006.

[18] M. Hosseini, D. T. Ahmed, S. Shirmohammadi, and N. D. Georganas, "A survey of application-layer multicast protocols," *IEEE Communications Surveys and Tutorials*, vol. 9, no. 3. pp. 58–74, 2007.

[19] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz, "Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination," *NOSSDAV 01 Proc. 11th Int. Work. Netw. Oper. Syst. Support Digit. audio video*, no. June, pp. 11–20, 2001.

[20] S. Banerjee and B. Bhattacharjee, "Analysis of the NICE Application Layer Multicast Protocol," 2002.

[21] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller, "OMNI: An efficient overlay multicast infrastructure for real-time applications," *Comput. Networks*, vol. 50, no. 6, pp. 826–841, 2006.

[22] G. S. Rao, E. Jagadeeswararao, and N. S. Prathyusha, “Application Layer Multicasting Overlay Protocol - NARADA Protocol,” vol. 14, no. 6, 2014.

[23] M. Janic, N. Ineke, C. Wangi, X. Zhou, P. Van Mieghem, C. Science, and X. Zhou, “Hopcount in Application Layer Multicast Schemes,” *Topology*, pp. 1–8, 2005.

[24] a Rowstron and a.-M. Kermarrec, “SCRIBE: The design of a large-scale event notification infrastructure,” *Networked Gr. …*, no. November 2001, pp. 30–43, 2001.

[25] P. Francis, “Yoid: Extending the Internet Multicast Architecture,” *Unpubl. Pap. available httpwww aciri orgyoiddocsindex html*, pp. 1–40, 2000.

[26] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O’Toole Jr., “Overcast: reliable multicasting with on overlay network,” *OSDI’00 Proc. 4th Conf. Symp. Oper. Syst. Des. Implement.*, p. 14, 2000.

[27] A. Rowstron and P. Druschel, “Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems,” *Middlew. 2001*, vol. 2218, no. November 2001, pp. 329–350, 2001.

[28] L. OPNET Technologies Co., “OPNET.” [Online]. Available: http://www.opnet.com.

[29] S. Mccanne, S. Floyd, and K. Fall, “NS-2 (Network Simulator 2).” [Online]. Available: http://www.isi.edu/nsnam/ns/.

[30] T. Henderson, S. Floyd, and G. Riley, “NS-3 (Network Simulator 3).” [Online]. Available: http://www.nsnam.org/.

[31] A. Varga, “OMNeT++.” [Online]. Available: http://www.omnetpp.org.

[32] A. Varga and R. Hornig, “An Overview of the OMNeT++ Simulation Environment,” *Proc. 1st Int. Conf. Simul. Tools Tech. Commun. Networks Syst. Work.*, pp. 60:1–60:10, 2008.

[33] Z. Bojthe, L. Meszaros, B. Seregi, R. Horng, and A. Varga, “INET Framework.” [Online]. Available: http://inet.omnetpp.org/.

[34] V. Veselý, O. Ryšavý, and M. Švéda, “Protocol Independent Multicast in OMNeT ++,” no. c, pp. 132–137, 2014.

[35] I. Baumgart, B. Heep, and S. Krause, “OverSim: A Flexible Overlay Network Simulation Framework,” *2007 IEEE Glob. Internet Symp.*, pp. 79–84, 2007.

APÊNDICES

APÊNDICE A. Modificação de ficheiros do OMeT++ 4.2.2 para versões recentes do Linux

As partes que foram acrescentadas aos ficheiros, encontram-se realçadas a vermelho.

omnet-4.2.2/bin/omnetpp

```
#!/bin/sh
#
# Detects the platform and architecture, and starts the IDE with the right
launcher
#
IDEDIR=`dirname $0`/../ide
cd $IDEDIR
PLATFORM=`uname -sm`
LAUNCHER=omnetpp
echo Starting the OMNeT++ IDE...
if java -version 2>&1 | grep -i "libgcj" >/dev/null 2>/dev/null; then
  echo "A compatible JRE is required to run the IDE. "
  echo "Found GNU GIJ which is not supported. Please use Sun JRE or Open
JRE 1.5+"
  echo "If you have several JDKs installed on you machine, change the
default JVM."
  echo "You can switch between Java VMs with the 'sudo update-alternatives
--config java' command."
  exit 1;
fi
if test ! -d configuration; then
  echo "The IDE is not yet configured. Please run the 'configure' script in
the installation root folder!"
  exit 1;
fi
#set default language so GCC will report errors in english. see bug #3
export LANG=en_US.UTF-8

case $PLATFORM in
*MINGW*)
      ./${LAUNCHER}.exe 2>$IDEDIR/error.log &
        ;;
*Linux*x86_64*)
      ./${LAUNCHER} -Dorg.eclipse.swt.browser.DefaultType=mozilla
2>$IDEDIR/error.log &
        ;;
*Linux*)
      ./$LAUNCHER -Dorg.eclipse.swt.browser.DefaultType=mozilla
2>$IDEDIR/error.log &
        ;;
*Darwin*)
      open ./${LAUNCHER}.app 2>$IDEDIR/error.log
        ;;
*)
```

```
      echo OMNeT++ IDE is supported only on: Linux, Windows and MacOS X
Intel
      exit 1
        ;;
esac
```

omnet-4.2.2/src/utils/abspath.cc

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string>
#ifdef _WIN32
#include <direct.h>
#include <stdlib.h>
#endif
#include "../common/ver.h"
std::string toAbsolutePath(const char *pathname)
{
#ifdef _WIN32
    if ((pathname[0] && pathname[1]==':' && (pathname[2]=='/' ||
pathname[2]=='\\')) ||
        ((pathname[0]=='/' || pathname[0]=='\\') && (pathname[1]=='/' ||
pathname[1]=='\\')))
        return std::string(pathname);  // already absolute
    char wd[_MAX_PATH];
    if (pathname[0] && pathname[1]==':') // drive only, must get cwd on
that drive
    {
        if (!_getdcwd(toupper(pathname[0])-'A'+1,wd,_MAX_PATH))
            return std::string(pathname);  // error (no such drive?),
cannot help
        return std::string(wd) + "\\" + (pathname+2);
    }
    if (pathname[0]=='/' || pathname[0]=='\\')
    {
        // directory only, must prepend with current drive
        wd[0] = 'A'+_getdrive()-1;
        wd[1] = ':';
        wd[2] = '\0';
        return std::string(wd) + pathname;
    }
    if (!_getcwd(wd,_MAX_PATH))
        return std::string(pathname);  // error, cannot help
    return std::string(wd) + "\\" + pathname;   //XXX results in double
backslash if wd is the root
#else
    if (pathname[0] == '/')
        return std::string(pathname);  // already absolute
    char wd[1024];
    return std::string(getcwd(wd,1024)) + "/" + pathname; //XXX results in
double slash if wd is the root
#endif
}
int main(int argc, char *argv[])
{
    if (argc==1)
    {
         fprintf(stderr,
```

```
                "abspath -- part of " OMNETPP_PRODUCT ", (C) 2006-2008
OpenSim Ltd.\n"
                "Version: " OMNETPP_VERSION_STR ", build: " OMNETPP_BUILDID
", edition: " OMNETPP_EDITION "\n"
                "\n"
                "Returns the absolute path of the argument."
                 );
        exit(0);
    }
    printf("%s", toAbsolutePath(argv[1]).c_str());
    return 0;
}
```

omnet-4.2.2/src/common/matchexpression.y

```
/* Tokens */
%token STRINGLITERAL
/* Operator precedences (low to high) and associativity */
%left OR_
%left AND_
%left NOT
%pure_parser
%param {void *statePtr}
%start expression
%{
#include <stdio.h>
#include <stdlib.h>
#define YYDEBUG 1           /* allow debugging */
#define YYDEBUGGING_ON 0    /* turn on/off debugging */
#if YYDEBUG != 0
#define YYERROR_VERBOSE     /* more detailed error messages */
#include <string.h>         /* YYVERBOSE needs it */
#endif
void yyerror (void *statePtr, const char *s);
#include "matchexpression.h"
#include "matchexpressionlexer.h"
#include "patternmatcher.h"
#include "exception.h"
#define YYSTYPE  char *
typedef struct _MatchExpressionParserState {
    std::vector<MatchExpression::Elem> *elemsp;
    bool dottedpath;
    bool fullstring;
    bool casesensitive;
    MatchExpressionLexer *lexer;
} MatchExpressionParserState;
#define YYPARSE_PARAM statePtr
#define YYLEX_PARAM statePtr
inline int matchexpressionyylex (YYSTYPE *yylval, void *statePtr)
{
    return ((MatchExpressionParserState*)statePtr)->lexer-
>getNextToken(yylval);
}
using OPP::MatchExpression;
using OPP::PatternMatcher;
using OPP::opp_runtime_error;
%}
```

```
%%
expression
        : expr
        ;
expr
        : fieldpattern
        | '(' expr ')'
        | NOT_ expr
                { MatchExpressionParserState &state =
*(MatchExpressionParserState*)statePtr;
                  state.elemsp-
>push_back(MatchExpression::Elem(MatchExpression::Elem::NOT)); }
        | expr AND_ expr
                { MatchExpressionParserState &state =
*(MatchExpressionParserState*)statePtr;
                  state.elemsp-
>push_back(MatchExpression::Elem(MatchExpression::Elem::AND)); }
        | expr OR_ expr
                { MatchExpressionParserState &state =
*(MatchExpressionParserState*)statePtr;
                  state.elemsp-
>push_back(MatchExpression::Elem(MatchExpression::Elem::OR)); }
        ;
fieldpattern
        : STRINGLITERAL
                {
                    MatchExpressionParserState &state =
*(MatchExpressionParserState*)statePtr;
                    PatternMatcher *p = new PatternMatcher();
                    p->setPattern($1, state.dottedpath, state.fullstring,
state.casesensitive);
                    state.elemsp->push_back(MatchExpression::Elem(p));
                    delete [] $1;
                }
        | STRINGLITERAL '(' STRINGLITERAL ')'
                {
                    MatchExpressionParserState &state =
*(MatchExpressionParserState*)statePtr;
                    PatternMatcher *p = new PatternMatcher();
                    p->setPattern($3, state.dottedpath, state.fullstring,
state.casesensitive);
                    state.elemsp->push_back(MatchExpression::Elem(p, $1));
                    delete [] $1;
                    delete [] $3;
                }
        ;
%%
//----------------------------------------------------------------------
void MatchExpression::parsePattern(std::vector<MatchExpression::Elem>&
elems, const char *pattern,
                                   bool dottedpath, bool fullstring, bool
casesensitive)
{
      MatchExpressionLexer *lexer = new MatchExpressionLexer(pattern);
    // store options
    MatchExpressionParserState state;
    state.elemsp = &elems;
    state.dottedpath = dottedpath;
    state.fullstring = fullstring;
    state.casesensitive = casesensitive;
    state.lexer = lexer;
```

```
    // parse
    yyparse(&state);
}
void yyerror(void *statePtr, const char *s)
{
    // chop newline
    char buf[250];
    strcpy(buf, s);
    if (buf[strlen(buf)-1] == '\n')
        buf[strlen(buf)-1] = '\0';
    throw opp_runtime_error("Error parsing match expression: %s", buf);}
```

APÊNDICE B. Ficheiros para a simulação do PIM-SM no INET 2.99.1

PimSM.ned

```
package inet.examples.pim.sm.basic;
import inet.networklayer.configurator.ipv4.IPv4NetworkConfigurator;
import inet.node.ethernet.Eth100M;
import inet.node.ethernet.inet_ethernetline;
import inet.node.inet.MulticastRouter;
import inet.node.inet.StandardHost;
network pimSM
{
    @display("bgb=1078,987");
    submodules:
        DR1: MulticastRouter {
            parameters:
                @display("p=627,197");
            gates:
                ethg[3];
        }
        DR2: MulticastRouter {
            parameters:
                @display("p=293,472");
            gates:
                ethg[3];
        }
        DR3: MulticastRouter {
            parameters:
                @display("p=437,822");
            gates:
                ethg[3];
        }
        DR4: MulticastRouter {
            parameters:
                @display("p=740,822");
            gates:
                ethg[2];
        }
        BR1: MulticastRouter {
            parameters:
                @display("p=627,329");
            gates:
                ethg[4];
        }
        BR2: MulticastRouter {
            parameters:
                @display("p=437,472");
            gates:
                ethg[4];
        }
        BR3: MulticastRouter {
            parameters:
                @display("p=437,671");
            gates:
                ethg[4];
        }
        BR4: MulticastRouter {
            parameters:
                @display("p=597,513");
```

```
            gates:
                ethg[2];
        }
        RP: MulticastRouter {
            parameters:
                @display("p=740,671");
            gates:
                ethg[4];
        }
        Source1: StandardHost {
            parameters:
                @display("i=device/server;p=740,951");
            gates:
                ethg[1];
        }
        Receiver1: StandardHost {
            parameters:
                @display("p=152,472");
            gates:
                ethg[1];
        }
        Receiver2: StandardHost {
            parameters:
                @display("p=627,76");
            gates:
                ethg[1];
        }
        Receiver3: StandardHost {
            parameters:
                @display("p=437,940");
            gates:
                ethg[1];
        }
        Receiver4: StandardHost {
            parameters:
                @display("p=293,358");
            gates:
                ethg[1];
        }
        Receiver5: StandardHost {
            parameters:
                @display("p=293,822");
            gates:
                ethg[1];
        }
        Receiver6: StandardHost {
            parameters:
                @display("p=758,197");
            gates:
                ethg[1];
        }
        configurator: IPv4NetworkConfigurator {
            @display("p=60,217");
        }
    connections allowunconnected:
        Receiver2.ethg[0] <--> inet_ethernetline <--> DR1.ethg[1];
        DR1.ethg[0] <--> inet_ethernetline <--> BR1.ethg[0];
        BR1.ethg[1] <--> inet_ethernetline <--> BR2.ethg[3];
        BR1.ethg[2] <--> inet_ethernetline <--> BR4.ethg[0];
        BR1.ethg[3] <--> inet_ethernetline <--> RP.ethg[1];
        BR2.ethg[0] <--> inet_ethernetline <--> DR2.ethg[1];
```

```
        DR2.ethg[0] <--> inet_ethernetline <--> Receiver1.ethg[0];
        BR2.ethg[1] <--> inet_ethernetline <--> BR3.ethg[0];
        BR2.ethg[2] <--> inet_ethernetline <--> RP.ethg[2];
        BR3.ethg[1] <--> inet_ethernetline <--> BR4.ethg[1];
        BR3.ethg[2] <--> inet_ethernetline <--> RP.ethg[3];
        BR3.ethg[3] <--> inet_ethernetline <--> DR3.ethg[1];
        DR3.ethg[0] <--> inet_ethernetline <--> Receiver3.ethg[0];
        RP.ethg[0] <--> inet_ethernetline <--> DR4.ethg[1];
        DR4.ethg[0] <--> inet_ethernetline <--> Source1.ethg[0];
        Receiver4.ethg[0] <--> inet_ethernetline <--> DR2.ethg[2];
        Receiver5.ethg[0] <--> inet_ethernetline <--> DR3.ethg[2];
        Receiver6.ethg[0] <--> inet_ethernetline <--> DR1.ethg[2];
}
```

PIMSM.ini

```
[General]
#debug-on-errors = true
network = pimSM
tkenv-plugin-path = ../../../../etc/plugins
cmdenv-express-mode = false
#sim-time-limit = 100s
**.configurator.config = xmldoc("configPIM.xml")
**.pimConfig = xml("<config><interface mode=\"sparse\"/></config>")
**.RP.**.routerId = "10.2.2.2"
**.RP = "10.2.2.2"
**.Receiver?.numUdpApps = 1
**.Receiver?.udpApp[0].typename = "UDPSink"
**.Receiver?.udpApp[0].localPort = 5000
**.Source.numUdpApps = 1
**.Source?.udpApp[0].typename = "UDPBasicApp"
**.Source?.udpApp[0].destPort = 5000
**.Source?.udpApp[0].messageLength = 100B
[Config PIM-SM]
description = "PIM-SM"
**.Source.udpApp[0].destAddresses = "239.0.0.11"
**.Source.udpApp[0].startTime = 9s
**.Source.udpApp[0].sendInterval = 0.1s
**.Receiver?.udpApp[0].multicastGroup = "239.0.0.11"
**.Receiver1.udpApp[0].startTime = 10s
**.Receiver2.udpApp[0].startTime = 10s
**.Receiver3.udpApp[0].startTime = 10s
**.Receiver4.udpApp[0].startTime = 10s
**.Receiver5.udpApp[0].startTime = 10s
**.Receiver6.udpApp[0].startTime = 10s
```

configPIM.xml

```
<?xml version="1.0"?>
<config>
  <interface hosts="Source1" names="eth0" address="10.9.0.2"
netmask="255.255.255.0"/>
  <interface hosts="Receiver1" names="eth0" address="10.7.0.2"
netmask="255.255.255.0"/>
  <interface hosts="Receiver2" names="eth0" address="10.10.0.2"
netmask="255.255.255.0"/>
  <interface hosts="Receiver3" names="eth0" address="10.16.0.2"
netmask="255.255.255.0"/>
```

```
  <interface hosts="Receiver4" names="eth0" address="10.7.1.2"
netmask="255.255.255.0"/>
  <interface hosts="Receiver5" names="eth0" address="10.16.1.2"
netmask="255.255.255.0"/>
  <interface hosts="Receiver6" names="eth0" address="10.10.1.2"
netmask="255.255.255.0"/>
  <interface hosts="DR1" names="eth0" address="10.2.0.2"
netmask="255.255.255.0"/>
  <interface hosts="DR1" names="eth1" address="10.10.0.1"
netmask="255.255.255.0"/>
  <interface hosts="DR1" names="eth2" address="10.10.1.1"
netmask="255.255.255.0"/>
  <interface hosts="DR2" names="eth0" address="10.7.0.1"
netmask="255.255.255.0"/>
  <interface hosts="DR2" names="eth1" address="10.3.0.2"
netmask="255.255.255.0"/>
  <interface hosts="DR2" names="eth2" address="10.7.1.1"
netmask="255.255.255.0"/>
  <interface hosts="DR3" names="eth0" address="10.16.0.1"
netmask="255.255.255.0"/>
  <interface hosts="DR3" names="eth1" address="10.17.0.2"
netmask="255.255.255.0"/>
  <interface hosts="DR3" names="eth2" address="10.16.1.1"
netmask="255.255.255.0"/>
  <interface hosts="DR4" names="eth0" address="10.9.0.1"
netmask="255.255.255.0"/>
  <interface hosts="DR4" names="eth1" address="10.6.0.2"
netmask="255.255.255.0"/>
  <interface hosts="BR1" names="eth0" address="10.2.0.1"
netmask="255.255.255.0"/>
  <interface hosts="BR1" names="eth1" address="10.11.0.2"
netmask="255.255.255.0"/>
  <interface hosts="BR1" names="eth2" address="10.14.0.1"
netmask="255.255.255.0"/>
  <interface hosts="BR1" names="eth3" address="10.13.0.2"
netmask="255.255.255.0"/>
  <interface hosts="BR2" names="eth0" address="10.3.0.1"
netmask="255.255.255.0"/>
  <interface hosts="BR2" names="eth1" address="10.12.0.1"
netmask="255.255.255.0"/>
  <interface hosts="BR2" names="eth2" address="10.15.0.2"
netmask="255.255.255.0"/>
  <interface hosts="BR2" names="eth3" address="10.11.0.1"
netmask="255.255.255.0"/>
  <interface hosts="BR3" names="eth0" address="10.12.0.2"
netmask="255.255.255.0"/>
  <interface hosts="BR3" names="eth1" address="10.4.0.2"
netmask="255.255.255.0"/>
  <interface hosts="BR3" names="eth2" address="10.5.0.1"
netmask="255.255.255.0"/>
  <interface hosts="BR3" names="eth3" address="10.17.0.1"
netmask="255.255.255.0"/>
  <interface hosts="BR4" names="eth0" address="10.14.0.2"
netmask="255.255.255.0"/>
  <interface hosts="BR4" names="eth1" address="10.4.0.1"
netmask="255.255.255.0"/>
  <interface hosts="RP" names="eth0" address="10.6.0.1"
netmask="255.255.255.0"/>
  <interface hosts="RP" names="eth1" address="10.13.0.1"
netmask="255.255.255.0"/>
```

```
  <interface hosts="RP" names="eth2" address="10.15.0.1"
netmask="255.255.255.0"/>
  <interface hosts="RP" names="eth3" address="10.5.0.2"
netmask="255.255.255.0"/>
  <interface hosts="RP" names="lo0" address="10.18.0.1"
netmask="255.255.255.0"/>
  <route hosts="DR1" destination="10.18.0.0" netmask="255.255.255.0"
gateway="BR1"/>
  <route hosts="DR2" destination="10.18.0.0" netmask="255.255.255.0"
gateway="BR2"/>
  <route hosts="DR3" destination="10.18.0.0" netmask="255.255.255.0"
gateway="BR3"/>
  <route hosts="DR4" destination="10.18.0.0" netmask="255.255.255.0"
gateway="RP"/>
  <route hosts="BR1" destination="10.18.0.0" netmask="255.255.255.0"
gateway="RP"/>
  <route hosts="BR2" destination="10.18.0.0" netmask="255.255.255.0"
gateway="RP"/>
  <route hosts="BR3" destination="10.18.0.0" netmask="255.255.255.0"
gateway="RP"/>
  <route hosts="BR4" destination="10.18.0.0" netmask="255.255.255.0"
gateway="BR3"/>
  <route hosts="RP" destination="10.18.0.0" netmask="255.255.255.0"
interface="lo0"/>
</config>
```