UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

INDIQOS: UM SISTEMA PUBLICAÇÃO-SUBSCRIÇÃO COM QUALIDADE DE SERVIÇO

Nuno Miguel Rei Carvalho

MESTRADO EM INFORMÁTICA

Novembro de 2004

INDIQOS: UM SISTEMA PUBLICAÇÃO-SUBSCRIÇÃO COM QUALIDADE DE SERVIÇO

Nuno Miguel Rei Carvalho

Dissertação submetida para obtenção do grau de
MESTRADO EM INFORMÁTICA

pela

Faculdade de Ciências da Universidade de Lisboa

Departamento de Informática

Orientador:

Luís Eduardo Teixeira Rodrigues

Júri:

Edmundo Heitor da Silva Monteiro

Nuno Fuentecilla Maia Ferreira Neves

Ana Paula Pereira Afonso

Novembro de 2004

Resumo

Os sistemas Publicação-Subscrição têm emergido como uma alternativa ao modelo de comunicação Pedido-Resposta. A principal vantagem destes sistemas consiste no desacoplamento entre participantes, que não necessitam de ter conhecimento quer da localização, quer do número de elementos existentes no sistema.

Esta dissertação apresenta um modelo que permite suportar parâmetros de QdS em sistemas de Publicação-Subscrição. O modelo suporta a separação entre a caracterização dos eventos e a caracterização das propriedades de QdS ao mesmo tempo que permite um tratamento uniforme de ambos os aspectos. Descreve-se ainda a arquitectura de um mediador de eventos distribuído com suporte para QdS. Este mediador, denominado *IndiQoS*, utiliza uma rede sobreposta, entre-pares, de pontos de contacto com a capacidade de encaminhar eventos entre editores e assinantes respeitando requisitos de QdS. Para assegurar a QdS entre dos vizinhos da rede sobreposta, são utilizados os mecanismos de reserva de recursos oferecidos pela rede subjacente. Ao evitar a inundação de pedidos de subscrição e de actualizações do estado das ligações, o *IndiQoS* possui capacidade de escala em relação ao tamanho da rede e número de assinantes. Resultados experimentais demonstram a exequibilidade desta aproximação.

PALAVRAS-CHAVE: Sistemas Publicação-Subscrição, Qualidade de Serviço, Tabelas de Dispersão Distribuídas.

Abstract

Publish-Subscribe systems, have been emerging as an alternative to the direct communication model. The main advantage of the indirect communication model are the decoupling properties between applications of this kind of systems. Applications that use this model do not need to know the location nor the number of applications that they communicate with.

This Thesis presents a novel approach to support QoS parameters in publish-subscribe systems. It proposes a model that supports the decoupling of QoS characterization from the event characterization while, at the same time, offers an uniform treatment of both aspects. Furthermore, it describes the architecture of a distributed and scalable publish-subscribe broker with support for QoS. The broker, called *IndiQoS*, leverages on existing mechanisms to reserve resources in the underlying network and on an overlay network of peer-to-peer rendezvous nodes, to automatically select QoS-capable paths. By avoiding flooding of either QoS reservations or link-state information, *IndiQoS* is able to scale with respect to network size and number of reservations. Experimental results show the validity of our approach.

KEY WORDS: Publish-Subscribe systems, Quality of Service, Distributed Hash Tables.

Agradecimentos

Em primeiro lugar, os meus agradecimentos são dirigidos ao meu professor e orientador Luís Eduardo Teixeira Rodrigues. A sua dedicação e profissionalismo foram imprescindíveis para o sucesso deste trabalho. Em segundo lugar, quero dirigir os meus agradecimentos ao Filipe Araújo, pelo seu empenho e pelas discussões que geraram muitas ideias descritas neste trabalho. Quero agradecer também aos restantes elementos do grupo de investigação DialNP, Hugo Miranda e Alexandre Pinto, assim como a alguns dos membros passados deste grupo, João Martins e M. João Monteiro. Foram estas as pessoas com quem sempre gostei muito de trabalhar. Para além das pessoas que já citei, quero também deixar aqui uma palavra de agradecimento ao Luís Sardinha pela sua ajuda quando precisei de alguns recursos de que não dispunha no momento. A sua disponibilidade e prontidão demonstra bem a riqueza do seu caracter.

Le mie prossime parole sono dirette a una persona molto speciale che, nonostante il suo temperamento, ha tuttavia saputo sopportare il mio nei momenti più difficili. Voglio dedicare questo lavoro ad una persona che mi ha fatto crescere molto a livello personale con il suo appoggio e la sua dedicazione. Voglio ringraziare quella italiana che messo sotto sopra la mia vita: *Rosa Del Gaudio*.

Quero agora redireccionar os meus agradecimentos a todos os meus amigos, aqueles que sempre estiveram lá para me apoiar. Penso que não preciso de citar nomes, eles com certeza sabem quem são. Não quero terminar estes agradecimentos sem dedicar algumas palavras à minha família. A compreensão e paciência que sempre tiveram para comigo, assim como o apoio que sempre me deram, não tem qualquer preço.

Finalmente, quero agradecer ao Departamento de Informática e ao LaSIGE pelo acolhimento e condições de trabalho, assim como à Fundação para a Ciência e Tecnologia pelo apoio financeiro através do projecto *IndiQoS* (POSI/CHS/41473/2001).

Lisboa, Novembro de 2004
Nuno Miguel Rei Carvalho

ao joão pedro

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Os modelos de comunicação por eventos e, em particular, o modelo Publicação-Subscrição que segue um paradigma de comunicação indirecta, têm emergido nos últimos anos como uma alternativa ao modelo de comunicação pedido-resposta e, em particular, à invocação remota de procedimentos. Nestes modelos, existem dois tipos de participantes: os *Editores*, que efectuam a publicação de eventos, e os *Assinantes* que efectuam a subscrição de eventos. A principal vantagem do paradigma de comunicação indirecta consiste no desacoplamento entre participantes, que não necessitam de ter conhecimento quer da localização, quer do número de elementos existentes no sistema. Esta propriedade não só permite que certos tipos de reconfigurações às aplicações possam ser feitas "em tempo de execução", i.e., sem que haja paragens, como simplifica a reutilização de componentes de software noutras aplicações, aumentando assim a modularidade do sistema.

Uma das limitações presentes na maior parte das arquitecturas que suportam comunicação indirecta do tipo Publicação-Subscrição consiste na falta de suporte à expressão de parâmetros de Qualidade de Serviço (QdS),

como a largura de banda ou a latência. A falta de suporte à expressão de parâmetros de QdS neste tipo de sistemas é uma desvantagem importante, já que certo tipo de aplicações necessita de garantias de QdS para o seu bom funcionamento. Por outro lado, as soluções que visam oferecer garantias de QdS actualmente em uso ou em estudo não podem ser aplicadas directamente nestas aplicações, por serem essencialmente dirigidas aos sistemas de comunicação directa.

Os sistemas que fornecem garantias de QdS são tipicamente baseados no estabelecimento explícito de canais ou ligações, os quais servem de suporte para efectuar as reservas dos recursos necessários para fornecer a QdS pretendida. Esta abordagem encaixa naturalmente nos sistemas de comunicação directa, onde as ligações entre elementos de um sistema são explicitamente efectuadas, mas é de difícil utilização no modelo Publicação-Subscrição. Neste modelo, as aplicações não devem ser forçadas a estabelecer canais de comunicação e não devem precisar de ter conhecimento acerca da localização e do número de participantes envolvidos na comunicação. Devem apenas preocupar-se com a qualidade da informação que manipulam.

1.1 Sistemas Publicação-Subscrição

Os sistemas Publicação-Subscrição seguem o paradigma da comunicação indirecta. Neste tipo de sistemas existem dois tipos de clientes: *Editores* que publicam informação e *Assinantes* que recebem a informação publicada de que mostraram interesse. Um sistema Publicação-Subscrição é o responsável por receber a informação publicada, fazendo-a chegar até ao(s) assinante(s) interessado(s). É, portanto, um intermediário entre os

vários clientes.

Pela forma como está concebido, um sistema Publicação-Subscrição tem propriedades que podem ser vantajosas em certo tipo de aplicações. Uma das grandes vantagens dos sistemas Publicação-Subscrição consiste no desacoplamento entre os participantes em várias dimensões distintas (Felber, 2001):

Tempo As aplicações não necessitam de participar activamente na interacção ao mesmo tempo;

Espaço As aplicações não necessitam de se conhecer entre elas para comunicar;

Fluxo de dados Os assinantes não necessitam de estar bloqueados à espera das notificações. Estas aplicações podem estar a executar tarefas, sendo notificadas aquando a chegada de uma notificação a elas dirigida.

Recentemente foi ainda introduzida uma nova dimensão no desacoplamento entre participantes: o desacoplamento na QdS. Este desacoplamento separa os parâmetros de QdS do tipo e conteúdo dos eventos disseminados no sistema.

Estas propriedades permitem que certos tipos de reconfigurações às aplicações possam ser feitas "em tempo de execução", i.e., sem que haja paragens no sistema. A construção de um sistema de comunicação indirecta simplifica a reutilização de componentes de software noutras aplicações, aumentando assim a modularidade do sistema.

Uma das limitações presentes na maior parte das arquitecturas apresentadas anteriormente consiste na falta de suporte à expressão de parâmetros de QdS, como a largura de banda ou a latência na comunica-

ção. Esta limitação aplica-se a vários modelos, como o CORBA Event Service (OMG, 2001), CORBA Notification Service (OMG, 2002), Java Message Service (Hapner *et al.*, 2002), assim como a sistemas como o Cambridge Event Architecture (CEA) (Bacon *et al.*, 2000), Scalable Internet Event Notification Architecture (SIENA) (Carzaniga, 1998) ou o Hermes (Pietzuch & Bacon, 2002).

Surge, então, a necessidade de construir um sistema deste género, enriquecido com funcionalidades que garantam QdS às aplicações.

1.2 Integração de QdS em sistemas de comunicação indirecta

Os mecanismos existentes que servem de suporte aos sistemas distribuídos fornecendo garantias de QdS podem ser de dois tipos, que serão indicados de seguida.

Uma forma de garantir QdS na comunicação entre aplicações consiste no uso de sistemas como os *Serviços Integrados* (Braden *et al.*, 1994; Braden *et al.*, 1997; Wroclawski, 1997). Este tipo de sistemas são tipicamente baseados no estabelecimento explícito de canais ou ligações, canais estes onde são reservados os recursos necessários para fornecer a QdS pretendida.

Outro tipo de sistemas que garantem QdS consiste nos *Serviços Diferenciados* (Blake *et al.*, 1998). Neste tipo de sistemas, os fluxos de dados são diferenciados e é atribuída uma QdS diferente a cada um deles e foi construído com o intuito de reduzir quer o processamento, quer o estado que os encaminhadores intermédios necessitam de manter para oferecer as garantias de QdS necessárias. Esta abordagem é mais leve em termos de processamento de mensagens e quantidade de informação a manter,

mas é também menos poderosa no que toca à expressão de requisitos de QdS por parte dos intervenientes do sistema.

Qualquer uma das abordagens encaixam de forma natural nos sistemas de comunicação directa, onde as ligações entre elementos de um sistema são explicitamente efectuadas ou os caminhos dos fluxos de dados são bem conhecidos, mas é de difícil utilização no modelo Publicação-Subscrição.

1.3 Objectivos

O principal objectivo do trabalho aqui apresentado é conceber uma nova arquitectura de um sistema de comunicação indirecta, seguindo o paradigma Publicação-Subscrição, que fornece garantias de QdS às aplicações – a arquitectura *IndiQoS*. Esta arquitectura permite (*i*) fornecer às aplicações uma forma de exprimir os parâmetros de QdS pretendidos e (*ii*) remover da aplicação a tarefa de reservar recursos.

As reservas a efectuar necessitam de ser baseadas em informação dinâmica, como a localização, número e características dos participantes, assim como das características da informação que está a circular no sistema. Toda a informação necessária para manter o sistema deve ser processada de forma automática pelo próprio sistema de Publicação-Subscrição, delegando às aplicações apenas a tarefa de processar os eventos disseminados.

Finalmente, deve ser efectuada uma avaliação da arquitectura proposta, confrontando as opções tomadas com outras aproximações.

1.4 Resultados

O trabalho descrito nesta dissertação atingiu vários resultados, mais precisamente:

- O estudo efectuado permitiu a construção de uma nova arquitectura que segue o paradigma Publicação-Subscrição, garantindo QdS às aplicações;
- A arquitectura foi concretizada de forma a funcionar como um sistema distribuído (sobre uma rede de dados), sendo adaptável de forma a funcionar também sobre um simulador de rede para validação de resultados;
- A arquitectura proposta foi avaliada e comparada com outras aproximações, de forma a validar os resultados obtidos.

1.5 Estrutura da tese

Esta dissertação é composta por 5 capítulos, como descrito de seguida. O Capítulo 2 consiste num sumário dos vários componentes usados para conceber a arquitectura pretendida: (*i*) sistemas Publicação-Subscrição, (*ii*) mecanismos que fornecem garantias de QdS, (*iii*) características de redes *overlay* e, finalmente, (*iv*) formas abstractas de modelação de tráfego. O Capítulo 3 descreve a arquitectura *IndiQoS*, as opções tomadas na construção dos vários componentes e a interacção entre eles. O Capítulo 4 avalia o seu desempenho e, finalmente, o Capítulo 5 conclui esta dissertação e aponta direcções futuras.

Capítulo 2

QdS em sistemas Publicação-Subscrição

Os sistemas Publicação-Subscrição dispõem de algumas propriedades que os tornam atractivos em certo tipo de aplicações, como por exemplo aplicações que necessitam de efectuar difusão em grupos com elevado número de elementos. Estas propriedades não se encontram nos modelos de comunicação directa dos quais os sistemas baseados em Chamadas a Procedimentos Remotos (RPC) (do inglês *Remote Procedure Calls*) são representativos. Num sistema onde as aplicações necessitam de efectuar difusão de mensagens por todo um grupo com elevado número de elementos, se for usado RPC como ferramenta para a construção da comunicação, as aplicações terão de manter informação sobre os restantes elementos do grupo. Pelo contrário, se for usado um sistema Publicação-Subscrição na comunicação, essa informação não necessita de ser mantida pelas aplicações, libertanto-as do processamento de filiação e poupando recursos de memória. Infelizmente, a maioria das arquitecturas existentes de Publicação-Subscrição não consideram a especificação de parâmetros de Quali-

dade de Serviço (QdS) por parte das aplicações, ou essa especificação é de capacidade restrita (e.g. fiabilidade).

Neste capítulo irão ser abordados quer os sistemas Publicação-Subscrição, quer os sistemas com suporte para QdS, referindo as vantagens e desvantagens destes sistemas. Serão também dadas as directrizes para enriquecer uma arquitectura Publicação-Subscrição que fornece às aplicações a possibilidade de expressar requisitos de QdS na comunicação.

2.1 Sistemas Publicação-Subscrição

Nesta secção será descrito o modelo Publicação-Subscrição, que segue o paradigma de comunicação indirecta, dando ênfase ás suas propriedades e variantes e confrontando-o com paradigmas de comunicação alternativos.

2.1.1 Paradigmas de comunicação

Protocolos de Transporte Esta abstracção é uma forma de comunicação de baixo nível. Usando este tipo de mecanismos, o endereço dos intervenientes tem de ser conhecido. Pormenores como a representação uniforme dos dados ficam da responsabilidade da aplicação. Tipicamente, neste tipo de comunicação os intervenientes têm de estar activos em simultâneo no momento da comunicação, o que significa que o receptor terá de estar à espera dos dados no momento em que o emissor os envia.

Comunicação em grupo No contexto de comunicação, um grupo consiste num conjunto de processos que comunicam entre si para atingir um objectivo (Chockler *et al.*, 2001). Esse grupo partilha um endereço ou nome, que é usado quando se pretende enviar uma mensagem para todo o grupo.

Esta noção pode ser concretizada, por exemplo, usando os mecanismos de difusão fornecidos pelo IP em redes locais. Quando se pretende coordenar a actividade do grupo, cada elemento tem de manter informação sobre os restantes elementos. Para manter esta informação, é necessário ter um serviço de filiação que fornece informação actualizada sobre os processos que estão activos e que estão inactivos num determinado momento. Quando um elemento entra ou sai do grupo, todos os elementos activos recebem uma nova *vista*. Uma vista de grupo consiste numa lista dos elementos que estão activos num determinado momento.

Chamadas a Procedimentos Remotos Uma das mais conhecidas formas de efectuar a comunicação em sistemas distribuídos, consiste na invocação remota de procedimentos (RPC) (Birrell & Nelson, 1984). Com este tipo de interacção, alguns aspectos da comunicação estão já escondidos da aplicação, como por exemplo a representação uniforme dos dados. Este mecanismo surgiu com as linguagens procedimentais, tendo sido posteriormente extendido para as linguagens orientadas aos objectos. É particularmente indicado para suportar o paradigma de comunicação *Cliente-Servidor*, o qual pressupõe que os intervenientes têm de estar activos em simultâneo no momento da comunicação. Significa também que a comunicação tem uma semântica *ponto-a-ponto*, o que dificulta a construção de sistemas em que a interacção é efectuada seguindo a semântica *um-para-muitos* ou *difusão*, embora seja possível extender este paradigma de forma a considerar servidores replicados (Cooper, 1984).

Notificações Quando se pretende obter o desacoplamento entre o emissor e o receptor dos dados, uma operação de RPCs síncrona pode ser transformada em duas operações assíncronas: a primeira do cliente para o servi-

dor, que contém um pedido e uma referência para uma operação de retorno; a segunda, efectuada do servidor para o cliente, usando a operação de retorno oferecida pelo cliente, com a resposta ao pedido. Este tipo de interacção pode ser visto já como uma forma de Publicação-Subscrição, onde o cliente (assinante) se regista no servidor (editor) e recebe posteriormente as notificações solicitadas, de forma assíncrona. Com este tipo de interacção, consegue-se já obter desacoplamento no fluxo de dados.

Espaços Partilhados O paradigma da *Memória Partilhada Distribuída* (Gelernter, 1985; Sudell, 1998) proporciona aos participantes de um sistema distribuído um espaço partilhado onde se podem colocar tuplos de informação. Estes tuplos estão acessíveis por todos os participantes de um sistema distribuído. Este paradigma exporta três tipos de operações: (*i*) `out()` que acrescenta um tuplo de informação ao repositório partilhado; (*ii*) `in()`, que importa um tuplo de informação do repositório partilhado, move o tuplo do repositório para o cliente, e (*iii*) `read()` que lê um tuplo do repositório de informação sem o retirar. Com este tipo de interacção, é possível efectuar comunicação por *difusão*, ou seja, um emissor consegue enviar uma mensagem para muitos receptores, mas as operações de leitura são efectuadas de forma síncrona.

Filas de Mensagens Este tipo de interacção em sistemas distribuídos é semelhante aos Espaços Partilhados, no sentido em que existe também um repositório partilhado de troca de informação. Como nos Espaços Partilhados, consegue-se obter desacoplamento entre participantes no tempo e no espaço com este paradigma. A diferença entre estes dois paradigmas está na sua funcionalidade: este paradigma fornece garantias (e.g. ordenação) não necessariamente fornecidas pelos espaços partilhados.

Publicação-Subscrição Neste paradigma os clientes do sistema são de dois tipos: os Editores e os Assinantes. Os clientes interligam-se através de um Serviço de Eventos. A informação é gerada pelos Editores e é dirigida aos Assinantes que mostraram interesse em recebê-la. Uma subscrição por parte de um assinante pode ser efectuada de diferentes formas. Como será descrito nas Secções seguintes, os sistemas Publicação-Subscrição reúnem algumas das qualidades mais interessantes dos paradigmas descritos anteriormente. A possibilidade de efectuar comunicação por *difusão* e, ao mesmo tempo, obter propriedades de desacoplamento entre os participantes de um sistema distribuído são propriedades que facilitam a sua escalabilidade e flexibilidade.

2.1.2 Propriedades de um sistema Publicação-Subscrição

Um sistema Publicação-Subscrição é geralmente constituído por um *Serviço de Eventos* e por dois tipos de clientes: os *Editores* e os *Assinantes*. O serviço de eventos exporta uma interface para cada um destes clientes. Um editor indica ao serviço de eventos a informação que vai publicar e um assinante indica o seu interesse por informação com determinadas características, especificando-as. Depois destas operações, os assinantes começam a receber assincronamente a informação que é publicada pelos editores, de acordo com os padrões indicados.

Mais precisamente, um serviço de eventos exporta as seguintes operações: (*i*) `anunciar()` onde o editor indica que vai começar a publicar informação, indicando vários parâmetros sobre essa informação, (*ii*) a respectiva operação inversa `remover_anuncio()`, (*iii*) `subscrever()` com a qual o assinante mostra interesse em informação com determinadas características, (*iv*) a respectiva operação inversa `remover_subscricao()` e (*v*) `notificar()`

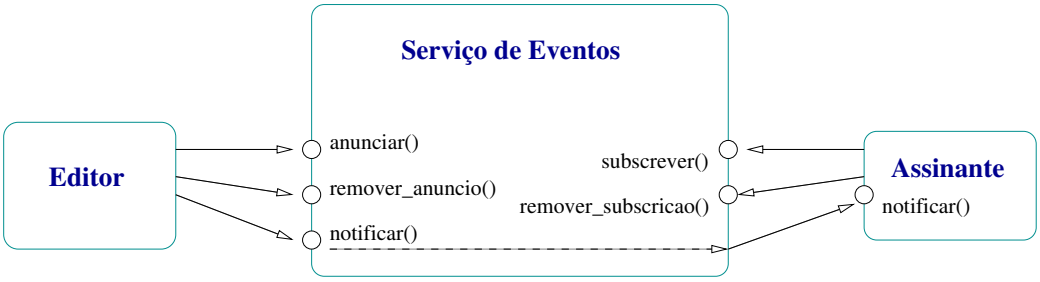


Figura 2.1: Funcionamento de uma arquitectura Publicação-Subscrição.

que é a operação usada pelo editor para publicar a informação propriamente dita. O funcionamento de um sistema Publicação-Subscrição encontra-se ilustrado na Figura 2.1.

Usando este sistema de comunicação, é possível obter propriedades bastante fortes de *desacoplamento* entre os participantes do sistema. O desacoplamento pode ser atingido a vários níveis (Felber, 2001), nomeadamente:

Espaço Os participantes do sistema não precisam de se conhecer entre eles, isto é, não precisam de conhecer a localização, nem o número de entidades, com quem comunicam.

Tempo Normalmente, os participantes do sistema não precisam de estar activos em simultâneo para conseguir comunicar. Um editor pode publicar um determinado evento enquanto o assinante está desligado. Esse assinante recebe o evento mais tarde.

Fluxo de dados Os editores não estão bloqueados enquanto produzem eventos e os assinantes não precisam de estar bloqueados à espera de um evento, isto é, são notificados da ocorrência de um novo evento enquanto estão a efectuar outra operação concorrentemente.

Estas propriedades são bastante importantes quando se deseja obter dinamismo e escalabilidade no sistema. Por exemplo, um sistema Publicação-Subscrição pode ser usado na construção de aplicações que estão constantemente a entrar e a sair do sistema. A construção de aplicações deste género sobre paradigmas como a comunicação em grupo levariam a uma difícil gestão por parte do serviço de filiação. Com um sistema Publicação-Subscrição este problema não se coloca devido às propriedades de desacoplamento.

2.1.3 Modelos de subscrição

Nos sistemas Publicação-Subscrição, os assinantes não recebem todos os eventos que são disseminados pelos editores. Normalmente, estão interessados apenas em alguns desses eventos, eventos esses que reúnem determinadas características que os tornam interessantes. Os diferentes tipos de sistemas Publicação-Subscrição distinguem-se pela forma como os assinantes especificam os eventos em que estão interessados. Estes sistemas podem ser baseados em (*i*) tópicos, (*ii*) conteúdo dos eventos ou (*iii*) tipo dos eventos.

Subscrição por Tópicos

O esquema de um sistema Publicação-Subscrição baseado em subscrição por tópicos é o mais antigo de todos. Os eventos publicados são classificados por tópicos e os assinantes indicam sobre que tópico desejam receber informação. Os tópicos podem estar hierarquicamente divididos em subtópicos e podem ser usados *wildcards* para referenciar vários tópicos ou até indicar que se deseja receber todos os eventos da sub-árvore de um determinado tópico.

Este esquema é bastante parecido com o paradigma de comunicação em grupo: assinar um tópico T pode ser visto como pertencer ao grupo T. Apesar da sua similaridade, a concretização de um sistema Publicação-Subscrição sobre um sistema de comunicação em grupo tem algumas dificuldades no que diz respeito à escalabilidade do sistema. Os tópicos podem ser muito variados, o que leva à utilização de muitos grupos. A informação e largura de banda necessários para gerir todos os grupos pode ser também bastante grande. Por outro lado, mesmo que se seja estabelecido um número limitado de subscrições, é necessário estabelecer um mapeamento entre as subscrições e os grupos (Levine *et al.*, 2000; Guimaraes & Rodrigues, 2003).

Usando o esquema de subscrição por tópicos é possível criar sistemas de filtragem bastante eficientes, que reduzem o custo de comunicação. Apesar disso, este esquema não deixa de ser estático, o que faz com que o poder de expressão por parte de um assinante seja bastante fraco.

Subscrição por Conteúdo

Os sistemas Publicação-Subscrição baseados em subscrição por conteúdo (Carzaniga *et al.*, 2000) foram criados para colmatar a falta de poder de expressão do esquema baseado em subscrição por tópicos. Nesta variante, as subscrições são efectuadas usando as propriedades dos eventos disseminados. Estas propriedades podem ser constituídas pelos atributos do próprio evento ou por *meta-dados* associados ao evento. Assim, um assinante selecciona os eventos que deseja receber usando uma combinação lógica de pares da forma *nome-valor*. Esta combinação forma uma subscrição com um padrão complexo, mas que tem um grande poder de expressão. Esta subscrição é transformada num filtro que é, por sua vez, aplicado

aos eventos.

A subscrição pode ser representada de vários modos, nomeadamente:

String É a forma mais usada para representar subscrições e é necessário usar uma gramática adequada, como o SQL ou a *Default Filter Constraint Language* definida pelo *Object Management Group* (OMG). Estas *Strings* são depois interpretadas pelo sistema.

Objecto Modelo Usando este modo de representação das subscrições, o assinante indica que está interessado em receber todas as notificações do mesmo tipo do objecto modelo. Assim, o assinante vai receber todas as notificações que estão de acordo com os valores colocados no objecto modelo.

Código executável Os assinantes fornecem um objecto que contém métodos capazes de filtrar eventos em tempo de execução. A concretização destes objectos são normalmente deixados a cargo do programador da aplicação.

Através da subscrição por conteúdo, consegue-se atingir uma granularidade bastante fina, mas a complexidade das expressões fazem com que nem sempre se consiga colocar os filtros no ponto óptimo da rede. Este aspecto pode dar origem a um desperdício na largura de banda usada pelo sistema, já que muitos eventos são filtrados apenas no assinante. Por outro lado, para atingir o mesmo poder de expressão com um sistema baseado em subscrição por tópicos, seria necessário criar muitos tópicos diferentes e dividir esses tópicos em muitos sub-tópicos, o que se tornaria inviável.

Subscrição por Tipo

Apesar de ter alguma falta de granularidade, a aproximação baseada em subscrição por tópicos acaba por ser uma forma de agrupar e estruturar a informação disseminada pelo sistema quer no seu conteúdo, quer na própria estrutura do evento. Assim, surgiu uma nova forma de especificar a informação que se quer receber usando a noção de *tipo de eventos* (Eugster *et al.*, 2000). Esta tipificação é efectuada usando os mesmos paradigmas de uma linguagem orientada aos objectos (e.g. Java), ou seja, usando a noção de extensão de tipos. Assim, subscrever um tipo de eventos não é mais do que indicar ao sistema que se deseja obter todos os eventos do tipo T. Se este tipo for extendido formando novos tipos, um assinante que subscreveu os eventos do tipo T vai receber todos os eventos que são deste tipo e dos respectivos subtipos. Desta forma, é possível subscrever a informação mais ampla e vaga ou mais específica, consoante as necessidades da aplicação.

Com esta aproximação é possível efectuar uma filtragem mais eficiente do que com a aproximação anterior, encontrando mais facilmente os locais óptimos para efectuar a filtragem na rede de dados. Isto faz com que se consiga optimizar a largura de banda na disseminação dos eventos. Na aproximação baseada em subscrição por conteúdo é necessário criar uma expressão que indica quais os dados que o assinante deseja receber, expressão esta que terá depois de ser interpretada pelo mediador do sistema Publicação-Subscrição. Com esta aproximação, apenas tem de se indicar qual o tipo de dados, sendo esta informação facilmente interpretada pelo sistema. Este aspecto é outro ponto positivo da aproximação aqui descrita.

2.1.4 Exemplos de sistemas Publicação-Subscrição

Nesta secção são descritos alguns sistemas Publicação-Subscrição, que serviram de base no trabalho efectuado no âmbito desta dissertação e que são representativos dos diversos sistemas existentes. Existem vários tipos de sistemas, nomeadamente (*i*) sistemas Publicação-Subscrição sem mediador, (*ii*) sistemas Publicação-Subscrição com um mediador centralizado e (*iii*) sistemas Publicação-Subscrição com um mediador distribuído, constituído por uma rede de encaminhadores.

Sistema Publicação-Subscrição sem mediador

Num sistema Publicação-Subscrição sem mediador, um assinante regista-se directamente num editor. O editor, por sua vez, envia notificações de forma assíncrona aos assinantes que se registaram. Um exemplo deste tipo de sistemas é o *Cambridge Event Architecture* (CEA) (Bacon *et al.*, 2000). O CEA é um sistema de comunicação assíncrona que segue um paradigma similar ao Publicação-Subscrição, denominado *publish-register-notify*. Neste paradigma, o editor começa por anunciar que tipo de eventos vai publicar e, após receber subscrições, envia notificações assíncronas para os assinantes interessados. A filtragem é normalmente feita pelos assinantes, mas pode também existir um mediador, cuja função consiste precisamente em aliviar os assinantes do processamento de filtragem de eventos. Note-se que na ausência de mediadores, existe uma árvore de disseminação por cada publicação, gerida pelos próprios editores, que dificilmente poderá ser partilhada com outro tipo de notificações ou com outros editores. Isto significa que os sistemas que seguem este paradigma apresentam limitações importantes no que diz respeito às possibilidades de efectuar optimizações dos recursos utilizados. No entanto, talvez a limitação mais

decisiva advenha do facto de que este tipo de sistemas, na versão sem mediador, não possui todas as características de desacoplamento dos outros sistemas Publicação-Subscrição, uma vez que editores e assinantes comunicam directamente entre si.

Sistema Publicação-Subscrição com mediador centralizado

Para que um sistema Publicação-Subscrição tenha todas as propriedades de desacoplamento, é necessário criar um mediador. Como todas as aplicações se registam no mediador, e trocam também informação também através do mediador, já se consegue obter a propriedade de desacoplamento no espaço. O mediador pode ser visto como um canal de comunicação que é usado pelas aplicações. Exemplos deste tipo de aplicações são o *CORBA Event Service* (OMG, 2001), o *CORBA Notification Service* (OMG, 2002) ou o *Java Message Service* (Hapner *et al.*, 2002). Estes sistemas são geralmente constituídos por um mediador e dois tipos de clientes: os produtores e os consumidores. Os clientes registam-se no mediador. Este, tendo informação sobre produtores e consumidores, quando recebe uma notificação de um produtor reencaminha-a para os consumidores interessados.

Uma grande desvantagem neste tipo de sistemas tem a ver com a tolerância a faltas do sistema. Como canal de comunicação, o mediador consiste num ponto de falha do sistema, já que os clientes não conseguem comunicar entre si sem o mediador. O mediador centralizado é também uma limitação no que diz respeito à escalabilidade do sistema, já que tem de manter informação sobre todos os clientes, para além de ter de processar todos os eventos que são disseminados pelos produtores.

Sistema Publicação-Subscrição com mediador distribuído

Para tornar um sistema Publicação-Subscrição mais escalável e tolerante a faltas, o mediador do sistema pode ser composto por uma rede de nós que cooperam entre si de forma a processar os pedidos dos clientes e a disseminação de eventos de forma distribuída. Existem vários sistemas que seguem este paradigma e nesta dissertação irá ser dado ênfase a dois: o *Scalable Internet Event Notification Architecture* (SIENA) (Carzaniga, 1998) e o *Hermes* (Pietzuch & Bacon, 2002).

O SIENA consiste num serviço de notificação de eventos e é composto por editores, assinantes e por um conjunto de encaminhadores, também designados por servidores. Os servidores encontram-se interligados entre si por um protocolo *entre-servidores* e os clientes ligam-se através de um protocolo *cliente-servidor*. Os servidores do SIENA podem ser dispostos de forma hierárquica, numa rede não hierárquica entre-pares ou ainda numa configuração híbrida. Este tipo de configurações faz com que o SIENA seja mais escalável e robusto do que o CEA, uma vez que as árvores de disseminação podem ser optimizadas e a informação para manter todo o sistema se encontra replicada pelos servidores. Mas, uma vez que todos os anúncios terão de ser difundidos por todos os encaminhadores de eventos do sistema, de forma a ser possível encaminhar subscrições para montante, este sistema revela também algumas limitações à capacidade de escala. A informação que é necessária guardar sobre os anúncios e subscrições que o sistema suporta terá de ser replicada pelos servidores, o que significa que o sistema não é escalável no número de anúncios e subscrições a armazenar. Note-se que para reduzir o problema é possível efectuar fusões entre anúncios, sempre que haja uma relação de inclusão entre estes. O mesmo se passa com as subscrições.

O *Hermes* é um sistema Publicação-Subscrição baseado em tipo[1] constituído por três entidades: editores, assinantes e encaminhadores de eventos. Os encaminhadores de eventos comunicam através de uma rede sobreposta à rede normal IP. É sobre essa rede lógica que circulam os eventos e as mensagens de controlo. Estas últimas têm como objectivo distribuir anúncios de publicações e subscrições. Quando um editor (ou um assinante) deseja ligar-se ao sistema estabelece uma ligação com um dos encaminhadores designado por *Ponto de Contacto* que fica responsável por encaminhar os eventos de um determinado tipo entre os editores e os assinantes. Para cada tipo de evento, um dos encaminhadores é deterministicamente nomeado o Ponto de Contacto – esta possibilidade é inerente ao próprio sistema entre-pares Pastry (Rowstron & Druschel, 2001) que serve de suporte ao *Hermes*. Desta forma, quando um assinante deseja subscrever um determinado tipo de evento, comunica com o Ponto de Contacto para que seja definido um caminho entre o editor e o(s) assinante(s). O caminho é sempre definido sobre a rede lógica construída inicialmente. Um dos pontos positivos desta arquitectura consiste no facto de ser mais escalável do que o sistema anterior, já que a informação sobre anúncios e subscrições não necessita de estar replicada por todos os servidores.

2.2 Arquitecturas com Qualidade de Serviço

Certas aplicações de sistemas distribuídos necessitam de ter a possibilidade de expressão de parâmetros de Qualidade de Serviço (QdS) na comunicação, como a largura de banda ou a latência. Esta possibilidade é inexistente em muitos sistemas e mais flagrante quando o sistema distri-

[1] Os eventos são tipificados e a subscrição dos eventos é feita primeiramente pelo tipo, sendo também possível adicionar critérios de filtragem.

buído é construído sobre um modelo de comunicação indirecta do tipo Publicação-Subscrição. Apesar das vantagens do modelo Publicação-Subscrição, a falta de suporte à expressão de parâmetros de QdS é uma limitação importante. Certo tipo de aplicações (e.g. disseminação de vídeo ou monitorização de sistemas) necessita de garantias de QdS para o seu bom funcionamento.

Por outro lado, as soluções existentes para a construção de sistemas com garantias de QdS não podem ser aplicadas directamente em sistemas do tipo Publicação-Subscrição, por serem essencialmente dirigidas aos sistemas de comunicação directa. Os sistemas que fornecem garantias de QdS são tipicamente baseados no estabelecimento explícito de canais ou ligações, os quais servem de suporte para efectuar as reservas dos recursos necessários para fornecer a QdS pretendida. Esta abordagem encaixa naturalmente nos sistemas de comunicação directa, onde as ligações entre elementos de um sistema são explicitamente efectuadas, mas é de difícil utilização no modelo Publicação-Subscrição. Neste modelo, as aplicações não devem ser forçadas a estabelecer canais de comunicação e não devem precisar de ter conhecimento acerca da localização e do número de participantes envolvidos na comunicação. Devem apenas preocupar-se com a qualidade da informação que manipulam. Nesta secção são descritos modelos existentes que têm como objectivo fornecer garantias de QdS na comunicação entre aplicações de sistemas distribuídos.

Para garantir QdS na comunicação entre aplicações de um sistema distribuído, sistema esse que funciona sobre uma rede normal IP, existem duas arquitecturas distintas: os *Serviços Integrados* (Braden *et al.*, 1994) e os *Serviços Diferenciados* (Blake *et al.*, 1998). Para além destas arquitecturas, foi ainda proposto um conjunto de extensões a protocolos de encaminha-

mento, que são também discutidas nesta dissertação.

2.2.1 Serviços Integrados

Os Serviços Integrados (Braden *et al.*, 1994) consistem num conjunto de mecanismos que estabelecem um caminho entre o emissor e o receptor e reservam os recursos necessários nesse caminho. Os intervenientes do sistema necessitam de manter estado sobre os recursos reservados.

A *framework* dos Serviços Integrados fornece às aplicações a possibilidade de escolha entre vários mecanismos de entrega de pacotes de dados, assim como a reserva dos recursos necessários para garantir uma determinada QdS. Para suportar esta possibilidade são necessárias duas coisas: (*i*) os elementos da rede (e.g. encaminhadores) que estão entre o emissor e o receptor têm de suportar os mecanismos de controlo da QdS que esta a ser garantida pelo sistema e (*ii*) tem de existir uma forma de anunciar os requisitos das aplicações aos elementos da rede que estão no caminho entre o emissor e o receptor. Este último deve também efectuar troca de informação de controlo entre os elementos da rede, para que seja efectuada a gestão da QdS fornecida. Assim, os Serviços Integrados são constituídos por um *Controlador de Tráfego* e um *protocolo de sinalização*. O Controlador de Tráfego é constituído por um conjunto de mecanismos de controlo que gerem os recursos existentes (e.g. largura de banda) e o protocolo de sinalização tem como função a troca de informação entre os elementos da rede. A Figura 2.2 ilustra os vários componentes dos Serviços Integrados e a interacção entre eles.

O Controlador de Tráfego é constituído por três componentes: o *Escalonador de Pacotes*, o *Controlo de Admissão* e o *Classificador*.

Escalonador de Pacotes Este componente gere a comutação das mensagens,

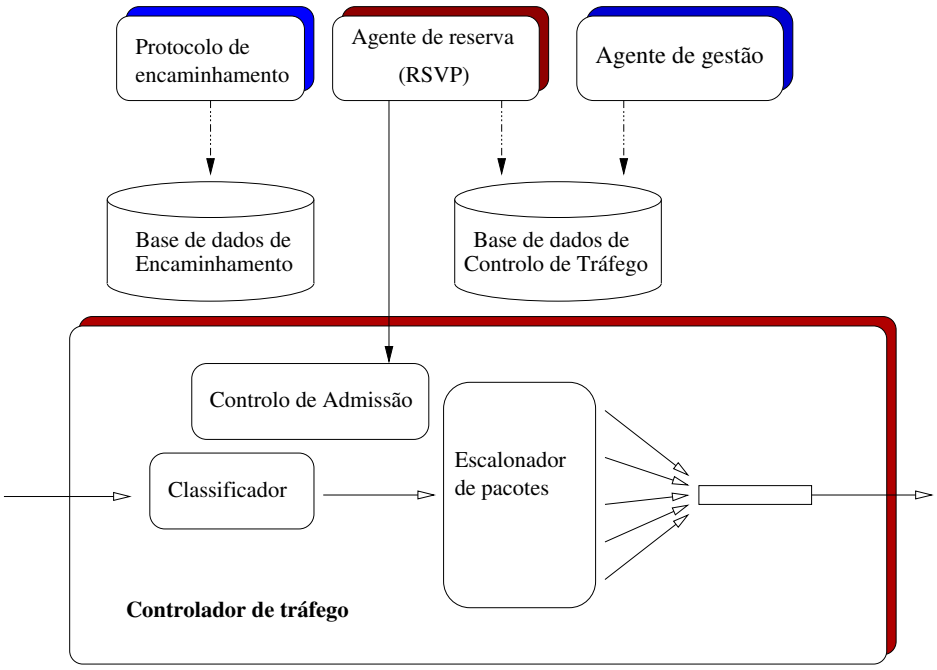


Figura 2.2: Arquitectura de Serviços Integrados.

usando um conjunto de filas. Estas filas podem ser concretizadas usando diferentes políticas e existem várias filas para vários fluxos de dados;

Controlo de Admissão Este módulo concretiza o algoritmo de decisão que um determinado encaminhador usa para determinar se um novo fluxo de dados (com determinados requisitos de QdS) pode ser garantido sem comprometer a QdS dos restantes fluxos.

Classificador As mensagens necessitam de ser classificadas antes de serem tratadas pelo escalonador. Esta classificação é passada ao escalonador de mensagens, que usa esta informação para determinar que tipo de tratamento é dado à mensagem.

Para além destes mecanismos, é necessário um protocolo de reserva

de recursos. Este protocolo é necessário para criar e manter informação específica sobre os fluxos de dados em todos os intervenientes na comunicação – o emissor, o receptor e os encaminhadores. Para este efeito, é usado o *ReSerVation Protocol* (RSVP) (Braden *et al.*, 1997). Se o componente de Controlo de Admissão aceitar o novo pedido, são efectuadas as devidas alterações ao Classificador e ao Escalonador de Pacotes de forma a garantir também a QdS ao novo fluxo. Finalmente, todos os encaminhadores têm um Agente de Gestão. Este agente terá de ser capaz de modificar a base de dados do Controlo de Tráfego para gerir políticas de controlo e partilha de recursos.

O protocolo de reservas pode ser concretizado de várias formas, dependendo do *Modelo de Reservas*. O Modelo de Reservas descreve como uma aplicação pode negociar a reserva de uma determinada QdS. O modelo mais simples consiste em responder positiva ou negativamente a um pedido específico. Outra hipótese é dar à aplicação a oportunidade de escolha num determinado conjunto de classes de QdS ou ainda escolher qualquer valor entre um espaço de especificação de fluxos. O protocolo de reserva de recursos é iniciado pelo receptor dos dados. Um receptor pede uma reserva que lhe seja apropriada. Este pedido segue todo o caminho até ao emissor. Se todos os encaminhadores permitirem o novo fluxo, as reservas são efectuadas e as alterações necessárias à base de dados do Controlo de Tráfego de cada encaminhador. O facto de o protocolo ser iniciado pelo receptor facilita o tratamento de receptores heterógeneos, sendo também coerente com o *IP Multicast*.

A reserva de recursos é efectuada por fluxo e tem de ser coerente com o encaminhamento de dados. Quando um novo fluxo é criado, as reservas de recursos são efectuadas nos encaminhadores que estão no caminho

desse fluxo. Assim, a QdS só pode ser garantida se o encaminhamento for efectuado pelos encaminhadores que reservaram os recursos para esse fluxo.

2.2.2 Serviços Diferenciados

A solução encontrada com os Serviços Integrados para oferecer garantias de QdS na comunicação necessita que haja troca de informação entre os encaminhadores. Esta troca de informação serve para manter coerência de estado sobre os fluxos de informação e reservas existentes, mas também para a admissão de novos fluxos. Esta solução não escala com um aumento do número de nós da rede. Uma solução mais escalável no número de nós do sistema consiste nos Serviços Diferenciados.

A arquitectura de *Serviços Diferenciados* (Blake *et al.*, 1998) é baseada num modelo simples onde o tráfego que entra numa determinada rede é classificado e condicionado nas fronteiras dessa rede, sendo atribuído a diferentes classes. A cada classe de serviço é atribuído um diferente comportamento no encaminhamento do tráfego. Cada agregado de tráfego (ou classe) é identificado por um único código. Este código é colocado directamente no cabeçalho IP da mensagem, usando o campo *Differentiated Service* (Nichols *et al.*, 1998). No núcleo da rede, os pacotes são encaminhados tendo em conta esse código. Numa rede de Serviços Diferenciados existem dois tipos de nós: os nós fronteira e os nós interiores. Os nós fronteira são os responsáveis por efectuar a ligação entre redes de Serviços Diferenciados e as outras redes. Cada nó fronteira é responsável pela classificação do tráfego aquando a sua entrada. A Figura 2.3 ilustra uma rede de Serviços Diferenciados.

A arquitectura de Serviços Diferenciados atinge escalabilidade colo-

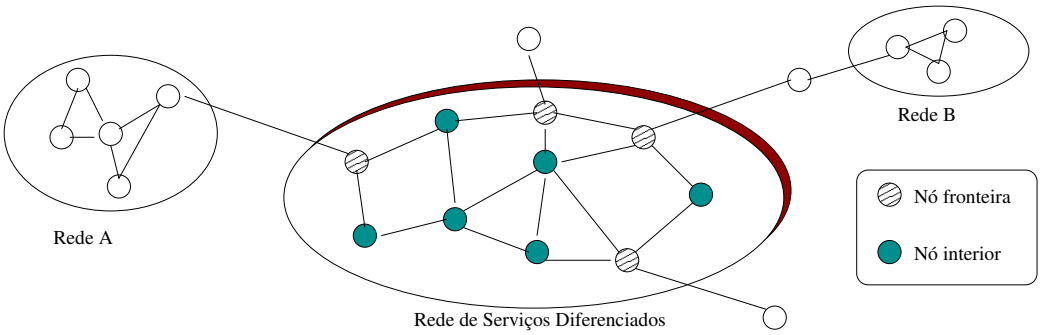


Figura 2.3: Rede de Serviços Diferenciados.

cando a complexidade do processamento apenas nos nós fronteira. Os nós interiores da rede de Serviços Diferenciados têm um comportamento *por-salto*, permitindo a reserva de recursos em cada nó e não necessitam de manter estado de encaminhamento para diferentes fluxos de dados. Os algoritmos de classificação e marcação de pacotes – que requerem mais processamento – apenas precisam de correr nos nós fronteira, e não nos nós interiores da rede. Os recursos da rede são reservados usando políticas de fornecimento de serviço, políticas estas que indicam como é que o tráfego deve ser marcado aquando a sua entrada na rede de Serviços Diferenciados e como é que deve ser encaminhado dentro da rede.

Esta arquitectura é composta um número de elementos funcionais localizados nos nós da rede, incluindo um conjunto de comportamentos de encaminhamento e funções de classificação de pacotes, concretizando uma diferenciação entre vários *Serviços* numa rede. Um Serviço é internamente definido usando algumas características da transmissão de pacotes numa direcção. Estas características podem ser especificadas de uma forma quantitativa em termos de fluxo de dados, atraso na rede, taxa de perda, entre outros. A diferenciação de serviços é necessária quando se deseja acomodar requisitos de várias aplicações heterogéneas.

2.3 Encaminhamento de tráfego

As arquitecturas apresentadas na secção anterior não têm em conta o encaminhamento de tráfego com QdS. Por exemplo, no caso dos Serviços Integrados apenas existem mecanismos de controlo do tráfego e um protocolo de sinalização que é usado para efectuar reservas de recursos. A arquitectura é explicitamente independente do protocolo de encaminhamento que o sistema está a usar. Assim, o protocolo não tem em conta parâmetros de QdS na decisão do encaminhamento das mensagens.

Quando um protocolo efectua encaminhamento de tráfego sem ter em conta parâmetros de QdS, este pode ser encaminhado por caminhos já congestionados ou onde não existem mais recursos. Um caminho maior para um protocolo de encaminhamento usual pode significar um caminho por onde existem ainda recursos. Nesta secção irão ser descritas algumas formas de encaminhamento que têm em conta parâmetros de QdS, tentando assim optimizar a utilização dos recursos da rede de dados.

2.3.1 Protocolos de encaminhamento

Para efectuar encaminhamento de tráfego tendo em conta parâmetros de QdS várias soluções foram apresentadas. Uma das soluções consiste em extensões ao protocolo de encaminhamento *Open Short Path First* (OSPF) (Moy, 1994b). O OSPF é um protocolo de encaminhamento interno que se baseia no estado das ligações entre os nós. Para tal, mantém uma base de dados que descreve a topologia do sistema autónomo. Esta base de dados é usada para calcular o caminho mais curto até ao destino de uma determinada mensagem. Um nó que corre OSPF detecta facilmente as mudanças de estado nas ligações entre nós do sistema e propaga o seu

novo estado local para todos os seus vizinhos. Estas alterações são também propagadas para os restantes elementos, sendo a rede inundada até que todos os nós da rede voltam a ter o seu estado coerente. As extensões efectuadas ao OSPF para suporte de encaminhamento com QdS (QoSPF) (Apostolopoulos *et al.*, 1999) usam o mesmo mecanismo. Neste caso, para além da topologia da rede, os nós guardam informação sobre os recursos existentes. Quando se efectuam reservas de largura de banda numa parte da rede, as alterações são propagadas pelos restantes elementos para que estes actualizem a sua base de dados. Com esta informação, os encaminhadores podem calcular o caminho até um determinado ponto da rede tendo em conta várias métricas. Neste caso, o QoSPF tem em conta as seguintes métricas:

Largura de banda de cada ligação Só são seleccionados caminhos onde haja uma quantidade disponível de largura de banda maior ou igual à largura de banda requerida;

Latência de cada ligação A latência de cada ligação terá também de ser anunciada pelo protocolo e serve para eliminar os caminhos cuja soma das latências das várias ligações que compõem o caminho excede a latência mínima pedida;

Número de saltos até cada destino Esta métrica é usada para medir o custo de um determinado caminho para a rede. Um caminho com menos saltos na rede, normalmente ocupa menos recursos nessa rede. Assim, perante vários caminhos que podem suportar um determinado pedido, é escolhido o que tiver menos saltos.

Com estas métricas e com a informação sobre a topologia da rede e os recursos disponíveis, o QoSPF aplica um algoritmo de *Dijkstra* que, para

cada número de saltos, encontra o caminho com maior valor de largura de banda, removendo os caminhos que não obedecem ás restrições de latência.

Estes são os princípios básicos para se conseguir obter encaminhamento de tráfego tendo em conta requisitos de QdS, mas o protocolo até agora descrito não suporta difusão. Os protocolos de encaminhamento que suportam difusão têm como função construir uma árvore de disseminação que liga o emissor dos dados aos respectivos receptores. Estas árvores podem ser construídas de várias formas:

Árvore baseada no emissor Neste caso, o ponto de ligação entre uma árvore e um novo receptor é definido pelo emissor. O receptor começa por enviar uma mensagem directamente ao emissor, que depois calcula o ponto de ligação na árvore para esse emissor. Este paradigma é seguido por protocolos como o *Multicast OSPF* (MOSPF) (Moy, 1994a) ou o *Source Specific Multicast* (SSM) (Diot *et al.*, 2003). Estes protocolos são particularmente interessantes na obtenção de caminhos eficientes do emissor até aos receptores, mas apenas (*i*) na presença de poucos receptores e (*ii*) quando se conhece antecipadamente o emissor;

Árvore baseada num ponto central Neste tipo de protocolos, a árvore é construída tendo em conta um ponto central, normalmente bem conhecido. Através deste ponto, consegue-se construir uma árvore partilhada por todos os utilizadores de um grupo, sejam eles emissores ou receptores. Todos os emissores de um grupo partilham uma única árvore, originando difusão *Muitos-para-Muitos*. Esta aproximação foi aplicada pelo *Core Based Tree* (CBT) (Koh & Kang, 2001) e pelo *Protocol Independent Multicast-Sparse Mode* (PIM-SM) (Estrin *et al.*, 1998);

Árvore baseada em parâmetros de QdS Por último, existem as árvores de disseminação que são construídas com base em parâmetros de QdS. Um dos exemplos de sistemas que seguem este paradigma é o *QoS Manager for Internet Connections* (QoSMIC) (Yan *et al.*, 2002). O QoSMIC constroi a árvore tendo a restrição da largura de banda disponível. Um novo receptor junta-se ao sistema ligando-se ao ponto mais próximo da árvore que tiver disponível a largura de banda necessária. Para tal, é necessário que os nós envolvidos na árvore tenham conhecimento sobre a topologia da rede, incluindo a largura de banda disponível e latência das ligações. Esta necessidade aplica-se também a outros protocolos do mesmo género[2].

Os protocolos como o CBT, cuja árvore é construída a partir de um ponto central, descobrem esse ponto utilizando uma função de dispersão, que mapeia um grupo *G* num endereço de forma determinista. Assim, ao contrário das outras formas de construir árvores de difusão, não é necessário manter estado adicional. Este aspecto reduz a quantidade de memória gasta para manter o protocolo, assim como a quantidade de mensagens a trocar entre os nós da rede. Um aspecto negativo desta aproximação prende-se com a selecção do nó central da árvore: o algoritmo pode não escolher o nó mais apropriado e a quantidade de recursos consumidos pode não ser óptima. Outro aspecto negativo prende-se com a concentração do tráfego no ponto central da rede.

Note-se ainda que o problema de encontrar a árvore de difusão óptima já foi identificado e estudado e consiste no problema das árvores de *Steiner* (Ramanathan, 1996). Já foi provado que encontrar a solução óptima

[2]Por exemplo, o algoritmo *QoS Dependent Multicast Routing Algorithm* (QDMR) (Guo & Matta, 1999).

para uma árvore de *Steiner* é *NP-Completo* (Garey & Johnson, 1977). É por esta razão que a construção de árvores de difusão que têm em conta a QdS apenas o fazem para um parâmetro (tipicamente a largura de banda ou a latência), tentando depois minimizar os restantes.

2.3.2 Sistemas entre-pares

Outra forma de obter encaminhamento de dados numa rede é usando sistemas entre-pares (do inglês *Peer-to-Peer*). Os sistemas entre-pares são redes sobrepostas sobre a rede IP, que escondem algumas características da rede (que não são normalmente necessárias para as aplicações) e que fornecem às aplicações uma forma de encaminhar mensagens baseada numa *Tabela de Dispersão Distribuída*[3] (TDD). Uma TDD consiste numa tabela de dispersão, onde os pares $(chave, valor)$ estão distribuídos por vários nós de uma rede entre-pares. Com esta opção, para encaminhar uma mensagem basta fazer um pedido à TDD com uma determinada chave. A mensagem é encaminhada de forma automática pelo sistema entre-pares.

De seguida serão descritos alguns exemplos deste tipo de sistemas: o *Pastry* (Rowstron & Druschel, 2001), o *Tapestry* (Zhao *et al.*, 2001) e o TOPLUS (Ross *et al.*, 2003).

Pastry Cada nó numa rede *Pastry* tem um identificador único no sistema. Na presença de uma mensagem e de um valor numérico, um nó *Pastry* encaminha para o nó (vizinho) que está numericamente mais próximo desse valor e assim sucessivamente até chegar ao nó destino. Através desta forma de encaminhamento, o número esperado de saltos de uma mensagem numa rede *Pastry* é da ordem de $O(\log(N))$, onde N é o número de

[3]Expressão traduzida do inglês *Distributed Hash Table*.

nós existentes na rede *Pastry*. Este sistema tem em conta a localização real dos nós da rede, ou pelo menos tenta ter uma noção de distância entre os nós, já que procura minimizar essa distância de acordo com o número de saltos do encaminhamento da rede IP.

Tapestry O *Tapestry* é uma infraestrutura de localização e encaminhamento de mensagens, que fornece um serviço de encaminhamento directamente para a cópia mais próxima de um determinado objecto ou serviço, sem necessidade de uso de recursos centralizados. O *Tapestry* é diferente do *Pastry* na forma como encaminha as mensagens. Inspirado no *Plaxon* (Plaxton *et al.*, 1997), o *Tapestry* faz o encaminhamento de mensagens de uma forma incremental. Cada nó tem uma tabela de vizinhos com múltiplos níveis, onde cada nível representa um sufixo do identificador do nó de destino. Assim, uma mensagem para o nó 4598 é encaminhada dígito por dígito (e.g. ***8 → **98 → *598 → 4598). O número de saltos máximo no encaminhamento de uma mensagem é da ordem de $O(\log_b(N))$, num sistema com o espaço de nomes de tamanho N, usando identificadores de base b.

Topology-Centric Look-Up Service (TOPLUS) O TOPLUS é também um serviço de encaminhamento sobre redes entre-pares, mas que tem em consideração a topologia real da rede. Os resultados do encaminhamento efectuado pelo TOPLUS são comparáveis ao encaminhamento do IP. Neste sistema, os nós são agrupados e dispostos de uma forma hierárquica. As ligações entre os vários grupos são efectuadas em forma de árvore. O encaminhamento é efectuado de uma forma muito simples: em cada ponto da sub-árvore, o TOPLUS verifica se o destino está num dos ramos; se estiver, a mensagem é enviada para esse ramo, senão a mensagem é enviada

para a hierarquia acima. Os grupos são formados usando o endereço IP dos nós.

Com este tipo de redes, a construção das aplicações está bastante facilitada, mas o caminho que uma mensagem percorre até chegar ao seu destino (do ponto de vista da rede IP) pode não ser o óptimo. Para colmatar esta falha, existem redes entre-pares que têm em conta a localização geográfica dos nós, embora este aspecto esteja ainda a ser objecto de estudo.

2.4 Expressão da QdS pelas Aplicações

Nesta secção são apresentadas as características que servem de linhas condutoras na definição de arquitecturas Publicação-Subscrição com QdS, no que diz respeito à expressão de requisitos de QdS por parte das aplicações. Uma das grandes vantagens dos sistemas Publicação-Subscrição consiste no desacoplamento entre os participantes do sistema (Felber, 2001). Para além dos vários tipos de desacoplamento já existentes, de seguida vai ser dado ênfase a uma nova noção de desacoplamento: o desacoplamento na QdS. Este desacoplamento separa os parâmetros de QdS do tipo e conteúdo dos eventos disseminados no sistema.

Em sistemas Publicação-Subscrição, as reservas necessárias para a disseminação com QdS devem ser efectuadas em tempo de execução, baseadas nas propriedades definidas pelos assinantes, nas propriedades dos eventos anunciados pelos editores e nos recursos disponíveis. Um aspecto importante a considerar no sistema é que os assinantes devem poder expressar os seus requisitos de QdS usando uma notação idêntica à que usam para definir outro tipo de requisitos, nomeadamente requisitos relaciona-

dos com o conteúdo das notificações. Por outro lado, os editores devem anunciar as *características* da sua publicação, na forma de um *perfil de QdS*. Esta informação é usada em tempo de execução pelo sistema para estimar a quantidade de recursos necessários para um determinado fluxo de eventos e para verificar se os pedidos dos assinantes podem ser satisfeitos.

Para exemplificar como deverão ser efectuadas as publicações e subscrições no sistema apresentado, considere-se o seguinte exemplo, originalmente introduzido em (Araújo & Rodrigues, 2002). Um edifício com várias divisões está equipado com sensores de temperatura. Estes sensores anunciam a temperatura da sua divisão num evento do tipo *Temp*, com os seguintes atributos: (*i*) *sala*, que indica onde está a ser recolhida a temperatura, (*ii*) *temperatura*, que indica qual a temperatura da sala aquando da disseminação do evento e (*iii*) *precisão*, que indica a precisão do sensor. Nos exemplos seguintes será usada uma notação baseada em (Eugster *et al.*, 2001). Usando este modelo, as subscrições podem ser feitas do seguinte modo:

Subscriber s = **subscribe** Temp
where (sala = $sala1$)

ou

Subscriber s = **subscribe** Temp
where (temperatura $>$ 60)

A primeira expressão corresponde a uma subscrição de eventos provenientes da sala $sala1$ e a segunda corresponde a uma subscrição de eventos de qualquer sala, em que a temperatura excede os 60 graus. As publicações podem ser definidas da seguinte forma:

```
Publisher p = new Publisher
              of Temp
              withProfile (sala = sala1,
                           temperatura = qualquer,
                           precisão = 0.01);


e = new Temp (sala = sala1,
              temperatura = 16,
              precisão = 0.01)


p.publish (e)
```

Publisher consiste num componente auxiliar que é usado para disseminar eventos. Permite também informar o sistema sobre o tipo de eventos a produzir. Esta informação toma a forma de anúncios no sistema. No exemplo anterior é apenas considerado o perfil do conteúdo da informação que vai ser disseminada. Esta informação pode ser usada pelo sistema para optimizar a disseminação de eventos (Carzaniga *et al.*, 2001). De seguida será discutida uma forma de anunciar informação sobre QdS em conjunto com a informação sobre o tipo de evento.

Considere-se agora que cada um destes sensores tem diferentes QdS's associadas, por exemplo, um sensor dissemina eventos esporadicamente, apenas quando a temperatura atinge um determinado valor e outro sensor dissemina a temperatura periodicamente. A informação sobre QdS deverá ser incluída quer nos anúncios, quer nas subscrições. A adição de meca-

nismos que permitem às aplicações expressar parâmetros de QdS pode ser efectuado de várias formas.

Uma aproximação possível consistiria em codificar a informação sobre a QdS no tipo de evento. Por exemplo, se um sensor disseminasse *esporadicamente* um evento com a informação da temperatura numa determinada sala e outro sensor disseminasse *periodicamente* um evento também sobre a temperatura de uma determinada sala, poderiam ser criados dois subtipos, um para disseminação esporádica e outro para periódica. Apesar de intuitiva, esta aproximação teria algumas desvantagens. Com vários atributos de QdS, esta aproximação levaria a uma explosão de tipos diferentes para a mesma informação. Outra razão para rejeitar esta aproximação consiste também no facto de só ser possível determinar alguns atributos de QdS em tempo de execução, como por exemplo a latência. A latência depende da localização do assinante em relação ao editor e da carga dos nós intermédios numa determinada altura.

Assim, tendo em conta os aspectos anteriores, as operações de anúncio e subscrição devem ser enriquecidas com atributos de QdS, que podem ser definidos de forma semelhante à definição de conteúdos de informação. Para tal, o editor terá de anunciar um *perfil* do tipo de informação que vai ser disseminada e também um perfil da QdS necessária para disseminar os eventos. As operações deverão ser da seguinte forma:

```
Publisher p = new Publisher
              of Temp
              withProfile (sala = sala1,
                           temperatura = qualquer,
                           precisão = 0.005)
              withQoSProfile Esporadico
```

É de notar ainda que alguma informação, como o período dos eventos, ajuda a caracterizar a forma do tráfego que vai ser produzido pelo editor. Desta forma consegue-se também uma separação entre o tipo de informação e a QdS necessária para a sua disseminação. O assinante deverá expressar os requisitos de QdS usando uma condição de filtragem semelhante à que é usada para o conteúdo:

```
Subscription s = subscribe Temp
                 where (temperatura > 60)
                 withQoS ((Periodico(periodo < 1))
                          and (latencia < 10))
```

Note-se que esta informação deverá ser interpretada pelo sistema de forma a efectuar não só as correspondências de informação, como as reservas necessárias. Note-se também que algumas subscrições podem não ser aceites por falta de recursos disponíveis. Em suma, deverá ser o sistema a efectuar as ligações e as reservas de recursos de forma transparente para a aplicação.

2.4.1 Modelação e caracterização do tráfego

Caberá às próprias aplicações fazerem a caracterização do tráfego que pretendem publicar ou subscrever. Esta caracterização deverá corresponder a um compromisso possível entre dois objectivos antagónicos. Por um lado, deve ser possível às aplicações gerarem tráfego com um padrão que seja suficientemente flexível de forma a corresponder às suas necessidades. Por outro lado, a caracterização deve fornecer informação suficiente ao sistema de forma a permitir a criação de reservas de recursos com dimensão adequada, tendo em vista a desejável optimização na utilização desses mesmos recursos.

Modelo de limites de tráfego

Uma das contribuições de (Bettati *et al.*, 1999) consiste num modelo de caracterização de tráfego numa rede de dados, que permite a definição de descritores de tráfego precisos. Neste modelo são usadas funções de tráfego máximo e mínimo para definir um volume de tráfego gerado por um ou mais emissores.

Bettati *et al.* começa por definir uma função $R_{i,X}(t)$ que consiste na quantidade de dados de uma ligação M_i proveniente do nó X, no intervalo de tempo $[0,t)$. $R_{i,X}(t)$ descreve o tráfego que sai do nó X pela ligação M_i. Mas esta função está dependente do tempo, não sendo muito adequada como descritor de tráfego. São então consideradas duas funções que descrevem os limites esperados de tráfego numa ligação e que podem ser usadas para descrever volumes de tráfego para essa ligação. $F_{i,X}(I)$ e $f_{i,X}(I)$ definem na quantidade máxima e mínima de dados que parte do nó

X pela ligação M_i durante o intervalo de tempo I:

$$F_{i,X}(I) = max_{s \geq 0}(R_{i,X}(s+I) - R_{i,X}(s))$$

$$f_{i,X}(I) = min_{s \geq 0}(R_{i,X}(s+I) - R_{i,X}(s))$$

As funções $F()$ e $f()$ são usadas como descritores de tráfego nas ligações entre os nós de uma rede e formam a mais fina caracterização de trágego que sai de um nó da rede, para além de ser determinista e invariante no tempo. As funções definidas anteriormente podem ser definidas por um grande número de pontos, o que as torna pesadas de manipular. Assim, são usados limites das funções máximo e mínimo ($B(I)$ e $b(I)$) para caracterizar o tráfego de dados e efectuar a respectiva reserva de recursos. Assim, é necessário que estas funções caracterizem o tráfego de uma forma o mais fina possível para prevenir reservas excessivas de recursos. Terá de existir um limiar entre caracterização fina (o que implica manipular as funções com mais pontos) e ligeireza na manipulação das funções.

Modelo de balde de testemunhos

O modelo de *balde de testemunhos*[4] permite um tratamento matemático relativamente simples por parte do sistema, permitindo simultaneamente a geração de padrões de tráfego suficientemente complexas por parte das aplicações. A utilização do modelo de balde de testemunhos na caracterização do tráfego das aplicações está descrito (Tang & Tai, 1999).

Um balde de testemunhos é um contador não negativo que acumula testemunhos a uma taxa constante r até atingir a capacidade máxima b. Para caracterizar o tráfego de dados usando este modelo, é necessário de-

[4]Expressão traduzida do inglês *Token Bucket*.

finir valores apropriados para r e b de forma a que todos os eventos sejam enviados, satisfazendo os requisitos da aplicação.

Este modelo pode ser extendido, acrescentando uma fila. O tamanho desta fila (que vamos denominar q) pode ser definido através dos parâmetros r e b (Tang & Tai, 1999). A fila é usada para atenuar tráfego em rajadas e para baixar os valores dos parâmetros r e b. Neste último caso fará com que os recursos a alocar sejam menores. Por outro lado, a fila tem custos: reserva adicional de memória e atraso dos pacotes de dados.

2.5 Sumário

As propriedades inerentes aos sistemas Publicação-Subscrição fazem com que este tipo de sistemas se torne bastante atractivo na construção de certo tipo de aplicações em sistemas distribuídos. Quando se pretende efectuar difusão entre os vários participantes e, ao mesmo tempo, obter um elevado grau de escalabilidade, um sistema Publicação-Subscrição adequado facilita a construção das aplicações. Os sistemas Publicação-Subscrição conseguem obter um elevado grau de escalabilidade e flexibilidade devido ás suas propriedades de desacoplamento, porque os participantes não necessitam de manter informação sobre os outros participantes com quem comunicam. É também possível adicionar e remover aplicações ao sistema, sem que seja necessário efectuar paragens. Estas propriedades só podem ser inteiramente garantidas nos sistemas Publicação-Subscrição. A Tabela 2.1 confronta os vários modos de interacção estudados, indicando quais as propriedades de desacoplamento que conseguem ser garantidas.

Os sistemas Publicação-Subscrição baseados em tópicos são os mais simples e fáceis de construir, mas são também os menos flexíveis no que

Tipo de Interacção	Desacoplamento		
	Espaço	Tempo	Fluxo de dados
Protocolos de Transporte	Não	Não	Emissor
Comunicação em Grupo	Não	Não	Sim
RPC	Não	Não	Emissor
Notificações	Não	Não	Sim
Espaços Partilhados	Sim	Sim	Emissor
Filas de Mensagens	Sim	Sim	Emissor
Publicação-Subscrição	Sim	Sim	Sim

Tabela 2.1: Propriedades de desacoplamento dos vários modos de interacção.

toca ao poder de expressão por parte das aplicações. Por outro lado, os sistemas baseados em conteúdos são os que têm o mais elevado poder de expressão, mas a interpretação das expressões criadas pelas aplicações pode levar um algum desperdício de recursos no sistema. Para obter um considerável poder de expressão por parte das aplicações sem comprometer os recursos do sistema, pode-se optar por concretizar um sistema baseado em subscrição por tipo, tendo filtros quando necessário. Também se consegue obter um maior grau de escalabilidade se o serviço de eventos for constituído por uma rede de encaminhadores. Desta forma consegue-se obter também quer um bom grau de tolerância a faltas, quer uma maior optimização da largura de banda utilizada. A replicação da informação sobre as publicações e subscrições por todos os encaminhadores que fazem parte do Serviço de Eventos também é de evitar, sendo mais sensato o uso de Pontos de Contacto. A informação armazenada pelos encaminhadores determinados como Pontos de Contacto pode ser também replicada para obter um maior grau de tolerância a faltas.

As arquitecturas existentes que visam oferecer garantias de QdS na comunicação de um sistema distribuído distinguem-se em dois aspectos: por um lado a arquitectura de Serviços Integrados suporta uma grande quan-

tidade de classes de serviço. Os clientes podem efectuar pedidos de reservas e para isso é usado um protocolo de sinalização; por outro lado, com os Serviços Diferenciados apenas um número limitado de classes de serviço são suportadas. Apesar disso, a forma encontrada nesta arquitectura para suportar QdS é bastante mais escalável, já que não necessita de trocar informação para efectuar reservas de recursos. Nesta arquitectura, as políticas das classes de serviço são definidas de uma forma estática e os utilizadores apenas têm de escolher uma das classes de serviço fornecidas.

As arquitecturas descritas na Secção 2.2 não definem como deve ser efectuado o encaminhamento das mensagens que são trocadas pelo sistema. O encaminhamento é efectuado por um protocolo genérico, exterior a este tipo de arquitecturas. Este aspecto pode ser bastante negativo para o sistema: um protocolo de encaminhamento que não tem em conta os requisitos de QdS das aplicações pode encaminhar o tráfego por caminhos já congestionados ou sem recursos disponíveis, recursos estes que podem existir tomando outros caminhos. Existem vários protocolos de encaminhamento que têm em conta requisitos de QdS. Outra forma de efectuar encaminhamento de tráfego prende-se com o uso de redes entre-pares e *Tabelas de Dispersão Distribuídas* (DDT). Com este tipo de redes, a construção das aplicações está bastante facilitada, mas o caminho que uma mensagem percorre até chegar ao seu destino pode não ser o óptimo. Para colmatar esta falha, existem redes entre-pares que têm em conta a localização geográfica dos nós, embora este aspecto esteja ainda a ser objecto de estudo. Uma das desvantagens deste tipo de sistemas consiste na inexistência de serviços de redes entre-pares que ofereçam garantias de QdS ou uma forma de efectuar reservas de recursos de rede.

Num sistema Publicação-Subscrição todo o tráfego relevante é gerado

pelas aplicações. Assim, têm de ser estas a indicar ao sistema o volume de tráfego que vai ser gerado. A melhor forma até agora encontrada para o efeito consiste em enriquecer as operações de anúncio e subscrição com atributos de QdS. A forma de expressão destes atributos deve ser suficientemente flexível, permitindo a geração de padrões de tráfego suficientemente complexas por parte das aplicações e, simultaneamente, um tratamento matemático bastante simples por parte do sistema. Estas características podem ser encontradas no modelo de balde de testemunhos. Os anúncios e as subscrições devem estar acompanhadas com informação sobre a modelação de tráfego, para além do tipo de informação que vai ser publicada/recebida. Mais precisamente, o editor deverá anunciar (*i*) o tipo de informação que vai publicar e (*ii*) uma modelação do tráfego que vai ser gerado para essa publicação. O assinante, por outro lado, deve subscrever (*i*) o tipo apropriado de informação que deseja receber, (*ii*) uma condição de filtragem da informação, (*iii*) o tráfego esperado, usando o modelo de balde de testemunhos e, (*iv*) opcionalmente, a latência máxima de propagação de cada evento. Esta informação deverá ser interpretada pelo sistema Publicação-Subscrição de forma a efectuar não só as correspondências de informação, como as reservas de recursos necessárias para o bom funcionamento das aplicações. Algumas subscrições podem não ser aceites por falta de recursos disponíveis. As reservas de recursos devem ser efectuadas pelo sistema de forma transparente para a aplicação.

Capítulo 3

Arquitectura IndiQoS

Este capítulo apresenta a arquitectura *IndiQoS*, que integra a definição de parâmetros de QdS num sistema Publicação-Subscrição. Esta arquitectura foi concebida tendo em conta requisitos como a escalabilidade e modularidade do sistema, assim como a expressão de parâmetros de QdS e transparência de reserva de recursos por parte das aplicações.

De uma forma geral, a arquitectura aqui apresentada é semelhante a um sistema Publicação-Subscrição como o Hermes (Pietzuch & Bacon, 2002), tendo sido enriquecida com mecanismos de reserva de recursos. A arquitectura é constituída por dois tipos de aplicações, os Editores e os Assinantes, e por um Serviço de Eventos. O Serviço de Eventos é constituído por um conjunto de encaminhadores que se interligam através de uma rede de nós *entre-pares*. A API definida para as aplicações foi enriquecida com uma forma de expressão genérica de parâmetros de QdS. As reservas de recursos são efectuadas pelo sistema, de uma forma transparente para a aplicação. A arquitectura foi dividida em vários módulos (ou camadas), o que facilita a sua reutilização em várias aplicações. O encaminhamento de dados é efectuado pelo sistema e tem em conta os requisitos de QdS

especificados pelas aplicações.

As Secções seguintes descrevem com pormenor quer os requisitos tidos em conta na concepção da arquitectura, quer os vários componentes da mesma. É depois definido como interagem estes componentes e quais são os mecanismos usados pelo sistema para definir os caminhos entre Editores e Assinantes.

3.1 Requisitos da arquitectura

Na concepção da arquitectura foram considerados os seguintes aspectos: (*i*) a escalabilidade do sistema, (*ii*) a modularidade do sistema, (*iii*) a expressão de parâmetros de QdS e (*iv*) a transparência na reserva de recursos para as aplicações. De seguida serão descritos estes requisitos com mais pormenor, confrontando-os com as características dos sistemas Publicação-Subscrição.

3.1.1 Escalabilidade do sistema

Como discutido no Capítulo 2, existem várias formas de construir sistemas Publicação-Subscrição, nomeadamente sistemas sem mediador, sistemas com mediador centralizado ou sistemas com mediador distribuído. Os sistemas sem mediador não têm todas as propriedades de um sistema Publicação-Subscrição, ficando assim excluídos do leque de opções existentes. Para sistemas de larga escala quer em termos de número de subscrições, quer em termos de número de Editores e Assinantes, um mediador centralizado consiste num ponto de congestão para o sistema, o que o torna limitado em termos de escalabilidade. Por este motivo, na definição da arquitectura *IndiQoS* é necessário ter em conta arquitecturas descentra-

lizadas, onde o mediador se encontra distribuído numa rede de encaminhadores.

Dentro dos sistemas com mediador distribuído existentes podem-se destacar dois: o SIENA e o *Hermes*. Estes sistemas foram já referidos na Secção 2.1.4. Em termos de escalabilidade, a maior diferença entre estes dois sistemas consiste na informação que é mantida pelos encaminhadores que fazem parte do Serviço de Eventos. No SIENA, a informação para manter todo o sistema encontra-se replicada pelos servidores. Uma vez que todos os anúncios terão de ser difundidos por todos os encaminhadores de eventos do sistema, este sistema revela também algumas limitações à capacidade de escala. A informação que é necessária guardar sobre os anúncios e subscrições que o sistema suporta terá de ser replicada pelos servidores, o que significa que o sistema não é escalável no número de anúncios e subscrições a armazenar. Note-se que para reduzir a dimensão do problema é possível efectuar fusões entre anúncios (ou subscrições), sempre que haja uma relação de inclusão entre estes.

O *Hermes*, por outro lado concretiza a noção de *Pontos de Contacto*. Um Ponto de Contacto é responsável por armazenar a informação sobre os anúncios e subscrições de um determinado tipo de eventos. Este Ponto de Contacto é atribuído de forma determinista por uma *Tabela de Dispersão Distribuída*[1] (TDD). A informação sobre diferentes tipos de eventos deve encontrar-se armazenada em diferentes servidores, ficando assim a informação distribuída pelos vários servidores do sistema. Por outro lado, ficando a informação sobre um tipo de eventos armazenada apenas num servidor, este constitui um ponto de falha para aquele tipo de informação. Para colmatar esta lacuna, o sistema pode replicar esta informação por

[1] Expressão traduzida do inglês *Distributed Hash Table*.

dois ou mais servidores e usar vários Pontos de Contacto para o mesmo tipo de informação. Ainda assim, a informação a guardar por um servidor no *Hermes* é sempre bastante mais reduzida do que a informação a guardar por um servidor do SIENA. As mensagens de controlo necessárias para manter a informação replicada também é mais reduzida. Assim, o sistema *Hermes* mostra mais potencialidades de escala no número de anúncios e subscrições. A arquitectura aqui descrita é, assim, baseada na noção de *Pontos de Contacto*.

3.1.2 Modularidade do sistema

Para tornar o sistema mais modular é necessário criar interfaces bem definidas entre os vários componentes do sistema, de forma a que possam ser interligados mas, por outro lado, construídos e modificados de forma independente. Nesta arquitectura foram definidos dois pontos onde imerge a necessidade de separação de componentes: (*i*) a construção do sistema *IndiQoS* de forma independente das aplicações e (*ii*) a construção do mesmo sistema de forma independente da rede de dados e da solução usada para obter garantias de QdS. Para que o sistema seja construído de forma independente das aplicações que o vão usar, é necessário definir uma interface do sistema para a aplicação (API) e uma interface da solução de rede para o sistema. A API usada pelas aplicações é definida na Secção 3.2.1, mais à frente neste capítulo e integra e definição dos requisitos da QdS pretendida. Para separar o sistema *IndiQoS* da rede de dados são usados componentes de rede genéricos, denominados *Infopipes*. Estes componentes são descritos também em pormenor mais à frente nesta dissertação.

3.1.3 Expressão de parâmetros de QdS

Outro dos aspectos chave a considerar no desenvolvimento da arquitectura *IndiQoS* é o conjunto de parâmetros de QdS a suportar e a forma como estes devem ser expressados pelas aplicações. De entre o conjunto de parâmetros possível, a versão do sistema *IndiQoS* apresentada nesta dissertação, garante a largura de banda e a latência, devido à sua simplicidade e ao suporte oferecido pelas redes de dados subjacentes, mas numa fase posterior, o *IndiQoS* poderá suportar mais parâmetros de QdS, como o *jitter*, a disponibilidade ou a taxa de perdas.

As aplicações usam o modelo do *Balde de Testemunhos* para expressar as suas necessidades de QdS. Como descrito no Capítulo 2, este modelo permite um tratamento matemático relativamente simples por parte do sistema, permitindo simultaneamente a geração de padrões de tráfego suficientemente complexas por parte das aplicações. Assim, para caracterizar o volume de tráfego de dados usando este modelo, a aplicação terá de definir os valores apropriados para a taxa e a capacidade do Balde de Testemunhos, de forma a que todos os eventos sejam enviados, satisfazendo os seus requisitos. No caso do Assinante desejar também uma latência mínima no envio dos eventos, esta terá de ser expressa à parte. Desta forma, a QdS expressa pelas aplicações é constituída por (*i*) um volume de tráfego definido por parâmetros do modelo de Balde de testemunhos e (*ii*), opcionalmente, a latência.

3.1.4 Transparência na reserva de recursos para as aplicações

Para que seja garantida a QdS requerida pelas aplicações, é necessário efectuar reserva de recursos. Estas reservas devem ser efectuadas pelo

sistema, de forma a libertar as aplicações desta tarefa. As aplicações têm apenas de indicar ao sistema qual a QdS pretendida, efectuando-o como descrito na secção anterior. Com a informação sobre a QdS pretendida pelas aplicações e sobre o tipo de eventos, o sistema *IndiQoS* escolhe um caminho que tem a QdS necessária (se este existir) e efectua as respectivas reservas. Depois de escolhido o caminho e efectuadas as reservas de recursos, a aplicação é avisada do sucesso (ou insucesso caso o caminho não exista).

Mais precisamente, o Serviço de Eventos da arquitectura *IndiQoS* é constituído por uma rede de encaminhadores que interliga Editores e Assinantes. Para além de efectuar o encaminhamento dos eventos disseminados pelos Editores, o sistema *IndiQoS* tem como função definir caminhos entre os Editores e os Assinantes, cujo processamento só pode ser efectuado tendo conhecimento sobre (*i*) a rede lógica definida sobre a rede IP, (*ii*) as publicações e as subscrições que incluem os recursos de QdS necessários e (*iii*) os recursos do sistema ainda disponíveis. Assim, uma das funcionalidades essenciais do sistema é o de determinar as árvores de distribuição óptimas para as notificações dos Editores, sendo os nós da rede os únicos pontos de bifurcação possível. A dificuldade desta tarefa reside no facto de os percursos óptimos dependerem dos recursos pedidos pelos assinantes e pelos recursos ainda disponíveis em cada encaminhador de eventos.

3.2 API do sistema *IndiQoS*

Como em qualquer sistema Publicação-Subscrição, no *IndiQoS* existem dois tipos de aplicações: os Editores e os Assinantes. Os Assinantes subs-

crevem um determinado tipo de informação anunciada pelos Editores. Para tal, as aplicações usam uma interface exportada pelo sistema *IndiQoS*. Esta interface está descrita nas Secções seguintes e foi definida usando a linguagem Java.

3.2.1 Definição de QdS pretendida

Para definir a QdS necessária de uma forma genérica e normalizada, foram criadas as classes `TokenBucket` e `QoS` que serão usadas pelas aplicações e interpretadas pelo sistema *IndiQoS*. Estas classes estão descritas nas Listagens 3.1 e 3.2.

Listagem 3.1: Classe TokenBucket.

```
package api.qos;

public class TokenBucket {
        public TokenBucket(double r, long b){}
        public long getCapacity () {}
        public double getRate () {}
        public void setCapacity (long capacity ) {}
        public void setRate (double rate ) {}
        public double getBandWidth(){}
}
```

Listagem 3.2: Classe QoS.

```
package api.qos;

public class QoS {
        public QoS(TokenBucket tb, long latency ){}
        public QoS(TokenBucket tb){}
        public long getLatency () {}
        public boolean isLatencyRequired () {}
        public TokenBucket getTokenBucket() {}
        public void setLatency (long latency ) {}
        public void setTokenBucket(TokenBucket tokenBucket) {}
        public double getBandWidth(){}
        public boolean fitsIn (QoS qos){}
}
```

3.2.2 Eventos da aplicação

Para definir um novo tipo de informação, deverá ser criado um evento que estenda a classe abstracta `IndiQoSEvent` definida na Listagem 3.3, i.e., todos os eventos da aplicação que circulem na rede deverão ser derivados da classe base `IndiQoSEvent`. Esta classe dispõe de métodos para serializar os dados do evento, métodos estes que devem ser redefinidos pelas classes que a estendem. Como ilustrado na Listagem 3.3, esta classe dispõe de métodos para devolver *Chaves* que serão usadas por uma Tabela de Dispersão Distribuída (TDD). Estes métodos serão usados pelo sistema *IndiQoS*, como será descrito mais à frente nesta dissertação.

A classe `IndiQoSEvent` dispõe também um método que devolve o tipo de eventos, sendo este tipo representado por uma classe criada para o efeito (Listagem 3.4).

Listagem 3.3: Classe que define um evento.

```
package api;

public abstract class IndiQoSEvent implements QuickSerializable {
        public IndiQoSEvent(int n_keys){}
        public IndiQoSEvent(InputBuffer buffer ){}
        public void serialize (OutputBuffer buffer ) {}
        public static BigInteger [] createKeys(IndiQoSEventType et, int n_keys){}
        public static IndiQoSEventType getEventType(Class clazz ){}
        public abstract double getSize ();

}
```

3.2.3 API para o Editor

A interface `Advertisement` apresentada na Listagem 3.5 deverá ser utilizada pelos Editores para anunciar e posteriormente publicar a informação de um determinado tipo. Através desta API, o Editor anuncia que

Listagem 3.4: Classe que define um tipo.

```
package api;

public class IndiQoSEventType {
        private Class eventType;
        public IndiQoSEventType(Class et ) {
                eventType = et;
        }
}
```

pretende divulgar informação de um determinado tipo T e que vai usar um determinado volume de tráfego modelado por um *Balde de Testemunhos*. Para tal, o Editor cria um objecto do tipo `Advertisement`, indicando o tipo de eventos e o balde de testemunhos. Antes de começar a disseminar eventos daquele tipo, a aplicação terá de invocar o método `advertise()`. Caso a operação termine com sucesso[2] o Editor pode começar a publicar eventos. Para tal, vai usar o método `publish(IndiQoSEvent e)`. Para indicar ao sistema que vai parar de enviar informação daquele tipo, a aplicação usa o método `unadvertise()`. Desta forma, o sistema *IndiQoS* pode libertar recursos que deixaram já de ser necessários.

3.2.4 API para o Assinante

O Assinante, por outro lado, tem uma API para subscrever e receber um determinado tipo de informação. A subscrição é efectuada usando o método `activate()` da interface `Subscription`, apresentada na Listagem 3.6. Quando o Assinante deseja parar de receber informação, basta invocar o método `deactivate()`. Sendo a subscrição de eventos efectuada por tipo com a possibilidade de efectuar filtragem de eventos, aquando a criação da subscrição, é também especificado um filtro para a informação. Na

[2] Ou seja, não é lançada a Excepção `CouldNotAdvertise`.

Listagem 3.5: Interface usada pelo Editor.

```
package api;

public interface Advertisement {
        public abstract void advertise ()
                throws CouldNotAdvertise;
        public abstract void unadvertise ();
        public abstract IndiQoSEventType getEventType();
        public abstract TokenBucket getTokenBucket();
        public abstract void setTokenBucket(TokenBucket tokenBucket);
        public abstract void publish(IndiQoSEvent event)
                throws CouldNotPublishException;
}

public class CouldNotAdvertise extends Exception {
        public CouldNotAdvertise() {
                super();
        }
        public CouldNotAdvertise(String message) {
                super(message);
        }
}

public class CouldNotPublishException extends Exception {
        public CouldNotPublishException() {
                super();
        }
        public CouldNotPublishException(String message) {
                super(message);
        }
}
```

subscrição é também especificado um objecto onde serão recebidas as notificações de forma assíncrona. Estes dois últimos objectos são uma concretização das interfaces `Filter` (Listagem 3.7) e `Notifiable` (Listagem 3.8) e são definidas pela aplicação. O assinante recebe assincronamente os eventos de tipo que subscreveu no método `handler()` da classe que concretiza a interface `Notifiable`.

Listagem 3.6: Interface usada pelo Assinante.

```
package api;

public interface Subscription {
        public void activate ()
                throws CouldNotActivateSubscription ;
        public void deactivate ();
        public IndiQoSEventType getEventType();
        public Filter getFilter ();
        public Notifiable getNotifiable ();
        public QoS getQos();
        public void setFilter ( Filter filter );
        public void setNotifiable ( Notifiable notifiable );
        public void setQos(QoS qos);
}

public class CouldNotActivateSubscription
        extends Exception {
        public CouldNotActivateSubscription () {
                super();
        }
        public CouldNotActivateSubscription ( String message) {
                super(message);
        }
}
```

Listagem 3.7: Interface de filtragem de eventos.

```
package api;

public interface Filter {
        public boolean testFilter (IndiQoSEvent event );
}
```

3.2.5 Exemplo

Como exemplo de utilização da API definida, tomemos o caso do sistema monitor de temperaturas das salas de um edifício do capítulo anterior. Para tal é necessário definir um evento do tipo `Temp`. Este evento terá de estender a classe `IndiQoSEvent` com os atributos `sala`, `temperatura` e `precisao`. A Listagem 3.9 ilustra o código a efectuar para criar o novo tipo `Temp`.

Depois de definido o novo tipo, é possível criar um Editor. No exemplo definido na Listagem 3.10, o Editor é criado indicando que vai disse-

Listagem 3.8: Interface do objecto receptor de eventos.

```
package api;

public interface Notifiable {
        public void handler(IndiQoSEvent e);
}
```

Listagem 3.9: Evento de exemplo.

```
public class Temp extends IndiQoSEvent {
        public String sala;
        public int temperatura;
        public double precisao;
        public Temp(String sala, int temperatura, double precisao) {
                super();
                this.sala = sala;
                this.temperatura = temperatura;
                this.precisao = precisao;
        }
        public double getSize() {
                return sala.length() + 4 + 8;
        }
}
```

minar eventos do tipo `Temp` com um volume de tráfego definido por um balde de testemunhos (classe `TokenBucket`). Depois de criado, o Editor indica ao sistema que deseja começar a enviar eventos (linha 8 da Listagem 3.10). Se a operação tiver sucesso, o editor pode começar a criar eventos e a dissemina-los pelo sistema, usando o método `publish()` (linha 18 da Listagem 3.10).

O Assinante, por outro lado, necessita antes de implementar duas interfaces: a interface `Filter` e a interface `Notifiable`. Uma concretização simplificada destas duas interfaces, assim como uma possível concretização de um Assinante estão exemplificados na Listagem 3.11. Depois de concretizadas as interfaces necessárias, é possível criar o Assinante. Assim, no momento da sua criação, o Assinante deve indicar qual o tipo de eventos que deseja receber, o filtro, a classe receptora dos eventos e a QdS

Listagem 3.10: Exemplo de um Editor.

```
public void runAdvertiser (){
        IndiQoSEventType eventType = IndiQoSEvent.getEventType(Temp.class);
        TokenBucket tokenBucket = new TokenBucket(10, 0);
        Advertisement advertisement =
                new AdvertisementImpl(eventType,tokenBucket);

        try {
                advertisement . advertise ();
        } catch (CouldNotAdvertise ex) {
                System.err . println ("Could␣not␣advertise."+
                                                "␣Reason:␣" + ex.getMessage());
                return;
        }

        Temp e = new Temp(1,"Sala1" ,16, 0.01);

        try {
                advertisement . publish(e);
        } catch (CouldNotPublishException ex1) {
                System.err . println ("Could␣not␣publish␣event."+
                                                "␣Reason:␣" + ex1.getMessage());
        }

        advertisement . unadvertise ();
}
```

requerida para esta subscrição. Depois deverá indicar ao sistema que deseja começar a receber eventos. Para isso invoca a operação `activate()` (linha 29 da Listagem 3.11) e o sistema irá definir um caminho até ao Editor, se existir. Depois, o Assinante pode continuar a efectuar outro processamento, já que as notificações são recebidas de forma assíncrona. Quando o Assinante deseja parar de receber eventos apenas tem de invocar a operação `deactivate()` (linha 38 da Listagem 3.11).

Através desta API, quer o Editor quer o Assinante interagem com o sistema *IndiQoS*, indicando a informação que este necessita para ligar os Assinantes aos Editores, escolhendo caminhos de forma a fornecer os recursos necessários às aplicações.

Listagem 3.11: Exemplo de um Assinante.

```
class TempFilter implements Filter {
        public boolean testFilter (IndiQoSEvent event) {
                if (event instanceof Temp){
                        Temp t = (Temp) event;
                        return (t.temperatura > 60);
                }
                return false;
        }
}
class TempNotifiable implements Notifiable {
        public void handler(IndiQoSEvent e) {
                if (e instanceof Temp){
                        System.out.println ("May␣day!␣May␣day!"+
                                "Temperature␣of␣room␣"+((Temp)e).sala +
                                "␣reached␣"+((Temp)e).temperatura);
                }
                else System.err.println ("Event␣not␣for␣me...");
        }
}
public void runSubscriber(){
        IndiQoSEventType eventType = IndiQoSEvent.getEventType(Temp.class);
        TokenBucket tokenBucket = new TokenBucket(10, 0);
        QoS qos = new QoS(tokenBucket,10);
        Filter filter = new TempFilter();
        Notifiable notifiable = new TempNotifiable();
        Subscription subscription =
                new SubscriptionImpl(eventType, filter , notifiable ,qos);
        try {
                subscription.activate ();
        } catch ( CouldNotActivateSubscription ex) {
                System.err.println ("Could␣not␣publish␣event."+
                                                "␣Reason:␣" + ex.getMessage());
                return;
        }
        /*
         * faz outro processamento ...
         */
        subscription.deactivate ();
}
```

3.3 Componentes da arquitectura *IndiQoS*

Tendo em conta os requisitos definidos anteriormente, a arquitectura *IndiQoS* é composta por dois tipos de aplicações (Editores e Assinantes) interligadas por um Serviço de Eventos que se encontra distribuído por vários nós de uma rede *entre-pares*. Neste capítulo irão ser descritos em pormenor estes componentes, indicando (*i*) como interagem, (*ii*) como é efectuada a escolha dos caminhos entre os Editores e os Assinantes e a res-

pectiva reserva de recursos e (*iii*) as funcionalidades usadas para se obter uma optimização dos recursos usados.

O Serviço de Eventos é o componente que interliga os Editores aos Assinantes. Este componente é constituído por um conjunto de encaminhadores interligados por uma rede de nós entre-pares, como ilustrado na Figura 3.1. Estes encaminhadores cooperam entre si para formar árvores de disseminação de eventos que garantem os requisitos de QdS pedidos pelas aplicações. A rede entre-pares efectua o encaminhamento das mensagens usando uma *Tabela de Dispersão Distribuída* (TDD). A TDD usada na corrente concretização do sistema *IndiQoS* consiste no *Bamboo* (Rhea *et al.*, 2003). O Bamboo é uma TDD concretizada sobre uma moldura de composição de protocolos que comunica por eventos. Para além de ter uma noção de *proximidade* entre vizinhos baseada na latência das ligações, o Bamboo dispõe da opção de activação de um mecanismo para optimização dos caminhos entre vizinhos. Este mecanismo consiste apenas em mudanças das ligações entre vizinhos de forma a optimizar o estado global da rede entre-pares.

Usando a noção de *Pontos de Contacto* definida no Capítulo 2, a cada tipo de evento é deterministicamente associada uma chave. A cada chave está associado um nó da rede. Desta forma existe um nó na rede que é o responsável por manter a informação sobre um determinado tipo de eventos: o *Nó de Contacto*. O endereço (ou chave) deste Nó é descoberto localmente, usando um algoritmo determinista que se baseia no tipo de evento. No Bamboo, para efectuar encaminhamento de mensagens basta enviar um pedido com uma determinada chave. Assim, de uma forma geral, o Editor começa por se registar no Nó de Contacto daquele tipo de evento. O Assinante que deseja assinar o mesmo tipo de informação, por sua vez,

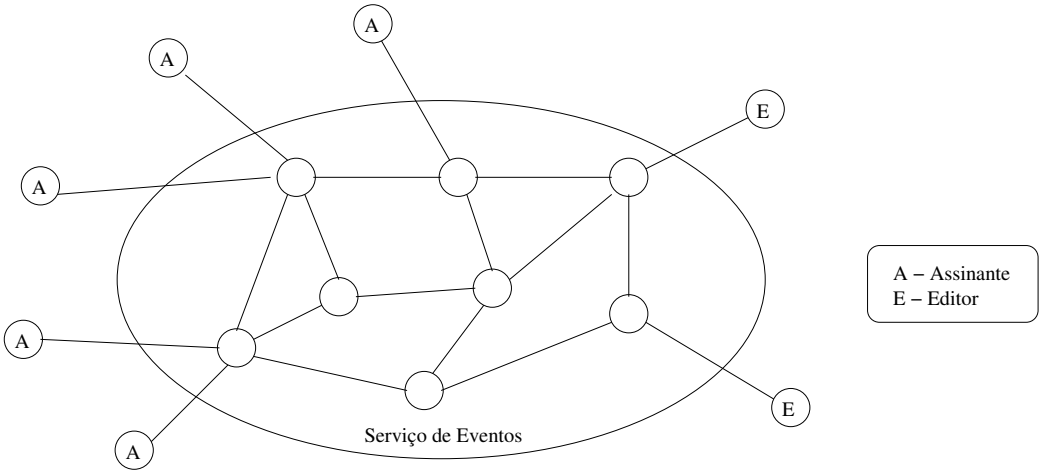


Figura 3.1: Visão geral da arquitectura *IndiQoS*.

regista-se também no mesmo Nó de Contacto. Esse Nó de Contacto, perante o registo do Editor e do Assinante e caso a QdS pedida possa ser satisfeita, começa a reencaminhar os eventos que chegam do Editor para o Assinante. É desta forma que são efectuados os anúncios e subscrições.

O uso dos Pontos de Contacto para interligar os Editores aos Assinantes tem três aspectos negativos: (*i*) o Ponto de Contacto pode ser um ponto de congestão para um determinado tipo de dados, (*ii*) os caminhos escolhidos podem não ser os óptimos, já que o caminho teria sempre de passar pelo Ponto de Contacto e (*iii*) a informação sobre um tipo de informação está guardada apenas num encaminhador, o que pode significar um ponto de falha do sistema. Para resolver estes problemas são usados dois mecanismos adicionais: o uso de vários Pontos de Contacto para um tipo de dados e a intercepção de pedidos por parte de nós intermédios que dispõem já de ligações que satisfazem o pedido. Estes mecanismos serão explicados em pormenor mais à frente neste capítulo.

Os clientes interagem com o Serviço de Eventos usando uma interface

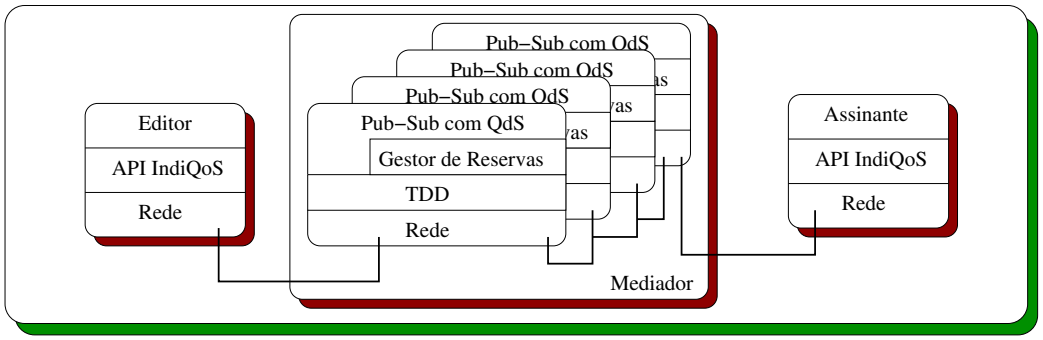


Figura 3.2: Interacção entre os componentes do sistema *IndiQoS*.

bastante simples, que esconde da aplicação todos estes pormenores. Esta interface está definida na Secção 3.2 deste capítulo. A interface é criada no processo do cliente e usa um intermediário para se ligar a um dos nós da rede entre pares. Os nós da rede entre pares interagem entre si usando a TDD. Como ilustrado na Figura 3.2, o Editor comunica com o Serviço de Eventos usando um dos nós da rede entre-pares, rede esta que é responsável por definir o caminho até aos nós que estão ligados aos Assinantes. Uma vez chegado ao nó de ligação da rede entre-pares, o pedido da aplicação (que pode ser um anúncio, uma subscrição ou um evento a ser disseminado) é encaminhado pela TDD pelos nós da rede até chegar ao seu destino.

Desta forma, a arquitectura *IndiQoS* está dividida em vários componentes, que efectuam diferentes funções. Estes componentes são resumidos de seguida.

Publicação-Subscrição sobre uma TDD No sistema *IndiQoS* o encaminhamento é efectuado por uma TDD. A TDD limita-se a receber pedidos para encaminhar uma mensagem até ao nó associado a uma determinada chave. Durante o encaminhamento, os nós intermédios

do caminho de uma determinada mensagem são informados sobre a passagem da mensagem. Esta funcionalidade dá aos nós hipótese de acrescentar informação à mensagem (e.g. sobre a QdS existente nas ligações) ou até para a interceptar caso exista já uma subscrição que passa pelo nó corrente e que satisfaz o pedido em questão. A TDD é responsável por definir os caminhos entre o Editor e os Assinantes. Os Pontos de Contacto são responsáveis por manter informação dos tipos de eventos que estão a ser disseminados pelo sistema e gerir os recursos necessários. Para efectuar a gestão de recursos da rede, o componente de Publicação-Subscrição de cada Nó da rede entre-pares dispõe de um subcomponente especifico que se destina apenas a esta tarefa, denominado *Gestor de Recursos*.

Rede de dados com QdS Na concretização da rede de dados são usados componentes genéricos, descritos mais à frente neste capítulo. Estes componentes devem ser concretizados usando arquitecturas como os Serviços Integrados ou os Serviços Diferenciados, descritas na Secção 2.2 desta dissertação. Estas arquitecturas fornecem mecanismos de reserva de recursos na rede de dados. Os componentes genéricos de rede devem fornecer uma interface para as camadas acima, de forma a fornecer informação sobre os recursos existentes na rede de dados.

Gestão de recursos A gestão dos recursos de rede necessários para manter a QdS pedida pelas aplicações é efectuada por um subcomponente da camada de Publicação-Subscrição, o *Gestor de Recursos*. Este componente encontra-se naturalmente distribuído por todos os nós da rede entre-pares e necessita de manter informação sobre quais as li-

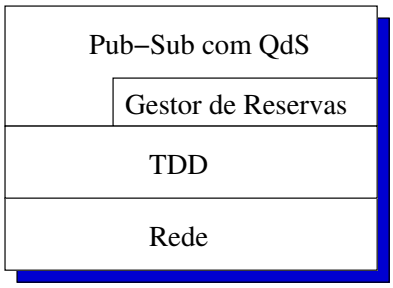


Figura 3.3: Pilha protocolar dos encaminhadores *IndiQoS*.

gações entre o seu nó e os respectivos vizinhos. Sobre cada ligação é necessário obter a quantidade de recursos disponibilizada pela camada de rede. Depois de obter esta informação, é este componente que deve efectuar a gestão dos recursos, recebendo pedidos de reserva e libertação dos mesmos para cada ligação da camada Publicação-Subscrição. Desta forma é diminuída a quantidade de mensagens de sinalização do protocolo que concretiza os componentes de rede genéricos.

Para concretizar os componentes da arquitectura *IndiQoS*, cada nó da rede entre-pares pode ser definido usando a pilha de protocolos ilustrada na Figura 3.3 e é constituída por (*i*) uma camada de Publicação-Subscrição, que tem como função gerir as publicações, as subscrições e os recursos reservados para garantir a QdS, (*ii*) uma camada de distribuição da Tabela de Dispersão, que garante o encaminhamento das mensagens e (*iii*) a camada de rede. Nas seguintes Secções serão discutidos em pormenor cada um dos componentes da arquitectura apresentada.

3.3.1 Bamboo: uma Tabela de Dispersão Distribuída

Uma *Tabela de Dispersão Distribuída* (TDD) permite a gestão distribuída de uma projecção de chaves para valores. No sistema *IndiQoS* é usada a TDD Bamboo (Rhea *et al.*, 2003). O Bamboo é uma TDD baseada no sistema *Pastry* (Rowstron & Druschel, 2001). A cada nó Bamboo é atribuído um identificador numérico que consiste no resultado de uma função de dispersão aplicada ao par "endereço IP e porto" onde o nó está a receber os pacotes de dados.

Uma das características do Bamboo reside no facto de ter mecanismos que lhe permite comportar-se bem em níveis altos de entrada e saída de nós na rede. O uso da rede na troca de mensagens de controlo, que servem para manter a informação correcta na tabela de encaminhamento, cresce de forma logarítmica.

Pode-se usar esta TDD de duas formas distintas. Uma delas consiste em usar uma interface de `get()` e `put()` equivalente às tabelas de dispersão convencionais. Para uma aplicação construída sobre o Bamboo que usa esta interface, os valores são guardados num dos encaminhadores da rede entre-pares de uma forma opaca para as aplicações da TDD. Outra forma de usar a TDD Bamboo consiste em usar a seguinte interface:

RouteInit Indica ao Bamboo para iniciar o encaminhamento de uma mensagem com uma determinada chave;

RouteUpcall O Bamboo indica à aplicação que uma mensagem está a passar por aquele nó, mas que não é o nó de destino;

RouteContinue A aplicação indica ao Bamboo para que continue o encaminhamento da mensagem;

RouteDeliver Entrega de uma mensagem encaminhada pelo Bamboo ao nó de destino.

Esta é a interface usada pelo sistema *IndiQoS*, já que é necessário interceptar os pedidos das aplicações nos nós intermédios. Todas as mensagens de controlo percorrem o caminho definido pela TDD, notificando cada um dos nós sobre a passagem de cada mensagem.

Cada nó mantém uma tabela de encaminhamento e escolhe os seus vizinhos baseando-se em dois critérios: (*i*) a similaridade do identificador do candidato com o seu e (*ii*) a proximidade na rede em termos de latência. Para ter informação actualizada, a tabela de encaminhamento é mantida através de dois algoritmos:

Algoritmo de afinação global Este algoritmo corre em todos os nós da rede entre-pares e faz com que o estado global da rede estabilize num ponto óptimo. Esta função comporta-se melhor em redes mais estáticas, mas é mais lento relativamente ao algoritmo de afinação local.

Algoritmo de afinação local Este algoritmo é também baseado no *Pastry*, onde periodicamente escolhe um nó da sua tabela de encaminhamento e pede informação sobre os vizinhos desse nó para preencher a sua própria tabela.

Para além dos algoritmos existentes para manter a tabela de encaminhamento em cada nó, o Bamboo dispõe também de um algoritmo de entrada de novos nós. Este algoritmo também é baseado no *Pastry*. Para se juntar ao sistema, um novo nó precisa do endereço de um nó que pertença já ao sistema. Com esse endereço, tem acesso a um dos nós do sistema. Esse nó informa o nó que deseja entrar sobre um vizinho próximo dele. O nó que deseja entrar, comunica com esse vizinho e começa a preencher

a sua própria tabela de encaminhamento. Através do algoritmo de afinação local, este novo nó preenche a sua tabela de encaminhamento rapidamente, afinando depois as suas ligações com o algoritmo de afinação global.

3.3.2 Publicação-Subscrição numa Tabela de Dispersão Distribuída

Baseando-se na arquitectura definida no sistema *Hermes*[3], o *IndiQoS* dispõe de um Serviço de Eventos distribuído e descentralizado. Os encaminhadores do Serviço de Eventos encontram-se sobre uma rede de nós entre-pares que é definida pela TDD descrita anteriormente: o *Bamboo*. Também como no sistema *Hermes*, o *IndiQoS* usa a noção de *Pontos de Contacto*. Os Pontos de Contacto são responsáveis por guardar informação sobre um determinado tipo de eventos.

Aos nós da rede entre-pares podem ligar-se as aplicações. Os nós que interagem com as aplicações recebem os seus pedidos e devolvem as respectivas respostas. Os pedidos das aplicações podem ser os seguintes:

Anúncio do Editor O Editor anuncia ao sistema que vai iniciar o envio de um determinado tipo de eventos. O sistema terá de registar o Editor no Ponto de Contacto correspondente ao tipo que é anunciado e efectuar as respectivas reservas no caminho até ao Ponto de Contacto;

Remoção de um anúncio do Editor O Editor anuncia ao sistema que vai parar o envio de um determinado tipo de eventos. O sistema deve remover o registo do Ponto de Contacto e libertar os recursos que foram previamente reservados;

[3] Ver Secção 2.1.4 do capítulo anterior.

Subscrição do Assinante O Assinante subscreve um tipo de eventos indicando esta informação ao sistema. O sistema terá de registar o Assinante no Ponto de Contacto correspondente ao tipo que é subscrito, verificar se a QdS pedida pode ser satisfeita por algum Editor registado e efectuar as respectivas reservas no caminho até ao Ponto de Contacto;

Remoção de uma subscrição do Assinante O Assinante indica ao sistema que deseja parar de receber os eventos a que subscreveu. O sistema deve remover o registo do Ponto de Contacto e libertar os recursos que foram previamente reservados;

Publicação de um evento O sistema deve usar os caminhos já previamente definidos para encaminhar o evento publicado pelo Editor até ao(s) Assinante(s) que o subscreveram.

Para efectuar estas operações, é preciso definir caminhos entre o Editor e o(s) Assinante(s). Os caminhos são definidos usando os mecanismos de encaminhamento oferecidos pela TDD.

Os Pontos de Contacto também têm uma função importante neste aspecto. Sendo os Pontos de Contacto os elementos que têm conhecimento sobre os anúncios e subscrições existentes no sistema para aquele tipo de informação, o caminho entre um Editor e um Assinante vai necessariamente passar por ele. Como pode ser ilustrado na Figura 3.4, o Editor regista-se no Ponto de Contacto e é descoberto um caminho até ele. O mesmo é efectuado pelo Assinante. Desta forma já existe um caminho desde o Editor até ao Assinante que passa pelo Ponto de Contacto.

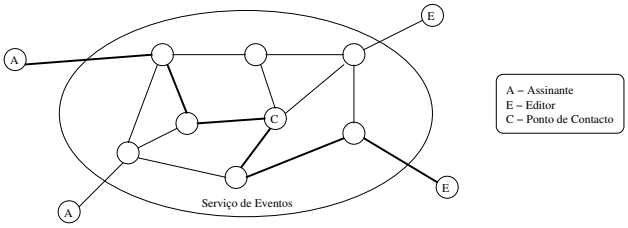


Figura 3.4: Caminho definido entre o Editor e o Assinante.

Algoritmo de descoberta das Chaves da TDD

Um dos pontos positivos na escolha de uma arquitectura que usa uma TDD, consiste na descoberta do nó na rede que é responsável por manter toda a informação sobre um determinado tipo de dados. Neste caso não se efectua explicitamente qualquer troca de mensagens para descobrir um determinado nó, nem se guarda qualquer informação sobre este assunto. Usam-se simplesmente as funcionalidades de uma TDD e esta encarrega-se de encaminhar as mensagens até ao ponto que é desejado. Para tal, basta enviar uma mensagem pela TDD que tem como endereço de destino uma determinada chave.

No caso de um sistema Publicação-Subscrição que usa a noção de Pontos de Contacto, interessa descobrir qual o nó responsável pela gestão de um determinado tipo de eventos. Para isso é usado um algoritmo muito simples que devolve uma chave diferente para cada tipo de dados. Este algoritmo baseia-se na chave de dispersão[4] da classe que representa um determinado tipo de eventos. Desta forma consegue-se obter uma distribuição de Pontos de Contacto pela rede entre-pares, distribuindo quer a carga na rede quer o estado que é necessário guardar sobre os anúncios e subscrições.

[4] A chave de dispersão consiste no valor devolvido pela função de dispersão (ou `hashCode()`) de uma determinada classe.

Árvores de distribuição de eventos

Até agora foi demonstrado como se efectua a ligação entre um Editor e um Assinante. Mas, num sistema Publicação-Subscrição normalmente existe um Editor a disseminar informação interessante para *n* Assinantes. Desta forma, o algoritmo de escolha de caminhos entre as aplicações do sistema deve formar uma árvore de distribuição de eventos, onde e Editor é a raíz dessa árvore e os Assinantes são as folhas. O Ponto de Contacto é o ponto de bifurcação dos eventos.

Construindo a árvore de disseminação de eventos desta forma, o Ponto de Contacto torna-se facilmente no ponto de congestão da rede, já que todos os eventos têm de passar por esse nó. Para aliviar o Ponto de Contacto de todo o trabalho, os outros nós do sistema podem interceptar um pedido do Assinante. Quando um pedido de subscrição de um assinante passa por um determinado nó da rede entre-pares, esse nó verifica se já existe alguma ligação que passa por esse nó e que satisfaz a subscrição. Se existir, é colocada uma bifurcação nesse nó para esse caminho, aliviando o Ponto de Contacto de algum trabalho.

Assim, com esta funcionalidade adicional, a árvore de distribuição de eventos é construída de forma distribuída, do seguinte modo: o Editor começa por se registar no Ponto de contacto. Para tal, envia uma mensagem pela TDD tendo em conta os requisitos de largura de banda disponível e minimizando a latência. Esta fase inicial está ilustrada na Figura 3.5.

Depois de registado o Editor, os Assinantes podem iniciar o seu processo de registo. À semelhança do Editor, os Assinantes enviam uma mensagem pela TDD até ao Ponto de Contacto. Os primeiros Assinantes reservam recursos num caminho até ao Ponto de Contacto. Mas, como ilustrado na Figura 3.6 à medida que outros Assinantes se vão registando

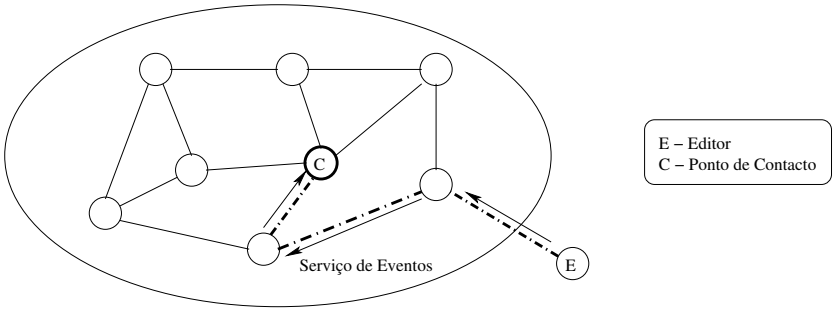


Figura 3.5: Caminho no Serviço de Eventos para o anúncio do Editor.

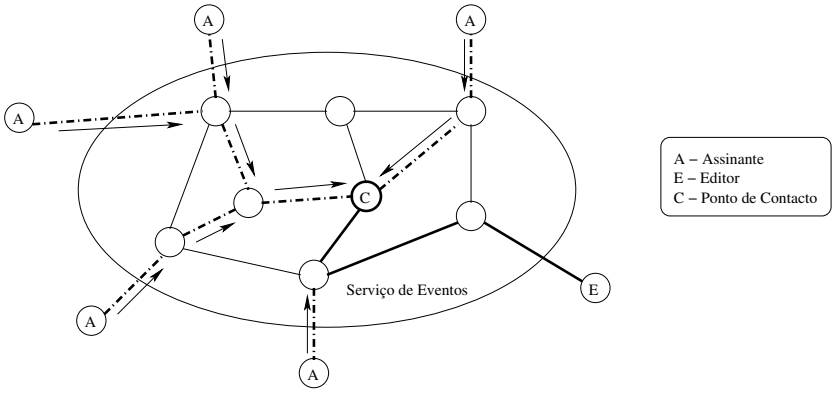


Figura 3.6: Caminhos no Serviço de Eventos para as subscrições dos Assinantes.

no sistema, alguns pontos de bifurcação vão sendo criados.

Depois de construída a árvore de disseminação de eventos, pode-se verificar que alguns pontos de bifurcação se encontram tipicamente mais perto dos Assinantes. Este comportamento do sistema faz com que alguns recursos sejam poupados, já que deixa de ser necessária a reserva de recursos de todos os Assinantes até ao Ponto de Contacto. Como se pode verificar na Figura 3.7, que ilustra a árvore final de disseminação de eventos, existe uma bifurcação de eventos não só no Ponto de Contacto C, mas também nos encaminhadores $R1$, $R3$ e $R4$.

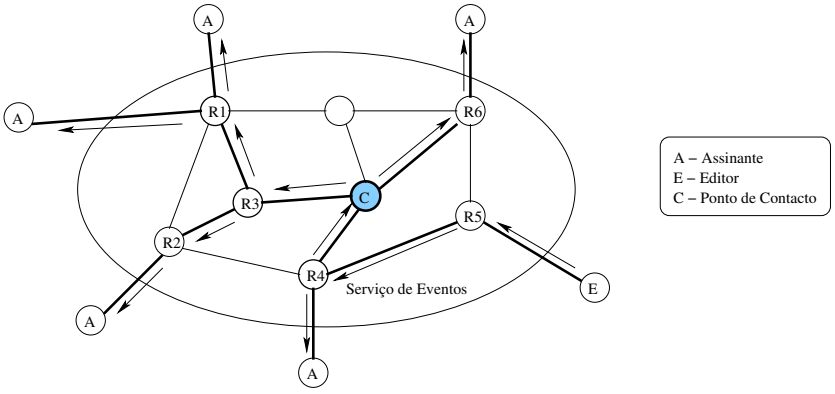


Figura 3.7: Árvore de distribuição de eventos de um determinado tipo.

3.3.3 Componentes abstractos de rede

Uma forma de tornar o sistema mais modular consiste em encapsular a concretização da QdS em componentes de software genéricos, que podem ser materializados usando diferentes soluções. Estes componentes, que serão descritos de seguida, são denominados *Infopipes* (Koster *et al.*, 2001) e, quando interligados entre si, possibilitam o fluxo de dados desde uma origem até determinado(s) destino(s). A utilização destes componentes esconde dos restantes níveis da arquitectura *IndiQoS* as particularidades de cada rede concreta, assegurando, não só uma maior modularidade, como a portabilidade das camadas superiores para um conjunto diversificado de redes de dados. Existem vários tipos de Infopipes, mas neste trabalho apenas vão ser tomados em conta os seguintes:

Netpipe Consiste num *Pipe* onde circula informação, em que a origem e o destino estão localizados em máquinas diferentes. É o responsável por fazer passar a informação pela rede e é sobre este Infopipe que vamos fazer incidir os parâmetros de QdS.

Source Estabelece uma ponte entre a origem da informação (emissor) e os

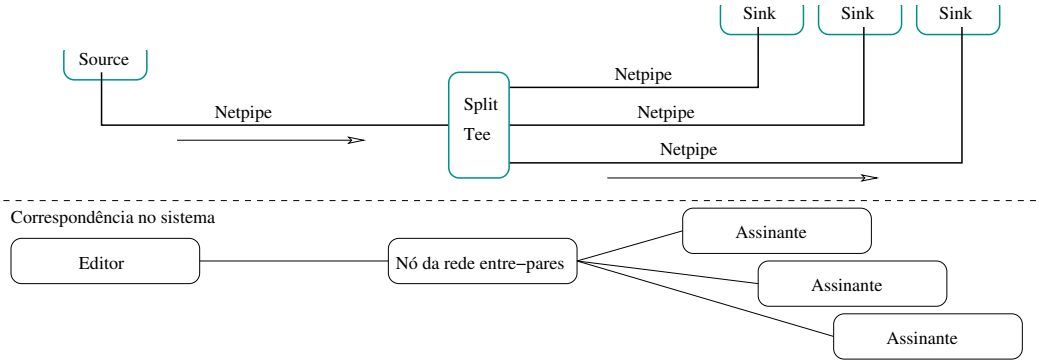


Figura 3.8: Exemplo de um Infopipeline com uma entrada (Source), um componente de difusão (Split Tee) e três saídas (Sink).

outros Infopipes. É através deste Infopipe que a informação vai ser enviada.

Sink Estabelece uma ponte entre os Infopipes e o destino dos items de informação (receptor). A informação colocada num *Source* atravessa o seu caminho e chega ao *Sink*, que entrega a informação à aplicação.

Split Tee É constituído por um ponto de entrada e *n* pontos de saída. Cada item de informação que entra neste Infopipe é replicado por todos os pontos de saída, concretizando assim a noção de *difusão*.

Um *Infopipeline* é formado interligando os Infopipes acima descritos. Como exemplificado na Figura 3.8, com Infopipelines é possível suportar fluxos de informação entre a origem e os respectivos destinos, formando uma árvore de disseminação de dados. Neste exemplo, os dados da aplicação são colocados no *Source* e são imediatamente enviados pelo Netpipe a que está ligado. Este Netpipe encaminha os dados até a um Nó intermédio, que tem um Infopipe de difusão (Split Tee). Neste Nó intermédio, os dados são reencaminhados por três Netpipes, que encaminham cada

um dos items de informação ao seu destino. A árvore de disseminação de dados pode também ser vista como uma rede virtual sobre a rede de dados subjacente que esconde as propriedades da rede de dados que não são relevantes para as restantes camadas da arquitectura *IndiQoS*.

Os Infopipes estão interligados entre si através de uma interface específica (Koster *et al.*, 2001). Para além da interface que permite interligar os Infopipes, estes exportam também uma interface para a aplicação, que possibilita a consulta/alteração de alguns parâmetros de cada Infopipe, destinada à aplicação que os usa. Para este trabalho foi efectuada uma concretização dos Infopipes anteriormente descritos. A concretização do Netpipe exporta uma interface para as aplicações que o criam para consulta e alteração dos parâmetros de QdS da rede de dados que esconde. Ao concretizar esta interface usando várias soluções de rede, a modularidade do sistema aumenta.

Assim, cada Nó da rede entre-pares interliga-se com os seus vizinhos usando um Netpipe, que interliga depois com outros Infopipes para formar os fluxos de informação. O Nó que cria o Netpipe passa a ter acesso à interface de consulta e reserva de recursos da rede.

3.3.4 QdS na arquitectura *IndiQoS*

A QdS da rede de dados é fornecida pela própria rede, que no sistema está representada pelos *Infopipes*. As reservas de recursos da rede são efectuadas pelo sistema *IndiQoS* usando a interface dos *Netpipes*. Os recursos são reservados por uma entidade denominada *Gestor de Recursos*, usando a interface dos *Infopipes*. Estes recursos são depois geridos internamente por este componente. É o Gestor de Recursos que efectua a reserva real dos recursos nos *Infopipes*, distribuindo-os depois pelas publicações e subscri-

ções que vão surgindo ao longo do tempo. Assim, quando uma nova publicação ou subscrição pede recursos ao Gestor de Recursos, este efectua uma reserva, atribuindo uma parte dos recursos, se existirem, ao pedido recebido.

Apesar de ser a TDD a encaminhar as mensagens até ao Ponto de Contacto, há aspectos sobre os parâmetros da QdS que são tidos em conta pelos Pontos de Contacto. Nesta versão inicial do sistema *IndiQoS* são garantidos dois parâmetros da QdS da rede: a latência e a largura de banda. Assim, no encaminhamento das mensagens, só são tidos em conta os caminhos que têm ainda largura de banda suficiente para satisfazer um determinado pedido e, dentro das ligações que têm largura de banda suficiente, é escolhida a ligação que tem uma menos latência.

As reservas no Gestor de Recursos são efectuadas localmente em cada encaminhador e só são pedidas quando os caminhos estão já definidos. Um cliente do sistema, depois de contactar o devido Ponto de Contacto, recebe uma resposta que contém já um caminho que satisfaz os requisitos de QdS. Com este caminho definido, basta enviar uma mensagem de controlo no sentido contrário para efectuar as respectivas reservas no Gestor de Recursos de cada encaminhador da rede entre-pares.

Como os caminhos se ligam no Ponto de Contacto, é este nó que tem a função de verificar se a QdS pedida pode ser satisfeita pelo sistema. Quando é efectuado o registo do Editor e do Assinante, o Ponto de Contacto dispõe da informação sobre a largura de banda reservada para o Editor, a largura de banda pedida pelo Assinante, as latências dos caminhos dele até às duas aplicações e a latência pedida pelo Assinante. Com esta informação, o Ponto de Contacto verifica se a QdS pode ser satisfeita. Caso existam vários Editores a publicar o mesmo tipo de informação mas com

diferentes QdSs, o Ponto de Contacto pode também escolher o Editor mais apto a satisfazer um pedido por parte do Assinante.

3.3.5 Sobre-reservas na rede de dados

Como foi já mencionado na secção anterior, as reservas de recursos necessárias para manter o sistema *IndiQoS* são efectuadas e geridas por um subcomponente do sistema que gere as publicações e subscrições, denominado *Gestor de Recursos*. O Gestor de Recursos usa a interface fornecida pelos *Infopipes* para reservar recursos na rede de dados.

Quando pretende efectuar uma reserva, o sistema Publicação-Subscrição usa o Gestor de Recursos. Desta forma, este componente consiste num intermediário de reserva e libertação de recursos, que se situa entre o sistema e a rede de dados. Este intermediário foi criado para facilitar e encapsular a gestão dos recursos reservados, mas também para se efectuarem sobre-reservas. O sistema *IndiQoS* pode, em cada ligação entre dois nós da rede entre-pares, pedir uma quantidade de recursos maior do que aquela que é realmente necessária num determinado momento. Esta quantidade é depois gerida internamente pelo Gestor de Recursos.

As sobre-reservas efectuadas pelo sistema *IndiQoS* têm o intuito de minimizar a quantidade de mensagens de sinalização que podem ocorrer nos protocolos de sinalização usados pelos *Infopipes*. Como demonstrado em (Sofia *et al.*, 2002a; Sofia *et al.*, 2003), a agregação de fluxos de dados com QdS pode aumentar a escalabilidade dos protocolos de reserva de recursos num sistema de larga escala. Ao invés de se efectuar um conjunto de reservas, uma para cada fluxo, efectua-se uma única reserva, agregando fluxos de informação que têm determinadas características.

3.3.6 Replicação dos Pontos de Contacto

Segundo a noção de Pontos de Contacto, um único nó é responsável por manter a informação sobre um determinado tipo de eventos. Apesar de tornar o sistema mais escalável, já que esta informação não necessita de ser replicada por todos os encaminhadores, o Ponto de Contacto consiste num ponto de falha do sistema *IndiQoS*: se um determinado nó está a ser usado como Ponto de Contacto de um tipo de eventos e esse nó falhar, o sistema deixa de conseguir fornecer o serviço para aquele tipo de eventos. Para resolver este problema podem ser acrescentados mais Pontos de Contacto para um único tipo de evento. Para associar *n* Pontos de Contacto a um único tipo de evento, basta modificar o algoritmo de descoberta de chaves para, perante um tipo de evento, devolver *n* chaves. Desta forma, o algoritmo de definição das árvores de distribuição tem de ser adaptado de forma a suportar *n* Pontos de Contacto por cada tipo de evento. O Editor, depois de encontrar as *n* chaves correspondentes ao tipo de evento, regista-se em todos os Pontos de Contacto. Os Assinantes, por outro lado efectuam um pedido aos *n* Pontos de Contacto, recebendo *n* respostas. Estas respostas são então avaliadas pelo Assinante, que escolhe o Ponto de Contacto mais indicado. Escolhido o Ponto de Contacto, o Assinante efectua o seu registo e é então ligado à árvore de disseminação.

A Figura 3.9 exemplifica uma árvore de distribuição para um determinado tipo de evento com dois Pontos de Contacto. Neste caso, a informação sobre os Editores existentes para um determinado tipo de dados encontra-se replicada em dois nós do sistema, o que faz com que o sistema *IndiQoS* se torne mais tolerante a faltas. Outra desvantagem em ter apenas um Ponto de Contacto para um determinado tipo de eventos consiste na congestão dos caminhos perto do Ponto de Contacto. A Figura 3.7 mos-

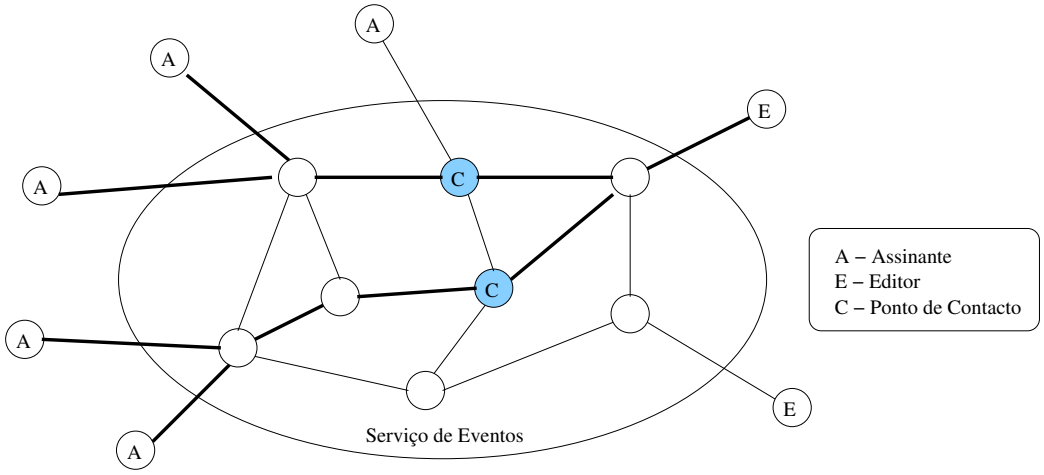


Figura 3.9: Árvore de distribuição de eventos de um determinado tipo com dois Pontos de Contacto.

tra a árvore de distribuição de eventos de um determinado tipo. Nesta Figura pode-se verificar que a árvore de distribuição se mais concentrada em redor do Ponto de Contacto. Já na Figura 3.9, os fluxos que compõem a árvore de distribuição com dois Pontos de Contacto encontram-se mais espalhados pelo sistema, evitando a congestão nos caminhos que se encontram perto do Ponto de Contacto.

3.3.7 Pontos de Contacto e a QdS

O aumento do número de Pontos de Contacto para um tipo de eventos tem também influência na escolha de caminhos que satisfazem a QdS pedida pelas aplicações. Com o aumento do número de pontos de Contacto passa a haver um leque maior na escolha de caminhos entre os Editores e Assinantes. Desta forma, com a introdução da replicação dos *n* Pontos de Contacto para um determinado tipo de evento, o algoritmo de definição de caminhos processa-se como demonstrado na Figura 3.10. Esta figura

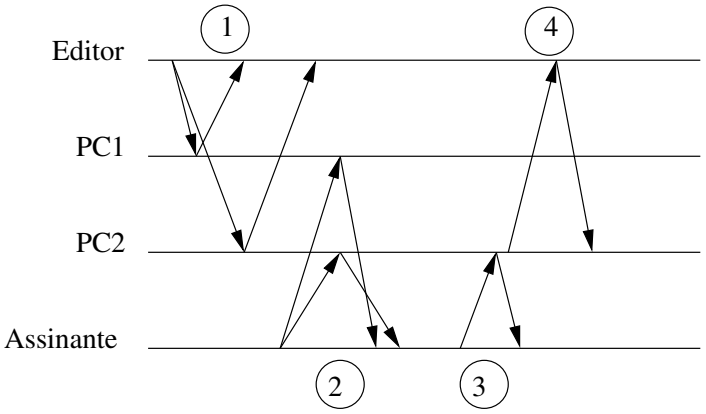


Figura 3.10: Publicação e subscrição de eventos com 2 Pontos de Contacto.

mostra como se processa o registo de um Editor e de um Assinante no sistema *IndiQoS* com 2 Pontos de Contacto para cada tipo de eventos.

O Editor começa por se registar em ambos os Pontos de Contacto (*PC*1 e *PC*2). O Assinante, por sua vez, regista-se também em ambos os Pontos de Contacto. Ao efectuar o registo, o Assinante recebe duas respostas (uma de cada Ponto de Contacto) com informação acerca da QdS que pode ser fornecida por cada um deles. Com esta informação, o Assinante calcula o Ponto de Contacto mais apropriado para lhe fornecer as notificações. No exemplo, o Ponto de Contacto escolhido é o *PC*2. O Assinante indica a *PC*2 a sua escolha e *PC*2 indica ao Editor que pode começar a enviar notificações. Este último passo é efectuado apenas no registo do primeiro Assinante.

Com apenas um Ponto de Contacto, as escolhas de caminhos entre o Editor e o Assinante é reduzido. Como o sistema tenta sempre encontrar o Ponto de Contacto, os caminhos que o rodeiam ficariam rapidamente congestionados e sem recursos livres. Com esta nova forma de definição de caminhos entre o Editor e o Assinante, o número de escolhas aumenta,

aumentando assim a possibilidade de encontrar um caminho que satisfaz os requisitos das aplicações. Outra característica positiva consiste no facto de que as reservas de recursos não ficam concentradas apenas em alguns pontos da rede, mas ficam mais distribuídas e dispersas na rede de nós entre-pares.

3.4 Sumário

Depois de apontados os requisitos necessários à construção de um sistema Publicação-Subscrição com QdS, este capítulo descreve a arquitectura *IndiQoS*. Esta arquitectura é composta por dois tipos de aplicações (Editores e Assinantes) e um Serviço de Eventos, que se encontra distribuído por uma rede de nós entre-pares.

Foi definida uma API exportada pelo sistema *IndiQoS* para a construção de aplicações. Esta API proporciona às aplicações métodos para iniciar e terminar o envio/recepção de dados, indicando ao sistema qual a QdS pretendida. Esta API tem também a particularidade de esconder das aplicações as reservas de recursos efectuadas.

O Serviço de Eventos do sistema *IndiQoS* usa a noção de Pontos de Contacto. Os Pontos de Contacto são os responsáveis por manter informação sobre um determinado tipo de informação, sendo também um ponto chave na definição dos caminhos entre um Editor e um Assinante. As aplicações começam por efectuar um registo no seu Ponto de Contacto indicando o tipo de informação e a QdS pretendida. As mensagens são encaminhadas usando uma TDD, o Bamboo. Sendo uma TDD, para enviar uma mensagem no Bamboo é necessário uma chave que identifica um nó. As chaves de um nó nomeado Ponto de Contacto são encontrados

localmente, através de um algoritmo determinista que se baseia no tipo de evento.

Um dos pontos chave da arquitectura *IndiQoS* consiste no uso de mais do que um Ponto de Contacto para um único tipo de eventos. A informação sobre um determinado tipo de nós encontra-se distribuída por um número *n* de Pontos de Contacto. Este aspecto torna o sistema mais tolerante a faltas e faz também com que se aumente o leque de escolhas na definição de caminhos entre o Editor e o Assinante. Com o uso de vários Pontos de Contacto, as aplicações registam-se em todos eles e o sistema escolhe os caminhos mais apropriados entre os Editores e os Assinantes.

Finalmente, outra característica do sistema *IndiQoS* reside no uso de componentes de rede genéricos. Estes componentes são denominados *Infopipes* e concretizam diferentes soluções de rede. Para além disso, estes componentes escondem das restantes camadas do sistema certos pormenores da rede que são desnecessários, proporcionando uma maior modularidade ao sistema.

Capítulo 4

Avaliação de resultados

Este Capítulo apresenta uma avaliação da arquitectura *IndiQoS*. Para tal, foram analisados vários aspectos da mesma, nomeadamente (*i*) a escalabilidade do sistema, (*ii*) a taxa de utilização da rede e (*iii*) a qualidade dos caminhos escolhidos. É também analisado o custo em termos de sinalização do sistema *IndiQoS*, que consiste na quantidade de mensagens de controlo necessárias para efectuar os anúncios e subscrições, que se pretende minimizar. Esta avaliação tem o intuito de mostrar as vantagens da utilização de uma rede entre-pares, assim como o de ilustrar a necessidade da replicação dos Pontos de Contacto no sistema. Assim, os critérios usados para avaliar estes aspectos da arquitectura *IndiQoS* foram os seguintes:

Escalabilidade do sistema Para verificar a escalabilidade do sistema foram efectuados testes variando o número de diferentes tipos de eventos a serem anunciados e subscritos pelas aplicações.

Utilização da rede Foram efectuados testes que medem a taxa de utilização dos recursos da rede de dados. Com estes testes consegue-se verificar o quanto o sistema é eficiente na escolha de caminhos entre

Editores e Assinantes.

Qualidade dos caminhos obtidos Para verificar a qualidade das árvores de disseminação de eventos encontradas pelo sistema *IndiQoS*, foi calculada a latência média dos caminhos entre Editores e Assinantes, confrontando o progresso deste valor com (*i*) o aumento do número de replicas dos Pontos de contacto e (*ii*) o aumento da diversidade de tipos.

Custo de sinalização Foi medida a quantidade de mensagens de controlo necessárias para encontrar os caminhos entre Editores e Assinantes.

Os resultados obtidos foram também comparados com outras arquitecturas, que são bastante eficientes na escolha de caminhos óptimos entre dois pontos de uma rede de dados. Estas arquitecturas baseiam-se em várias aproximações de encaminhamento com QdS, que serão descritas de seguida.

4.1 Outras formas de encaminhamento

Existem várias formas de efectuar encaminhamento de dados com restrições de QdS. Por um lado, existem algoritmos em que cada nó obtém e mantém uma representação da rede e usa essa representação para calcular o caminho óptimo entre dois nós dessa rede. Esta alternativa tem algumas desvantagens do ponto de vista prático. Para que qualquer nó consiga aplicar um algoritmo de caminho óptimo no grafo que representa a rede, esse nó terá de manter informação actualizada sobre a mesma. Assim, todas as mudanças da rede têm de ser divulgadas a todos os nós. Esta necessidade aplica-se não só ao estado de uma ligação entre dois nós (que

até se pode assumir como sofrendo variações pouco frequentes), mas também à quantidade de recursos disponível nessa ligação. Mas a divulgação da alteração do estado das ligações a todos os nós não só consome bastante largura de banda como está sujeita à latência das ligações da rede, isto é, uma alteração num ponto da rede só se torna visível mais tarde em pontos mais distantes dessa ligação. Esta solução é usada no âmbito dos sistemas autónomos (Osborne & Simha, 2003), mas não possui capacidade de escala.

Por outro lado existem outros protocolos que não mantêm mais do que o estado das ligações para os seus vizinhos directos. Neste caso, para descobrir um caminho entre dois pontos da rede de nós com uma determinada QdS, é enviado um pedido a todos os vizinhos. Estes nós, ao receberem o pedido e caso não consigam responder a esse pedido, reencaminham-no também para os seus vizinhos e assim sucessivamente. Assim, a rede é inundada com um pedido até chegar ao nó de destino para depois ser efectuada a respectiva resposta. Esta forma de pesquisa de caminhos numa rede de dados é mais eficiente na quantidade de memória usada localmente em cada nó e encontra caminhos com uma qualidade próxima do óptimo, apesar de ter de trocar um número elevado de mensagens de controlo. Esta solução é usada em *Flooding Based QoS Routing* (Pung *et al.*, 1999).

A solução apresentada nesta dissertação baseia-se numa solução intermédia. Por um lado, o conhecimento acerca das publicações e subscrições não está presente em todos os nós da rede de encaminhadores que compõe o mediador do sistema Publicação-Subscrição. Este aspecto faz com que não seja necessário propagar as alterações para todos os nós, mas os caminhos encontrados podem não estar tão perto do caminho óptimo.

Por outro lado, existem alguns nós da rede que guardam informação sobre as publicações e subscrições, nós estes cujo endereço é encontrado de uma forma determinista, sem ser necessária a troca de mensagens. Desta forma, evita-se que a rede seja inundada com mensagens de pesquisa de um nó. Assim, a solução apresentada nesta dissertação foi concebida com o intuito de ter um menor custo quer em termos de número de mensagens de controlo, quer em termos de memória necessária nos encaminhadores, tendo como contrapartida uma escolha menos óptima dos caminhos entre as aplicações.

4.2 Ambiente de testes

Os vários cenários de teste foram executados usando o simulador de rede existente na distribuição da Tabela de Dispersão Distribuída *Bamboo* (Rhea *et al.*, 2003). O simulador de rede usou um grafo representativo das redes de dados reais, gerado pelo GT-ITM (Calvert *et al.*, 1997). Todos os cenários de teste foram executados num computador com duplo processador AMD Opteron ™ 248 com uma velocidade de 2.2 GHz e com 4 Gb de memória RAM. O sistema operativo usado foi o *White Box Enterprise Linux release 3.0*. A aplicação de testes concretiza apenas as funcionalidades necessárias para efectuar os testes, nomeadamente a execução de pedidos com uma determinada QdS.

4.2.1 Simulador de rede

Como foi já descrito no Capítulo 3, a arquitectura *IndiQoS* foi construída usando a Tabela de Dispersão Distribuída (TDD) *Bamboo* (Rhea *et al.*, 2003). As distribuições mais recentes dessa plataforma contêm um simulador de

rede, que foi usado na avaliação da arquitectura descrita nesta dissertação.

Este simulador é usado no Bamboo para simular a rede física que suporta a TDD e usa um grafo gerado por um gerador de redes de dados. Este simulador efectua o encaminhamento das mensagens na rede física e para isso usa o algoritmo de *Dijkstra*. Note-se que não é o encaminhamento efectuado pela rede sobreposta, mas sim a rede física que a suporta. Desta forma, é calculado o tempo que cada mensagem gerada pelo Bamboo iria demorar a ser entregue ao nó seguinte no grafo da rede física, para efectuar a respectiva entrega no instante temporal adequado.

O simulador usado tem a particularidade de correr código real, que (com ligeiras alterações apenas ao nível da configuração dos nós) pode ser também usado em sistemas reais. Desta forma, a concretização apresentada nesta dissertação em conjunto com uma concretização realista dos *Netpipes* (e.g. usando Serviços Integrados) pode ser usada numa rede de dados concreta.

4.2.2 Geração de redes de dados

As redes usadas nas simulações foram geradas por uma ferramenta denominada GT-ITM (Calvert *et al.*, 1997). Esta ferramenta foi criada para gerar grafos representativos da Internet e pode gerar grafos de vários tipos. As redes de domínio de trânsito são constituídas por um conjunto de LANs interligadas entre si.

Desta forma, para criar um grafo usando esta ferramenta é necessário apenas criar um ficheiro de configuração onde se indica (*i*) o número de domínios de transito, (*ii*) o número de nós em cada domínio de transito, (*iii*) o número de nós em cada LAN, assim como (*iv*) o número de LANs que estão ligadas a um domínio de transito. Todos os valores dados são

valores médios. Cada sub-grafo é gerado independentemente dos restantes e, após a sua geração, é testada a conectividade. Grafos não conectados são descartados.

4.2.3 Aplicação de testes

A aplicação de testes usada nas simulações consiste num cliente do sistema *IndiQoS* bastante simples. Este cliente liga-se a um dos encaminhadores que compõem a rede de nós entre-pares para depois efectuar pedidos ao sistema. Os pedidos podem ser anúncios ou subscrições, consoante o tipo de aplicação que for. Em cada simulação efectuada, as aplicações foram colocadas de uma forma aleatória pelos nós do sistema a começam a efectuar pedidos até que se detecte uma sequência de 40 pedidos de subscrição recusadas por falta de recursos. Neste ponto assume-se que a rede está saturada e termina-se a simulação.

4.3 Descrição dos resultados

Esta secção apresenta cada um dos vários cenários avaliados. Em cada cenário, são descritos quais os resultados esperados, que são depois confrontados com os resultados obtidos nas simulações. Os cenários foram montados com a intenção de avaliar a arquitectura *IndiQoS* segundo os critérios definidos no inicio deste capítulo.

De seguida pretende-se mostrar a necessidade e os benefícios da TDD.

4.3.1 Benefícios da TDD

O uso de uma TDD na concretização de um sistema como o *IndiQoS* tem várias vantagens, como a distribuição da carga pelo sistema e uma pesquisa facilitada dos Pontos de Contacto. Nesta Secção pretende-se mostrar as vantagens do uso de uma TDD em termos de utilização da rede. Para tal, as simulações foram efectuadas com redes de 252 encaminhadores *IndiQoS*, dos quais 30% têm uma aplicação ligada, sendo esta aplicação um Assinante. Existe também um Editor para cada tipo de eventos a serem disseminados no sistema. Cada aplicação efectua pedidos, cuja reserva corresponde a 12.5% da largura de banda existente nas ligações. Nestas simulações fez-se variar o número de tipos anunciados/subscritos no sistema.

Discussão dos resultados

O uso de uma TDD facilita a diferenciação dos tipos de dados fornecidos pelo sistema. Os Editores anunciam um tipo de eventos registando-se num Ponto de Contacto. Esse Ponto de Contacto é escolhido por um algoritmo determinista que encontra uma chave através de um determinado tipo. Uma das características das TDD consiste na uniformidade com que os encaminhadores são escolhidos. Diferentes chaves (ou tipos) dão origem a diferentes encaminhadores com uma probabilidade muito alta. Os encaminhadores vão sendo ocupados uniformemente pela rede. Esta característica da TDD faz com que diferentes tipos fiquem a cargo de diferentes encaminhadores, fazendo com que a carga do sistema não se concentre apenas numa parte da rede.

A probabilidade de um encaminhador ser escolhido como ponto de contacto para vários tipos de eventos é bastante pequena e diminui com

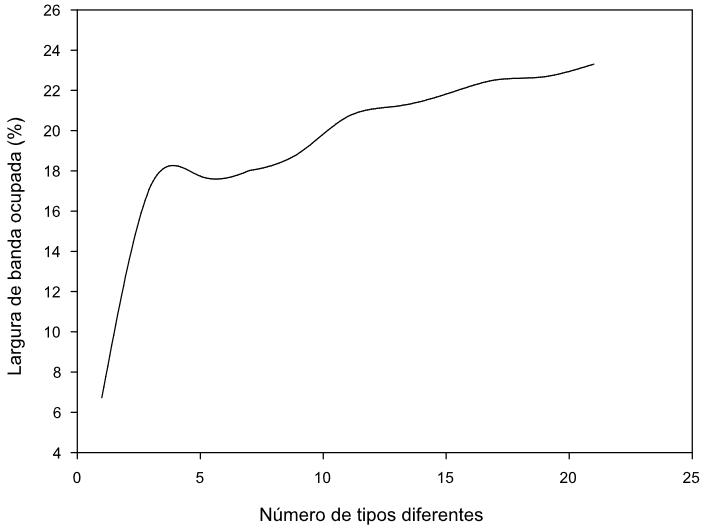


Figura 4.1: Taxa de utilização vs tipos de eventos (%).

o aumento do tamanho da rede. Como o sistema dispõe de vários Pontos de Contacto a aceitar pedidos de clientes (um por cada Tipo) e como os Pontos de Contacto se encontram espalhados pelo sistema, quanto maior for o número de diferentes tipos, maior deverá ser a taxa de utilização da rede de dados.

A Figura 4.1 mostra o crescimento da utilização máxima da rede, em função do número de diferentes tipos. Com apenas um tipo, só um encaminhador da rede entre-pares é responsável por manter informação sobre Editores e Assinantes. Isto é equivalente a uma solução centralizada e como se pode verificar a largura de banda rapidamente se esgota. Com o aumento do número de tipos, aumenta também o número de encaminhadores responsáveis por receber pedidos de Editores e Assinantes, aumentando também a taxa de utilização da rede.

4.3.2 Replicação dos Pontos de Contacto

Nesta sequência de testes pretende-se demonstrar a necessidade da replicação dos Pontos de Contacto no sistema *IndiQoS*. Sendo assim, estas simulações foram efectuadas com redes de 252 encaminhadores *IndiQoS*, dos quais 30% têm uma aplicação ligada, sendo esta aplicação um Assinante. Existe ainda um Editor que anuncia um determinado tipo de eventos. Cada aplicação efectua pedidos, cuja reserva corresponde a 12.5% da largura de banda existente nas ligações. Nestes testes fez-se variar o número de replicas dos Pontos de Contacto.

Discussão dos resultados

Tendo apenas um Ponto de Contacto para um tipo de eventos, o número máximo de ligações que conseguem ser efectuadas com sucesso está ligado à quantidade de largura de banda total que se consegue reservar até ao Ponto de Contacto. Cada Ponto de Contacto dispõe de uma largura de banda igual ao somatório da largura de banda de cada ligação na rede física que o Ponto de Contacto tem para os seus vizinhos. Vamos denominar esta quantidade de largura de banda como LB_{PC}. Se cada pedido efectuado por um cliente necessitar de uma largura de banda de LB_{P_T} (Largura de Banda do Pedido T), o Ponto de Contacto irá suportar no máximo $\frac{LB_{PC}}{LB_{P_T}}$ pedidos.

Ao aumentar o número de Pontos de Contacto de um determinado tipo, para além de se estar a aumentar as hipóteses de escolha de caminhos diferentes (aumentando a hipótese de encontrar um caminho mais próximo do caminho óptimo em termos de latência mínima), também se está a aumentar a largura de banda máxima que o sistema pode fornecer. Intuitivamente, se cada tipo de eventos tiver N replicas, temos aproxima-

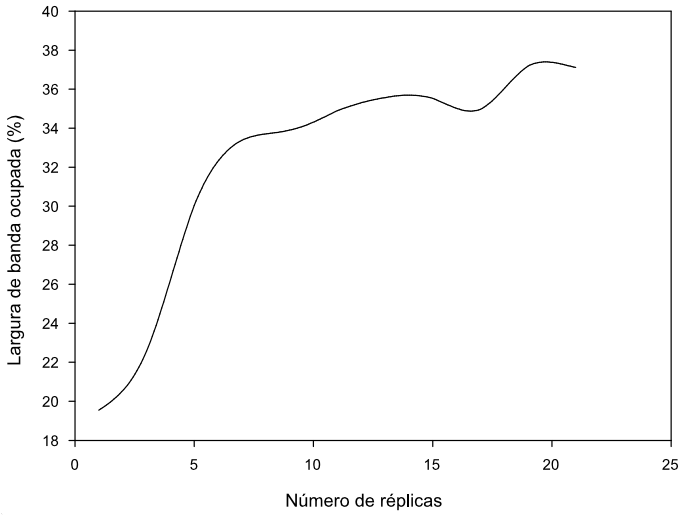


Figura 4.2: Utilização da rede vs replicação (%).

damente uma largura de banda máxima igual a $\sum_{i=1}^{N} LB_{PC_i}$. Desta forma, a taxa de utilização da rede aumenta com o número de replicas dos Pontos de Contacto. Para valores altos do número de réplicas este máximo nunca é atingido, já que a possibilidade de haver caminhos que se cruzam começa a aumentar.

O gráfico representado na Figura 4.2 mostra o aumento da utilização da rede com o aumento do número de replicas de um determinado tipo de eventos. Cada aplicação subscreve/anuncia o mesmo tipo de eventos. Como se pode verificar, com um número bastante baixo de replicas (5 neste caso), consegue-se já obter uma taxa de utilização bastante alta em relação à inexistência das mesmas. Isto significa que com a existência de um pequeno número de réplicas, o sistema consegue satisfazer mais pedidos de subscrição de eventos.

Outra vantagem que existente na replicação dos Pontos de Contacto consiste na qualidade dos caminhos entre o Editor e o Assinante. Sendo

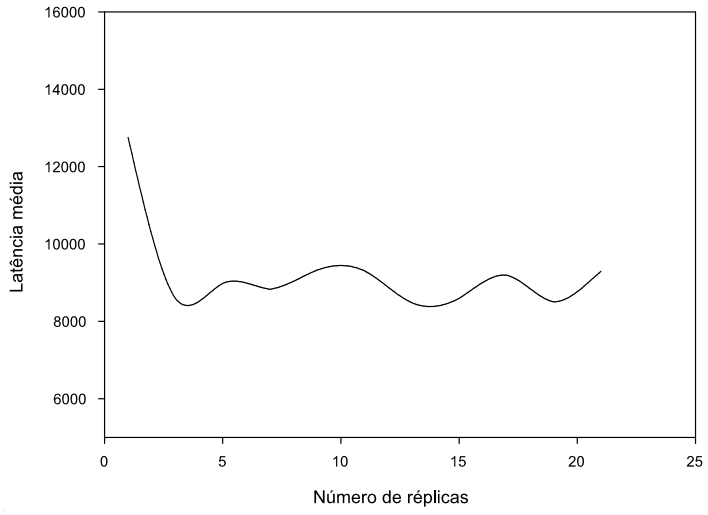


Figura 4.3: Latência média vs replicação (μs).

a TDD a escolher os caminhos e como o Ponto de Contacto é o ponto de ligação entre um Editor e um Assinante, se só houver um Ponto de Contacto apenas existe uma hipótese na definição do caminho. Aumentando o número de pontos de Contacto, aumentam-se as hipóteses de escolha de caminhos. Com vários Pontos de Contacto, o sistema pode calcular a latência em cada um dos caminhos escolhidos pela TDD e escolher o mais curto dos caminhos que satisfazem as necessidades de largura de banda. Um maior número de Pontos de Contacto significa um aumento das hipóteses de encontrar caminhos mais perto do óptimo.

Na Figura 4.3 pode-se verificar a latência média dos caminhos entre Editores e Assinantes, gerados pelo *IndiQoS*. Com um número bastante reduzido de replicas, os ganhos na latência dos caminhos são bastante significativos. Com um Ponto de Contacto verifica-se que os valores da latência média são bastante elevados. Isto deve-se ao facto de que qualquer caminho entre um Editor e um Assinante ter de passar pelo mesmo

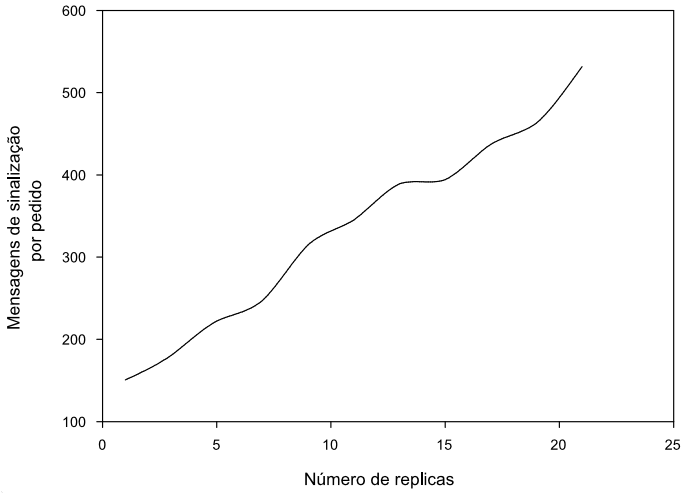


Figura 4.4: Sinalização vs replicação (*ms*).

Ponto de Contacto, independentemente da distância entre as aplicações. Como os Pontos de Contacto são colocados uniformemente no espaço da rede de nós entre-pares, a latência dos caminhos vai diminuindo com o crescimento do número de Pontos de Contacto.

O aumento do número de replicas do ponto de Contacto de cada tipo de dados tem como contrapartida um aumento do número de mensagens de sinalização, já que a aplicação tem de se registar em todos os Pontos de Contacto do tipo que deseja anunciar/subscrever. A Figura 4.4 mostra o número médio de mensagens de controlo geradas pelo sistema para cada subscrição. Ao aumentar o número de replicas, o número de mensagens de controlo também aumenta. Como se poderá ver na Secção seguinte, ao comparar a aproximação descrita nesta dissertação com outras alternativas, o custo de sinalização numa configuração do *IndiQoS* com poucas replicas, é bastante competitivo.

4.3.3 Analise comparativa

Este último cenário compara a qualidade dos caminhos obtidos pelo sistema *IndiQoS* com outras aproximações. Uma das formas de definição de caminhos entre Editores e Assinantes consiste em efectuar uma *Inundação de Pedidos*, que se baseia no protocolo *Flooding Based QoS Routing* (Pung *et al.*, 1999). Outra forma consiste em executar um algoritmo localmente sobre um grafo representativo da rede. Para tal, cada encaminhador necessita de saber o estado global da rede, sendo necessário efectuar actualizações de estado sempre que alguma ligação é alterada. Estas actualizações têm também de ser propagadas. Este tipo de protocolos são do tipo *Inundação de estado*. O algoritmo usado na definição dos caminhos entre Editores e Assinantes consiste no Dijkstra, algoritmo usado também no QoSPF (Apostolopoulos *et al.*, 1999).

Nesta Secção, os resultados do sistema *IndiQoS* são comparados com uma versão baseada no protocolo de *Inundação de pedidos* e outra baseada em *Inundação de estado*. Nos testes efectuados para confrontar os 3 sistemas existe apenas um tipo de eventos a ser anunciado/subscrito. Em todos os sistemas foi usada uma rede de 252 nós, onde em cada nó existe um encaminhador. Foram usadas duas configurações do sistema *IndiQoS*, com um e três Pontos de Contacto. As aplicações não efectuam pedidos de forma concorrente, já que não se pretende comparar o comportamento dos sistemas nestas condições.

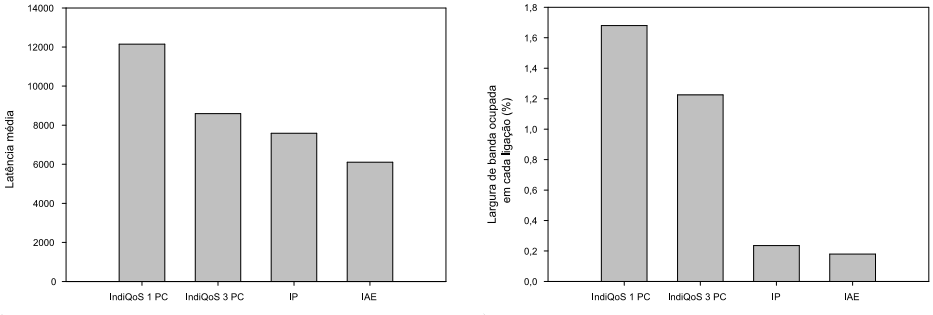
Discussão dos resultados

O protocolo baseado em *Inundação de Pedidos* não necessita de manter informação sobre os restantes nós da rede. Apenas necessita de manter informação sobre o estado das ligações que tem com os seus vizinhos. O pro-

tocolo foi concebido desta forma para evitar a propagação de actualizações de estado das ligações por todos os vizinhos. A forma encontrada para definir um caminho entre dois nós da rede consiste em inundar a rede com um pedido até que esse pedido chegue ao nó de destino. A primeira cópia de um pedido a chegar ao nó de destino contém o caminho óptimo na rede. Em conjunto com este mecanismo de pesquisa, existem mecanismos para podar os pedidos e para eliminar duplicados. Ao conceber um protocolo baseado neste tipo de pesquisa, evitando actualizações de estado, acredita-se que os pedidos são menos frequentes do que as mudanças de estado em todas as ligações de uma rede. Esta segunda opção, *Inundação de estado*, constrói ligações entre as aplicações, definidas segundo um grafo representativo da rede de dados, grafo este que é mantido por todos os nós. Com toda a informação necessária sobre a rede (ligações, largura de banda de cada ligação, respectiva latência e localização das aplicações), é possível descobrir localmente o caminho óptimo.

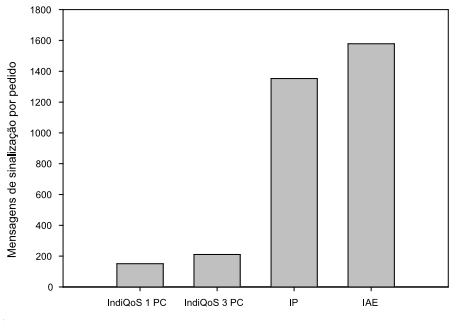
A Figura 4.5 confronta os três sistemas, em termos de custos de sinalização, latência média dos caminhos entre Editores e Assinantes e largura de banda média reservada na rede para cada ligação. Como se pode verificar, apesar dos valores de latência média e largura de banda média consumida numa ligação serem piores, o custo de sinalização do sistema *IndiQoS* é substancialmente mais pequeno. Como o *IndiQoS* efectua o encaminhamento sem conhecimento global da rede, não consegue obter caminhos tão bons como os outros sistemas. Ainda assim, o acréscimo da latência nos outros sistemas é mais pequeno, quando comparado com o decréscimo em termos de custos de sinalização.

O Ponto de Contacto do sistema *IndiQoS* dispõe de um conjunto de vizinhos, conjunto este que é definido pela TDD, ou seja, dispõe de menos



(a) Latência média (μs)

(b) Utilização da rede



(c) Sinalização

Figura 4.5: Comportamento nos três sistemas: *IndiQoS* (1 Ponto de Contacto e 3 Pontos de Contacto), Inundação de Pedidos (IP) e Inundação de Alterações de Estado (IAE).

ligações do que tem na rede física. Desta forma, a largura de banda disponível para chegar até ao Ponto de Contacto acaba por ser inferior, sendo também inferior o número máximo de ligações que o sistema consegue estabelecer. A partir dos resultados apresentados na Figura 4.5 pode-se concluir que os caminhos encontrados pelo sistema *IndiQoS* não são os óptimos, já que quer latência média, quer a largura de banda média consumida em cada ligação são superiores em relação aos outros sistemas. No caso do sistema *IndiQoS* com três Pontos de Contacto para cada tipo de eventos, a latência média dos caminhos gerados são aproximadamente duas vezes pior em relação aos outros sistemas, mas o custo em termos de sinalização é oito vezes melhor. Quando se considera a taxa de utilização da rede, o sistema *IndiQoS* com apenas três Pontos de Contacto consome em média seis vezes mais largura de banda em relação às restantes alternativas. Esta diferença diminui com o aumento do número de réplicas do Ponto de Contacto.

4.4 Sumário

Este Capítulo apresenta uma avaliação da arquitectura *IndiQoS*. Para tal, foram analisados aspectos como a taxa de utilização do sistema e a qualidade dos caminhos escolhidos, assim como a quantidade de mensagens de controlo necessárias para efectuar os anúncios e subscrições. Na avaliação apresentada mostraram-se as vantagens da utilização de uma rede entre-pares, assim como a necessidade da replicação dos Pontos de Contacto no sistema.

Os resultados mostram que com uma pequena quantidade de replicas de um Ponto de Contacto para um determinado tipo de evento, a taxa de

utilização do sistema aumenta, sendo possível satisfazer uma maior quantidade de subscrições. Verificou-se também que o aumento do número de replicas do Ponto de Contacto contribui para uma melhoria substancial dos caminhos encontrados para interligar Editores e Assinantes. É também mostrado que o sistema *IndiQoS* é escalável no aumento de tipos existentes no sistema.

Os resultados obtidos foram comparados com outras arquitecturas que são bastante eficientes na escolha de caminhos óptimos entre dois pontos de uma rede de dados. Para tal foi construída outra versão do sistema *IndiQoS*, usando o método de inundação de pedidos na descoberta dos caminhos entre os Editores e os Assinantes. Foi também concretizado um cenário de simulação que corresponde à inundação de alterações de estado nas ligações. Neste cenário, cada nó mantém uma representação da rede e usa o algoritmo de Dijkstra para calcular os caminhos entre Editores e Assinantes. As três soluções foram comparadas e verificou-se que o custo de sinalização do *IndiQoS* é bastante competitivo, em relação aos outros sistemas.

Capítulo 5

Conclusões e trabalho futuro

Os modelos de comunicação por eventos como o Publicação-Subscrição têm emergido nos últimos anos como uma alternativa ao modelo de comunicação pedido-resposta. Nestes modelos, existem dois tipos de participantes: os *Editores*, que efectuam a publicação de eventos, e os *Assinantes* que efectuam a subscrição de eventos. A principal vantagem dos sistemas Publicação-Subscrição consiste no desacoplamento entre participantes, que não necessitam de ter conhecimento quer da localização, quer do número de elementos existentes no sistema. Uma das limitações presentes na maior parte das arquitecturas que suportam comunicação indirecta do tipo Publicação-Subscrição consiste na falta de suporte à expressão de parâmetros de Qualidade de Serviço (QdS) na rede de dados. A falta de suporte à expressão de parâmetros de QdS neste tipo de sistemas é uma desvantagem importante, já que certo tipo de aplicações necessita de garantias de QdS para o seu bom funcionamento. Por outro lado, as soluções que visam oferecer garantias de QdS são tipicamente baseados no estabelecimento explícito de canais ou ligações, os quais servem de suporte para efectuar as reservas dos recursos necessários para fornecer a QdS preten-

dida. Esta abordagem encaixa naturalmente nos sistemas de comunicação directa, mas é de difícil utilização no modelo Publicação-Subscrição.

Assim, esta dissertação apresenta uma nova arquitectura de um sistema Publicação-Subscrição, que fornece garantias de QdS às aplicações – a arquitectura *IndiQoS*. Esta arquitectura permite fornecer às aplicações uma forma de exprimir os parâmetros de QdS pretendidos, assim como remover da aplicação a tarefa de efectuar reservas de recursos. Foi então concebida a arquitectura *IndiQoS* que consiste numa arquitectura Publicação-Subscrição, onde o Mediador se encontra distribuído por vários nós de uma rede entre-pares. Para distribuir o mediador do sistema *IndiQoS* foi usada uma Tabela de Dispersão Distribuída (TDD). Para melhorar o desempenho e a escalabilidade do sistema em termos de quantidade de aplicações aceites pelo mesmo, foi usada a replicação dos Pontos de Contacto usados para interligar Editores e Assinantes. O sistema dispõe de uma API com a qual as aplicações efectuam pedidos de anúncio ou subscrição com uma determinada QdS, sendo esse pedido processado pelo sistema *IndiQoS*. Com os dados obtidos, o sistema define os caminhos e reserva os recursos necessários. A arquitectura é também dividida em diferentes módulos para a tornar reutilizável quer em outras aplicações, quer no uso de diferentes soluções ao nível da rede de dados.

Esta dissertação apresenta uma avaliação da arquitectura que prova a sua viabilidade, mostrando as vantagens da utilização de uma TDD em conjunto com a replicação dos Pontos de Contacto. Os resultados obtidos foram comparados com outras arquitecturas que são bastante eficientes na escolha de caminhos óptimos entre dois pontos de uma rede de dados: um protocolo que se mantém uma representação da rede de dados (sendo necessário notificar todos os menbros do sistema em cada actuali-

zação numa determinada ligação) e um protocolo que evita actualizações de estado, inundando a rede a cada pedido.

Como trabalho futuro, pretende-se integrar na arquitectura *IndiQoS* o trabalho descrito em (Sofia *et al.*, 2002b; Sofia *et al.*, 2003). Este trabalho mostra como a agregação de fluxos de tráfego pode diminuir a quantidade de mensagens de sinalização dos protocolos de reserva de recursos.

Bibliografia

APOSTOLOPOULOS, G., WILLIAMS, D., KAMAT, S., GUERIN, R., ORDA, A., & PRZYGIENDA, T. 1999 (Aug.). *QoS Routing Mechanisms and OSPF Extensions*. RFC 2676.

ARAÚJO, F., & RODRIGUES, L. 2002. On QoS-Aware Publish-Subscribe. *Pages 511–515 of: Proceedings of the International Workshop on Distributed Event-Based Systems*. Vienna, Austria: IEEE. (Proceedings the 22nd International Conference on Distributed Computing Systems Workshops).

BACON, JEAN, MOODY, KEN, BATES, JOHN, HAYTON, RICHARD, MA, CHAOYING, MCNEIL, ANDREW, SEIDEL, OLIVER, & SPITERI, MARK. 2000. Generic Support for Distributed Applications. *IEEE Computer,* Mar.

BETTATI, R., ZHAO, W., & TEODOR, D. 1999. Real-time Intrusion Detection and Suppression in ATM Networks.

BIRRELL, A., & NELSON, B. 1984. Implementing Remote Procedure Calls. *ACM Transactions on Computer Systems,* **2**(1).

BLAKE, S., BLACK, D., CARLSON, M., DAVIES, E., WANG, Z., & WEISS, W. 1998 (December). *An Architecture for Differenciated Services*. RFC 2475.

BRADEN, ED.R., ZHANG, L., BERSON, S., HERZOG, S., & JAMIN, S. 1997 (September). *Resource ReSerVation Protocol (RSVP) — Version 1 Functional Specification*. RFC 2205.

BRADEN, R., CLARK, D., & SHENKER, S. 1994 (June). *Integrated Services in the Internet Architecture: an Overview*. RFC 1633.

CALVERT, KEN, DOAR, MATT, & ZEGURA, ELLEN W. 1997. Modeling Internet Topology. *IEEE Communications Magazine*, June.

CARZANIGA, ANTONIO. 1998 (December). *Architectures for an Event Notification Service Scalable to Wide-area Networks*. Ph.D. thesis, Politecnico di Milano.

CARZANIGA, ANTONIO, ROSENBLUM, DAVID S., & WOLF, ALEXANDER L. 2000 (Jan.). *Content-Based Addressing and Routing: A General Model and its Application*. Tech. rept. CU-CS-902-00. Department of Computer Science, University of Colorado.

CARZANIGA, ANTONIO, ROSENBLUM, DAVID S., & WOLF, ALEXANDER L. 2001. Design and Evaluation of a Wide-Area Event Notification Service. *ACM Transactions on Computer Systems*, **19**(3), 332–383.

CHOCKLER, GREGORY, KEIDAR, IDIT, & VITENBERG, ROMAN. 2001. Group communication specifications: a comprehensive study. *ACM Computing Surveys*, **33**(4), 427–469.

COOPER, ERIC C. 1984 (Aug.). Replicated Procedure Call. *In: Proceedings of the 3rd ACM symposyum on Principles of Distributed Computing*. ACM, Berkeley, CA 94720, USA.

DIOT, CHRISTOPHE, GIULIANO, LEONARD, SHEPHERD, GREG, ROCKELL, ROBERT, MEYER, DAVID, MEYLOR, JOHN, & HABERMAN, BRIAN. 2003 (July). *An Overview of Source-Specific Multicast (SSM).* RFC 3569.

ESTRIN, D., FARINACCI, D., HELMY, A., THALER, D., DEERING, S., HANDLEY, M., JACOBSON, V., LIU, C., P.SHARMA, & WEI, L. 1998 (June). *Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification*. RFC 2362.

EUGSTER, P. T., GUERRAOUI, R., & SVENTEK, J. 2000. *Type-Based Publish/-Subscribe*. Tech. rept. Swiss Federal Institute of Technology in Lausanne (EPFL).

EUGSTER, P. TH., GUERRAOUI, R., & DAMM, CHRISTIAN H. 2001 (October). Linguistic Support for Large-Scale Distributed Programming. *Pages 131–146 of: In 16th ACM Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA 2001)*.

FELBER, TH. EUGSTER. 2001. *The Many Faces of Publish/Subscribe*. Tech. rept. Swiss Federal Institute of Technology in Lausanne (EPFL).

GAREY, M. R., & JOHNSON, DAVID S. 1977. The Rectilinear Steiner Tree Problem in NP Complete. *SIAM Journal of Applied Mathematics*, **32**, 826–834.

GELERNTER, D. 1985. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, Jan., 80–112.

GUIMARAES, M., & RODRIGUES, L. 2003 (Apr.). A Genetic Algorithm for Multicast Mapping in Publish-Subscribe Systems. *Pages 67–74 of: Proceedings of the 2nd IEEE International Symposium on Network Computing and Applications*.

GUO, L., & MATTA, I. 1999. QDMR: An Efficient QoS Dependent Multicast Routing Algorithm. *In: Proc. of the Fifth IEEE Real-Time Technology and Applications Symposium (RTAS '99)*.

HAPNER, MARK, BURRIDGE, RICH, SHARMA, RAHUL, FIALLI, JOSEPH, & STOUT, KATE. 2002 (April). *Java Message Service*. Sun Microsystems.

KOH, SEOK JOO, & KANG, SHIN GAK. 2001 (June). Enhancement of the CBT Multicast Routing Protocol Based on Backbone Core Tree. *In: Proceeding of the International Conference on Parallel And Distributed Systems*.

KOSTER, R., BLACK, A. P., HUANG, J., WALPOLE, J., & PU, C. 2001 (Oct.). Infopipes for Composing Distributed Information Flows. *Pages 44–47 of: Proceedings of the International Workshop on Multimedia Middleware*. ACM.

LEVINE, BRIAN NEIL, CROWCROFT, JON, DIOT, CHRISTOPHE, GARCIA-LUNA-ACEVES, J. J., & KUROSE, JAMES F. 2000. Consideration of Receiver Interest for IP Multicast Delivery. *Pages 470–479 of: INFOCOM (2)*.

MOY, J. 1994a (Mar.). *MOSPF: Analysis and Experience*. RFC 1585.

MOY, J. 1994b (Mar.). *OSPF Version 2*. Tech. rept. Proteon, Inc.

NICHOLS, K., ANDF. BAKER, S. BLAKE, & BLACK, D. 1998 (December). *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*. RFC 2474.

OMG. 2001 (Mar.). *Event Service Specification*. Object Management Group.

OMG. 2002 (Aug.). *Notification Service Specification*. Object Management Group.

OSBORNE, ERIC, & SIMHA, AJAY. 2003. *Traffic Engineeering with MPLS*. Cisco Press.

PIETZUCH, P., & BACON, J. 2002. Hermes: A Distributed Event-Based Middleware Architecture. *In: 22nd IEEE International Conference on Distributed Computing Systems Workshops (DEBS '02)*.

PLAXTON, C. GREG, RAJARAMAN, RAJMOHAN, & RICHA, ANDREA W. 1997. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. *Pages 311–320 of: ACM Symposium on Parallel Algorithms and Architectures*.

PUNG, H. K., SONG, J., & L.JACOB. 1999 (September). Fast and Efficient Flooding Based QoS Routing Algorithm. *Pages 298–303 of: Proceedings of IEEE ICCCN99*.

RAMANATHAN, S. 1996. Multicast tree generation in networks with asymmetric links. *IEEE/ACM Transactions on Networking*, **4**(4), 558–568.

RHEA, SEAN, GEELS, DENNIS, ROSCOE, TIMOTHY, & KUBIATOWICZ, JOHN. 2003 (Dec.). *Handling Churn in a DHT*. Tech. rept. University of California at Berkeley.

ROSS, KEITH W., BIERSACK, ERNST W., FELBER, PASCAL, GARCES-ERICE, LUIS, & URVOY-KELLER, GUILLAUME. 2003. *Topology-Centric Look-Up Service*.

ROWSTRON, ANTONY, & DRUSCHEL, PETER. 2001. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. *Lecture Notes in Computer Science*, **2218**.

SOFIA, R., GUÉRIN, R., & VEIGA, P. 2003 (June). SICAP, a Shared-Segment Control Aggregation Protocol. *In: Proceedings of the High Performance Switching and Routing, HPSR*.

SOFIA, RUTE, GUÉRIN, ROCH, & VEIGA, PEDRO. 2002a (July). *An Investigation of Inter-Domain Control Aggregation Procedures*. DI/FCUL TR 02–8. Department of Informatics, University of Lisbon. Superseded by report 02-14.

SOFIA, RUTE, GUÉRIN, ROCH, & VEIGA, PEDRO. 2002b (Nov.). An Investigation of Inter-Domain Control Aggregation Procedures. *In: IEEE International Conference on Network Protocols*.

SUDELL, ANDREW B. 1998 (dec). *Design and Implementation of a Tuple-Space Server for Java*. `http://www.op.net/~asudell/is/linda/linda.html`.

TANG, PUQI PERRY, & TAI, TSUNG-YUAN CHARLES. 1999. Network Traffic Characterization Using Token Bucket Model. *Pages 51–62 of: INFOCOM (1)*.

WROCLAWSKI, J. 1997 (September). *The Use of RSVP with IETF Integrated Services*. RFC 2210.

YAN, SHUQIAN, FALOUTSOS, MICHALIS, & BANERJEA, ANINDO. 2002 (Feb.). *QoS-Aware Multicast Routing for the Internet: The Design and Evaluation of QoSMIC*.

ZHAO, B. Y., KUBIATOWICZ, J. D., & JOSEPH, A. D. 2001 (Apr.). *Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing*. Tech. rept. UCB/CSD-01-1141. UC Berkeley.