Pedro Rafael Pereira Martins

Licenciatura em Engenharia Informática

Um Algoritmo de Disseminação Filtrada baseado em Super Nós e na Redundância de Filtros

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática

Orientador : Sérgio Marco Duarte, Professor Auxiliar, Universidade Nova de Lisboa

Júri:

Presidente:	Prof. Doutora Maria Armanda S. Rodrigues Grueau
Arguente:	Prof. Doutor Hugo Alexandre Tavares Miranda
Vogal:	Prof. Doutor Sérgio Marco Duarte

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Dezembro, 2011

Um Algoritmo de Disseminação Filtrada baseado em Super Nós e na Redundância de Filtros

Agradecimentos

Em primeiro lugar, gostaria de agradecer ao Professor Sérgio Duarte, pelo apoio dado na realização deste trabalho, pela disponibilidade para esclarecer dúvidas e resolver problemas e pelas sugestões e opiniões que ajudaram ao desenvolvimento do trabalho.

Aos meus pais e à minha irmã, por sempre me ajudarem, motivarem, por me terem dado força para poder realizar este trabalho e por me proporcionarem a possibilidade de acabar os meus estudos.

À minha namorada, Teresa, por estar sempre a meu lado, motivando-me e dando me força para acabar o trabalho.

Aos meus amigos, Afonso, Filipe, Horta, Jaime, João Miguel, Mesquita, Miguel, Vasco, Ricardo, Veiga, pelo companheirismo, bons momentos que passamos e pelo que nos divertimos.

E aos meus colegas de faculdade, José Rio, João, Ramos, Ricardo e Saramago, por me acompanharem na faculdade durante estes anos de faculdade, ajudando-me no que fosse preciso.

Resumo

Os sistemas Editor/Assinante baseados no conteúdo têm como objectivo fazer chegar mensagens, ou notificações, somente aos participantes que expressaram interesse nesses dados específicos. Para tal, esta forma de difusão de informação, dita filtrada, recorre a filtros que inspeccionam o conteúdo das mensagens, de modo a seleccionar aquelas que contêm os padrões desejados, descartando-se as demais. Esta abordagem potencia comunicações mais eficientes, pois procura não encaminhar mensagens para as quais não há receptores. E do ponto de vista do receptor, evita-se perder tempo com mensagens não desejadas, algo que é cada vez mais importante num mundo onde a informação que nos chega cresce incessantemente.

A difusão filtrada tem-se revelado um problema difícil, em particular quando realizada com base em soluções descentralizadas. Muitas das dificuldades prendem-se com os problemas decorrentes da escala e do dinamismo da rede de participantes. O trabalho apresentado nesta dissertação, explora a redundância dos filtros, como forma de mitigar o impacto negativo da escala e de uma taxa excessiva de entrada de novos participantes.

Partindo de uma solução existente, baseada no conhecimento completo de todos os nós da rede, a nova solução proposta organiza a rede de disseminação em duas camadas. Na camada superior, povoada por super-nós, está representado o conjunto de filtros únicos no sistema, enquanto que participantes com filtros repetidos na rede são relegados para o nível inferior. Esta solução traduz-se em custos que dependem maioritariamente do ritmo de chegada de participantes cujos filtros ainda não estão representados no sistema, o que na presença de redundância significativa, tenderá a ser bastante inferior que ao custo original. Complementarmente, foram também desenvolvidos e avaliados experimentalmente mecanismos de tolerância a falhas, de modo a garantir a entrega fiável de mensagens com elevada probabilidade.

Palavras-chave: Editor/Assinante, disseminação filtrada, par-a-par, super nós, *churn*

Abstract

The objective of content based Publish/Subscribe systems is to send messages, or notifications, only to users interested in that specific data. In order to do that, this method of dissemination, said content based, uses filters to inspect messages content for selecting those who contain the wished patterns, discarding the rest. With this approach, efficient communications are enhanced, because messages without receptors aren't forwarded. From the recipient point of view, it avoids the waste of time with undesired messages, something that is very important in a world where information we receive grows a lot.

Content based routing has proven to be a difficult problem, specifically when used with descentralized solutions. A lot of the difficulties stuck to the recurrent scale and network dinamics problem. This thesis presented work explores filter redundancy, in order to relieve scale's negative impact and excessive churn rates.

Starting with a existent system, based on the total knowledge of the network, the presented solution builds the dissemination network in two layers. In the upper layer, populated by super nodes, is represented the unique set of filters in the system, while users with repeated filters in the network are sent to a bottom layer. This solution's cost is represented mostly by users, with filters without representation on the system, entry rate, which, with significant redundancy, will tend to be much lower than the original cost. In addition were developed some fault tolerant mechanisms, to ensure reliabe message delivery with high.

Keywords: Publish/Subscribe, content based routing, peer-to-peer, super nodes,*churn*

Conteúdo

Lista de Figuras

Lista de Tabelas

1

Introdução

1.1 Motivação

A disseminação de informação através da Internet tornou-se, nos dias de hoje, um facto incontornável, como a televisão, o rádio e a imprensa. A facilidade e o alcance deste novo meio levou a que, o volume da informação que é hoje disponibilizada esteja a crescer a um ritmo que é já difícil de acompanhar[Wik09], face ao tempo limitado que dispomos para a consumir. Assim, justificam-se cada vez mais, meios que procurem a difusão de informação de forma eficaz, dirigida, mas não intrusiva.

Nesse sentido, formas de disseminação de informação em que os conteúdos sejam encaminhados apenas para quem tenha, previamente, manifestado interesse neles, poderão poupar aos utilizadores tempo gasto na triagem de dados não solicitados. Consequentemente, estes métodos de disseminação de dados, contribuem para melhorar a experiência global nos sistemas e aplicações. O paradigma Editor/Assinante[MD10], baseado no conteúdo, é um modelo de comunicações que endereça precisamente esta questão.

O modelo Editor/Assinante genérico caracteriza-se pela existência de utilizadores que publicam informação, denominados de editores, enquanto que os participantes que consomem a informação publicada são subscritores ou assinantes. Este esquema de funcionamento desacopla editores e subscritores no espaço e no tempo[EFGK03]. Ou seja, um subscritor para receber um conteúdo desejado não precisa de interagir directamente com o editor que o publicou e pode receber a informação em qualquer altura no tempo, não sendo forçado a obtê-la no momento de publicação.

Quando a quantidade de informação publicada é muito grande, este esquema de publicação e subscrição de conteúdos pode tornar-se ineficiente, recebendo os utilizadores

muita informação que não desejam. Por essa razão, surgiu uma abordagem em que o conteúdo recebido é determinado por critérios definidos por cada assinante. A disseminação e subscrição de conteúdos pode, então, ser realizada por tópicos ou canais. Estes proporcionam uma forma simples, ou grosseira, de filtragem, que permite que o participante na rede limite a informação que lhe chega, de acordo com um certo tipo de conteúdo ou mensagens relacionadas.

O conceito de difusão filtrada, também conhecido por Editor/Assinante baseado no conteúdo vai mais longe, e propõe que cada mensagem, individualmente, seja disseminada unicamente para os nós com assinantes, cujos filtros aceitam esses dados. A aceitação é realizada com base em critérios diversos, que podem englobar parte, ou a totalidade, dos dados contidos em cada mensagem. Para tal, os utilizadores declaram interesse na informação que querem receber especificando, de forma explícita, um filtro. Este será avaliado para cada mensagem individualmente, sendo estas apenas encaminhadas para os assinantes quando satisfazem o padrão desejado. Nestes sistemas existe, também, a notificação da actualização de conteúdos subscritos, encaminhados pelo mesmo método para os assinantes.

Existem dois aspectos fulcrais neste tópico: *falsos positivos* e *falsos negativos*. Os primeiros ocorrem quando um utilizador recebe conteúdo que não especificou no seu filtro que queria receber. Já os segundos ocorrem quando um participante deseja receber um tipo de conteúdo, mas não recebe uma publicação desse conteúdo.

O funcionamento óptimo de um esquema de difusão filtrada requer que não existam falsos positivos, por uma razão de eficiência da rede, e que não existam falsos negativos por motivos de correcção do algoritmo.

Existem várias alternativas de arquitectura para implementar sistemas do tipo Editor/Assinante, desde um sistema centralizado até sistemas com vários níveis de descentralização, incluindo a descentralização total. Assim as abordagens centralizadas terão uma arquitectura Cliente/Servidor, enquanto que as descentralizadas procuraram explorar as características de um sistema *P2P*[1].

Um servidor centralizado a que todos os nós estão ligados é uma das possibilidades, chamando-se a este servidor *broker*. Nesta abordagem os editores de informação enviam-na para o servidor, notificando este os participantes que a desejam receber. Desta forma, para cada nova publicação, o servidor avalia todos os filtros escolhidos pelos subscritores, notificando os nós que manifestaram interesse nesse tipo de informação. Esta análise de filtros requer bastante trabalho por parte do *broker*, tornando-se este num ponto de congestão e falha crítico visto toda a informação passar por este bem como as consequentes notificações. Considerando estes problemas, uma abordagem mais descentralizada poderá ser uma alternativa capaz e mitigar esta questão e trazer melhorias significativas de desempenho, distribuição da carga do sistema e tolerância a falhas.

A solução pode passar, então, pela utilização de vários *brokers* na mesma rede, ligados entre si. Assim cada servidor fica responsável por um conjunto das subscrições do

[1] *Peer-to-Peer*

sistema e notificação dos assinantes. Através desta abordagem, o problema da difusão filtrada é particionado em vários sub-conjuntos de assinantes, atribuídos aos *brokers* existentes. Nesse sentido cada *broker* encaminha as notificações para os subscritores alvo ligados a ele, bem como para os outros servidores. Desta maneira, aquando da recepção de uma publicação, cada servidor analisa os filtros das suas subscrições, notificando os nós que aceitam o conteúdo.

O objectivo para uma disseminação filtrada eficaz num conjunto de *brokers* é a inexistência tanto de falsos positivos como negativos. A questão dos falsos positivos neste modelo estará dependente da maneira como as subscrições são atribuídas a cada *broker*. Ou seja, havendo várias subscrições iguais dentro da rede, se estas se concentrarem em menos *brokers*, então a parte do sistema que receberá mensagens específicas deste conteúdo será menor, melhorando a eficiência. O encaminhamento, e consequente desempenho do sistema, estará muito dependente das subscrições atribuídas a cada *broker*, podendo a disseminação tornar-se sistematicamente numa inundação se estas estiverem muito entrelaçadas entre eles.

Uma abordagem alternativa ao conjunto de *brokers* é a descentralização total do processo de disseminação filtrada, tendo em vista a diminuição de problemas de escalabilidade e distribuição da carga do algoritmo. Essa alternativa passa pela definição do problema da disseminação filtrada seguindo o paradigma *peer-to-peer* puro. Nesta abordagem os participantes do sistema têm todos a mesma função, estando ligados entre si, não havendo qualquer controlo central nem servidor. Desta maneira qualquer nó pode publicar novas mensagens, sendo esta informação disseminada a partir deste para os restantes participantes. Cada nó pode também ser subscritor e editor de conteúdo ao mesmo tempo. Tendo todos os participantes as mesmas funções, implicará que os nós interessados em determinada mensagem efectuem trabalho, recebendo-a e disseminando-a para outros possíveis interessados. Se este processo for realizado de forma adequada e tirar partido de uma elevada redundância no sistema, gera-se uma boa distribuição de carga do sistema pelos participantes, bem como a possibilidade de obter melhores mecanismos de tolerância a falhas entre nós. O desafio reside nisso mesmo, conciliar desempenho, correcção e tolerância a falhas, num ambiente caracterizado pelo elevado dinamismo da composição dos participantes do processo.

Esta tese tem como base o estudo dos sistemas Editor/Assinante para disseminação de informação de forma descentralizada com filtragem baseada no conteúdo, focando-se no equilíbrio da carga dentro do sistema. O tratamento desta temática enquadra-se no seguimento do trabalho realizado no âmbito do projecto *LiveFeeds*[2], como a seguir se descreve.

[2]Projecto PTDC/EIA/76114/2006, FCT/MCTES

1.2 Contexto

O projecto *LiveFeeds* tem como intuito a disseminação filtrada de informação, de forma cooperativa entre os utilizadores do sistema descentralizado. De maneira a implementar esta filtragem de forma correcta, cada nó conhece os demais dentro do sistema, sendo denominada esta característica de visibilidade completa para cada nó da rede. A aplicação distribui a carga de trabalho dos seus algoritmos pelos participantes, descentralizando assim o controlo do sistema. Existem também métodos para corrigir e ultrapassar falhas dentro do sistema. Para alcançar estas características, o *LiveFeeds* baseia o seu funcionamento em dois algoritmos: o de filiação e o de difusão. Estes serão descritos de seguida, de maneira a demonstrar que alcançam os atributos referidos previamente e detalhados na secção 2.2.

O algoritmo de filiação do sistema tem como propósito a junção de novos nós ao sistema, dando a conhecer aos participantes informações sobre estes, à medida que chegam. Desta forma, este dará origem à visibilidade completa da rede que cada nó terá, necessária para o correcto funcionamento do sistema *LiveFeeds*.

Os participantes estão divididos em fatias de acordo com os seus identificadores, havendo nós especiais para cada fatia da divisão, que agregam eventos de junção. Estes disseminam os eventos através de árvores aleatórias, de forma a que todos os nós obtenham a informação. Um nó sabe que se juntou ao sistema quando recebe uma mensagem com a indicação da sua chegada. O processo é *best-effort*, podendo existir incoerências na informação de visibilidade total dos nós devido a falhas. Este facto é superado por um processo epidémico em *background*, que faz convergir no tempo visões completas de cada nó.

No algoritmo de difusão de conteúdo a informação é disseminada dentro do sistema para os nós cujos filtros indicam que esta lhes interessa. Este segue uma abordagem semelhante à referida anteriormente, em termos de árvores de difusão aleatórias, contudo contém diferenças em termos de escolha de nós e aceitação das mensagens.

A ideia principal é a divisão da carga de encaminhamento da mensagem pelos nós que a aceitam, recebendo a informação apenas os participantes que declararam interesse na mesma. Com isto pretende-se que a carga seja distribuída dentro do sistema, e que a informação seja eficazmente encaminhada para os participantes que a desejam.

Apesar das boas perspectivas de funcionamento do sistema a partir destes dois algoritmos, este apresenta algumas limitações, principalmente ao nível de carga de trabalho e escalabilidade referente à taxa de entrada no sistema. Após análise do sistema de filiação em [MMDM09], verifica-se que os custos de largura de banda do algoritmo aumentam linearmente com o *churn*[SR06]. Assim, com o aumentar da taxa média de entrada de nós no sistema, os custos de comunicação do *LiveFeeds* crescem para o algoritmo de filiação. Esta limitação também tem em conta o tempo que cada nó esta no sistema, pois quanto menor a estadia na aplicação maior será o impacto na escabilidade do algoritmo.

Este crescimento do dinamismo do sistema e consequente dos gastos do mesmo, não

é escalável para taxas de entrada altas, sendo necessário encontrar mecanismos que mitiguem o problema do *churn*.

1.3 Objectivo

O objectivo deste trabalho é melhorar o sistema *LiveFeeds*, mantendo as suas características de funcionamento, com o intuito de minimizar algumas das limitações referidas. A ideia base é mitigar os problemas de escabilidade através de uma nova abordagem para o problema da filiação, e como esta se irá relacionar com a difusão filtrada. A solução procurará manter as suas propriedades de equilíbrio de carga, bem como correcção(evitando os falsos negativos) e eficiência, não havendo falsos positivos. Adicionalmente, procurar-se-á melhorar o aspecto de tolerância a falhas, mantendo o que já havia sido desenvolvido para o sistema.

Este trabalho pretende desenvolver soluções que exploram duas vias distintas. Na primeira, considerada mais importante, tenciona-se utilizar a redundância dos filtros presentes no sistema para melhorar a escalabilidade do mesmo. Assim, a intenção é mitigar as limitações do algoritmo de filiação, nomeadamente a da escalabilidade referente à taxa de entrada de nós e tempo de estadia no sistema, com uma nova abordagem para este.

A segunda via explorada pretenderá desenvolver mecanismos de tolerância a falhas e dimensionamento do sistema, para solução baseada na redundância dos filtros. Esta terá um papel importante na nova arquitectura desenvolvida pois, neste tipo de sistemas como são as redes *peer-to-peer*, são sempre necessários grandes cuidados com as falhas que ocorrem. Estes mecanismos pretendem reforçar o funcionamento do sistema, tornando-o mais resiliente.

1.4 Contribuições Esperadas

Este trabalho tem variadas contribuições no âmbito da problemática da disseminação filtrada baseada no conteúdo, tendo aplicação no domínio geral dos sistemas Editor/Assinante. Com o desenvolvimento da solução atrás proposta as contribuições esperadas para este trabalho são:

- Uma nova arquitectura baseada em *super nós*, explorando os filtros redundantes no sistema, mitigando o problema de escabilidade referido;
- Mecanismos de tolerância a falhas para a filiação, com a introdução de *super nós* redundantes, e para a disseminação filtrada, em termos da recuperação das falhas existentes;
- Uma avaliação experimental de resultados dos da nova arquitectura e dos mecanismos de tolerância a falhas, estudando o seu comportamento;
- Um protótipo com a implementação das ideias atrás referidas.

1.5 Estrutura do Documento

Este documento estará dividido em seis capítulos apresentados brevemente de seguida.

A primeira parte contém a introdução e motivação do problema, bem como toda a sua base e metodologia de desenvolvimento. Na segunda será descrito o trabalho relacionado referente às temáticas abordadas. O terceiro capítulo englobará toda a descrição do desenho da nova arquitectura para o sistema *LiveFeeds*, baseada em *super nós*. Seguidamente haverá um capítulo que descreve a concretização desse mesmo desenho. No quinto capítulo será desenvolvida uma análise experimental ao sistema, sendo tomadas conclusões sobre este após o estudo dos resultados. O capítulo final englobará a conclusão desta dissertação, enumerando as contribuições da mesma e acabando por referir desenvolvimentos futuros que podem ser feitos ao sistema.

2

Trabalho relacionado

Este capítulo do trabalho relacionado está dividido em três partes distintas, contendo cada uma temáticas importantes para o enquadramento dos temas, e desenvolvimento destes nesta dissertação. Inicialmente será falado do paradigma Editor/Assinantes.

Segue-se uma secção sobre o sistema *LiveFeeds*, pois este vem de encontro às características proporcionadas pela utilização do paradigma anterior. Desta forma, este sistema pretende a disseminação filtrada de conteúdos segundo este paradigma, no sentido de obter correcção e eficácia na entrega de mensagens e notificações. Tal é proporcionado pela implementação de um dos sistemas de encaminhamento descritos na secção sobre o paradigma Editor/Assinante.

A última secção descreve a temática das redes sobrepostas. O sistema *LiveFeeds*, além de implementar a difusão filtrada com as propriedades de correcção e eficácia, pretende a distribuição e equilíbrio de carga de trabalho entre os nós e, se possível, tolerância a falhas dentro do sistema. Estas duas últimas características podem ser alcançadas através de uma total descentralização da aplicação, funcionando o sistema segundo as ideias *peer-to-peer*, organizando-se os nós segundo redes sobrepostas.

2.1 Publicação e subscrição de informação

As redes de publicação e subscrição de informação seguem uma ideologia diferente dos sistemas normais, declarando cada participante o interesse em certo tipo de dados. Denomina-se este paradigma de Editor/Assinante[MD10][EFGK03]. Os conteúdos são enviados directamente para o grupo dos nós que desejam a informação, através de mensagens. Com esta abordagem os utilizadores necessitam apenas de conhecer os dados que desejam e não a localização de onde os poderão obter. Assim desenvolve-se um meio muito mais

simples de disseminação de informação, fazendo-a chegar aos utilizadores, não tendo estes de a procurar em determinado local.

Este tipo de redes servem de base a variadas aplicações como o Scribe[RKCD01] ou o Hermes[PB02], que retiram bastante proveito do seu funcionamento derivado das suas características únicas, tanto de encaminhamento de conteúdos como de organização dos utilizadores do sistema. Para tal, neste tipo de sistemas é fornecida uma *API* que funciona como base para a implementação de aplicações sob estas redes. Esta interface tem, basicamente, métodos de publicação de notificações, subscrição de determinados dados e desinteresse dos mesmos, funcionando e organizando-se o sistema dependendo destas operações chave.

Este tipo de abordagem para a disseminação de informação tem um grande potencial. Utilizando os meios e algoritmos de encaminhamento correctos[MD10], a informação é distribuída eficazmente por quem a deseja, sem sobrecarregar outros nós participantes no sistema. Desta maneira a utilização de redes sobrepostas para a aplicação destas ideias tem fundamento, aproveitando a potencial organização física do sistema, tolerância a falhas e propriedades únicas de pesquisa e encaminhamento que estas fornecem. Mais à frente estas redes serão descritas, abordando estes temas que podem servir de base a sistemas de publicação e subscrição de informação.

2.1.1 Redes Baseadas em Conteúdos

As redes baseadas em conteúdos[CRW04] separam os utilizadores em duas classes distintas, os editores e os assinantes. Os assinantes são aqueles nós que têm interesse em determinados conteúdos, escolhidos a partir de filtragem de informação. Por outro lado os editores são os participantes que geram notificações aquando da chegada de novos dados, de determinados conteúdos, ao sistema. Através desta organização, as mensagens são geradas e distribuídas somente a quem mostrou interesse nestas. Estas são publicadas num espaço próprio de informação e encaminhadas pela rede que suporta o sistema.

Existem três tipos de sistemas em que se publica a informação e esta é recebida pelos subscritores interessados nela, baseando-se nos dados de forma distinta.

Sistemas baseados em tópicos Estes possibilitam aos utilizadores associarem-se com determinados tópicos ou canais, recebendo as notificações dos mesmos. Os editores disseminam nestes informação relevante sobre determinado assunto, recebendo os subscritores dos mesmos essa informação na sua totalidade.

Sistemas baseados em conteúdos Esta abordagem, baseando-se em conteúdos, surge da aplicação de filtros sobre dados disseminados a partir dos canais referidos anteriormente, com intenção de tornar o sistema mais eficiente. Assim pode-se delimitar melhor a informação que se deseja receber, mesmo dentro de determinado tópico, só encaixando algumas mensagens nos filtros definidos para determinado tópico.

Sistemas baseados em canais Neste método de publicação de conteúdo, os dados são distribuídos através de determinado canal de informação, que pode ser subscrito. Esta abordagem pode englobar as duas alternativas anteriores de implementação.

Neste trabalho serão analisadas ao pormenor as redes baseadas em conteúdos, sendo descritas algumas arquitecturas interessantes para a implementação deste tipo de sistemas.

Para um nó receber uma mensagem de conteúdo, esta informação tem que coincidir com um ou vários interesses expressados pelo nó em questão. Esta filtragem é necessária para a manutenção de uma entrega eficaz e sem erros, nas notificações sobre os conteúdos que interessam a determinado nó. Num sistema de grande dimensão, como existem bastantes filtros associados a subscrições, é necessário que a correspondência entre notificações lançadas e subscrições feitas seja feita de maneira eficiente[MFP06]. Tal acontece para não degradar o funcionamento do sistema, tornando-o viável.

A filtragem é feita com base nos atributos disponíveis relacionados com o conteúdo em questão, nas notificações em que estes se enquadram. Desta feita, cada utilizador pode definir um filtro através de intervalos de predicados, informações exactas ou conjunção dos anteriores, em variados predicados. Quando é espalhada uma notificação no sistema, verifica-se se toda ela encaixa na filtragem anteriormente definida por parte do utilizador, só sendo entregue a este se tal acontecer[MFP06]. Assim mostra-se que esse nó estava efectivamente marcado para receber o conteúdo que havia sido disseminado, estando a notificação totalmente contida na filtragem definida.

Assumindo que cada conteúdo específico terá uma panóplia de nós interessados especificamente nele, é necessário encontrar formas de espalhar a informação apenas pelos utilizadores que a desejam, eliminando troca desnecessária de mensagens e chegada de informação a nós que não a desejam. Existe uma notação específica para denominar alguns casos importantes dentro do encaminhamento de informação.

Falso Positivo Os falsos positivos são as chegadas de conteúdo a nós que não o subscrevem, recebendo informação sobre a qual não tinha declarado interesse.

Falso Negativo Os falsos negativos são a ausência de encaminhamento de dados para um nó, ou notificação do mesmo, estando ele interessado no conteúdo que foi disseminado no sistema, não sendo entregue a esse mesmo nó.

Existem variados métodos para disseminar informação como referido, bem como algumas arquitecturas que suportam estas soluções. De seguida serão evidenciadas as mais relevantes para este trabalho, relacionando-as também com as redes sobrepostas que irão suportar a disseminação de informação.

2.1.2 Arquitectura Cliente/Servidor

As redes baseadas em conteúdos podem ser implementadas a partir de uma arquitectura Cliente/Servidor, sendo os servidores denominados *brokers* e os clientes subscritores. Existem duas abordagens para disseminar as notificações a partir dos servidores, a *push-based* e *pull-based*. A ideia da primeira é enviar imediatamente as notificações para os subscritores mal os dados estejam disponíveis, através de *multicast*, se este mecanismo estiver disponível. Porém, no outro paradigma, as notificações são memorizadas nos servidores, permitindo aos subscritores ir buscá-las a qualquer momento, quando necessitam da informação em questão.

Estes *brokers* encaminham as notificações de conteúdo para outros servidores existentes e subscritores dessa informação. Dependendo do tamanho do sistema e conteúdos espalhados, podem ser utilizadas redes de servidores que se ligam entre si como nós *P2P*. Desta maneira, cada um é responsável por determinadas subscrições feitas e pelo encaminhamento da informação no sistema. Assim podem-se criar *clusters* de nós subscritos a determinado servidor, juntando geralmente os utilizadores com interesses comuns nesse mesmo *broker*, facilitando a disseminação da informação. Porém, existem alternativas em que os servidores responsáveis por cada subscrição são os que estão mais facilmente disponíveis.

Os algoritmos normalmente utilizados nesta abordagem são variações de *multicast* entre os grupos de subscritores e os servidores. No entanto, se as subscrições entre os *brokers* estiverem muito entrelaçadas e o número de servidores for baixo, estas técnicas de *multicast* ideal assemelham-se a uma simples inundação da rede com um *overhead* de computação significativo. De notar que em termos de falsos negativos, uma abordagem de inundação é óptima pois entrega as notificações a todos os nós que marcaram o conteúdo, falhando, porém, na questão dos falsos positivos. Em casos de heterogeneidade de subscrições, número de servidores elevado em relação a subscritores e requisitos de latência não muito baixa, esta abordagem é satisfatória dando boas indicações de funcionamento.

Desta forma percebemos que esta arquitectura tem problemas óbvios de escalabilidade em relação ao número de servidores e subscrições. Estes ficam mais evidenciados na problemática da determinação de quais notificações encaixam em que filtros, sendo posteriormente necessário encontrar os nós que fizeram tal subscrição filtrada e encaminhar o conteúdo para eles. Todo este processo é trabalhoso para o sistema, gerando-se uma latência algo significativa que provoca atrasos na entrega de notificações. Tal pode ser ultrapassado por uma avaliação periódica apenas das notificações, gerando desfasamentos entre a chegada do conteúdo e a entrega do mesmo.

2.1.3 Construção de Tabelas de Encaminhamento pelo Caminho Inverso

Este tipo de algoritmos de encaminhamento servem-se da troca de mensagens de interesses em subscrições para criar entradas nas tabelas de encaminhamento dos nós, construindo assim o caminho entre editor e subscritor[MD10]. Assim, as notificações são encaminhadas para os nós referenciados nas tabelas, se estas corresponderem às subscrições associadas às ligações nas tabelas de encaminhamento dos nós por onde a mensagem é propagada[MFP06].

Subscrição de conteúdo A construção e manutenção destas tabelas é realizada através da inundação da rede por parte das subscrições feitas pelos nós. Desta maneira, para cada tipo de conteúdo, são criadas entradas nas tabelas de encaminhamento do sistema que fornecem uma *spanning-tree* partilhada para os editores de cada conteúdo. Para deixar de querer determinado conteúdo um nó inicia a inundação com uma mensagem de desinteresse, sendo o método o mesmo.

Avisos de publicação Este tipo de mensagens são enviadas pelos editores de forma a prevenir a disseminação da inundação de subscrições para zonas desinteressantes, que só aumentariam a latência do sistema. Os avisos são propagados a partir de quem publica conteúdos, da mesma maneira que as subscrições, introduzindo uma complexidade extra no sistema. Este esquema funciona bem nos casos em que existem subscritores de dados que não têm editor no sistema.

Compressão de tabelas de encaminhamento Este método é utilizado de maneira a limitar o espaço gasto em informação de subscrição nas tabelas de encaminhamento, tornando o sistema mais escalavel aquando da existência de muitos interesses. A compressão consiste na verificação de interesses que estão contidos uns nos outros, generalizando a tabela de encaminhamento e entregando as mensagens de maneira óptima. Outro método é a criação de uma entrada a partir da junção de duas condições, gerando um espaço de interesse maior, que pode levar a falsos positivos. É necessária uma grande gestão, tanto de subscrições, como de nós, para conseguir criar eficazmente estes sumários, espalhando-os pela rede de forma correcta. Esta abordagem simplifica o trabalho dos participantes em termos de escolha de nós para enviar a mensagem, aumentando a largura de banda necessária. A figura 2.1 seguinte mostra um exemplo de sumários.

Para um melhor encaminhamento de notificações e subscrições, utilizam-se diferentes organizações entre os nós, criando ligações específicas e pré-determinadas entre eles. Para suportar esta ligação, com critério, entre os mesmos são utilizadas redes sobrepostas, tirando partido das suas características de auto-organização, pesquisa e encaminhamento. Um dos meios utilizados para disseminar conteúdos é a construção de árvores com uma raíz. Esta é uma abordagem hierárquica em que se reduz o tamanho das tabelas de encaminhamento em cada nó, de maneira a tornar o sistema mais eficaz, escalavel e viável.

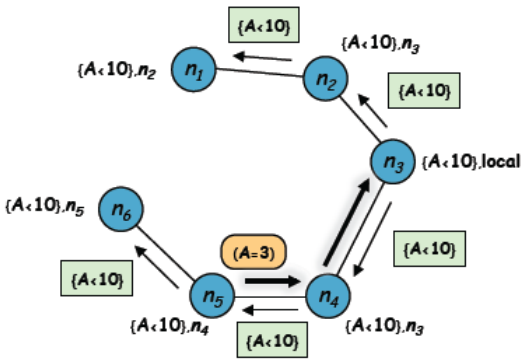


Figura 2.1: Exemplo de sumários e subscrição de conteúdo[MD10]

Nestas árvores as subscrições de cada nó são propagadas para cima nos níveis da árvore até à raíz, guardando em cada nó entradas na tabela de encaminhamento, especificando de onde veio a subscrição. Assim é garantido que o nó raíz conhece todas as subscrições e tem um caminho já definido para o subscritor alvo, através da entradas definidas pelas ligações para baixo na árvore de nós. Desta maneira, as notificações apenas necessitam de ser propagadas para baixo para as sub-árvores que têm subscrições desse conteúdo. Quando um nó, que não a raíz da árvore, possui uma nova notificação, este pode enviar a mesma para o nó raíz, que dissemina de seguida a informação como anteriormente foi referido. Assim se verifica que a árvore pode ser percorrida nos dois sentidos, conforme os propósitos da aplicação.

Esta topologia gera diferenças de carga em cada nó, visto a raíz conhecer todas as subscrições, sendo que os outros nós conhecem um número menor. Isto deve-se à organização hierárquica imposta, criando um ponto de falha único no nó raíz, quando o sistema cresce. Tal acontece pois a complexidade de intersecção entre subscrições e notificações aumenta com o número de entradas nas tabelas de encaminhamento, tendo que testar mais casos.

Uma das formas de evitar este ponto de falha único e a distribuição desigual da carga é utilizar múltiplas árvores, juntando as notificações de alguma forma, gerando uma árvore para cada conjunto destas. Outra forma de solucionar este problema é criar uma *spanning-tree* para cada editor, funcionando o sistema como descrito anteriormente. Nesta organização, cujo intuito é a disseminação de conteúdo, é necessário haver mecanismos de detecção de ciclos na rede, de maneira a evitar inundações da mesma. Tal é feito a partir da escolha do caminho mais pequeno de determinada subscrição até ao nó.

Existem duas maneiras diferentes de organizar este tipo de redes estruturadas em árvore: *mesh-first* e *tree-first*.

Mesh-First Neste tipo de sistemas a rede é estruturada como um grafo, podendo os nós enviar mensagens entre eles e para o nó raíz, baseando-se em chaves para o fazer.

Tree-First É uma abordagem em que a árvore de disseminação é logo construída, sendo necessária uma constante reestruturação e manutenção da mesma para a manter funcional.

Este algoritmo descrito dissemina os conteúdos de forma óptima, utilizando todos os métodos anteriores para aumentar a eficácia do algoritmo e diminuir a sua complexidade em termos de tempo e espaço.

2.1.4 Mapeamento em Chaves

As redes sobrepostas estruturadas têm uma base de organização, escalabilidade, encaminhamento e manutenção que servem os propósitos de uma implementação dos sistemas de publicação e subscrição de conteúdos. Nesta abordagem de mapeamento de chaves, são atribuídas chaves únicas a cada nó, de modo a tornar a organização e o encaminhamento eficazes a partir destes identificadores. Desta forma, para utilizar este tipo de redes como base para um sistema de publicação e subscrição, é necessário o mapeamento dos intervenientes do sistema em identificadores, como discutido em [BMVV05].

Tal é feito a partir de funções que mapeiam, tanto subscrições, como notificações, para chaves únicas, de maneira a garantir que cada tipo destas mensagens tem uma chave diferente. O uso destes identificadores é simples, visto que se encontram as subscrições de cada notificação nas chaves únicas obtidas a partir destas. É necessário evidenciar que cada nó é responsável por um espaço de chaves perto deste, gerindo as informações das chaves em redor do mesmo. Assim, se as subscrições estiverem registadas nos nós com as chaves correspondentes, as notificações que servem esses interesses são encaminhados para essa chave, sendo de seguida enviadas para os nós que as submeteram.

Desta maneira, as subscrições estão ligadas a um espaço de chaves que representa os seus interesses, estando interesses comuns juntos na mesma chave. Por seu lado, as notificações que satisfazem certas subscrições, são mapeadas para os mesmos nós das subscrições, recebendo aí a informação de que nós desejam esse conteúdo. Assim, encaminham as mensagens até esses nós através das funcionalidades da rede sobreposta utilizada. O mapeamento entre chaves, envio de notificação e posterior encaminhamento para os subscritores está evidenciado na figura 2.2.

Para evitar erros na entrega de mensagens, as chaves têm que ser bem calculadas de forma a garantir uma entrega fiável aos destinatários. A escolha destas chaves é feita a partir dos atributos relacionados tanto com notificações como subscrições. É relativamente simples criar funções para mapear atributos de igualdade em chaves, contudo, os atributos de intervalos são mais dificilmente geridos, tendo que ser manipulados para criar chaves correctas. De maneira a transformar estes intervalos em chaves únicas fiáveis, é necessário dividir o índice e os atributos em questão. Assim, são criados intervalos específicos para os atributos em que estes são utilizados, havendo uma correspondência de chaves para cada intervalo. Isto leva a que o índice de uma subscrição ou notificação

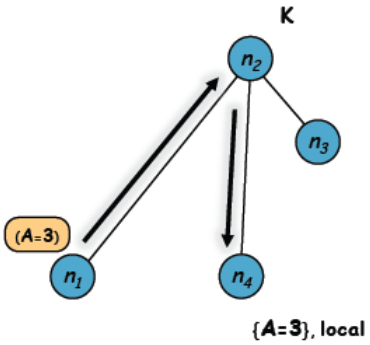


Figura 2.2: Mapeamento e encaminhamento de notificação[MD10]

se transforme num conjunto de chaves relacionadas, identificando o mesmo espaço de subscrições. Esta questão é abordada em [Aek06].

Esta abordagem pode trazer, não só uma grande diferença entre as cargas de cada nó, mas também alguma dificuldade de entrega aos nós certos, visto se criarem situações em que são feitas generalizações de chaves sob sumários já existentes, introduzindo falsos positivos. Para equilibrar a carga, utilizam-se métodos de reconstrução da rede de acordo com as responsabilidades de cada nó, transferindo-se algumas de maneira a tornar o mais equivalente possível o trabalho de cada elemento no sistema. Tal torna a rede mais escalavel, necessitando de um bom mecanismo de criação e gestão de chaves para não introduzir erros ou *overhead* em certos nós muito congestionados.

A entrega de mensagens, após o algoritmo de descoberta de quais notificações satisfazem que subscrições, a partir do mapeamento de chaves, é feita através do nó que publica a mensagem para todos os nós que manifestaram interesse nesse conteúdo. Isto é obtido através de um método de envio *multicast* para grupos, garantindo a entrega eficaz de conteúdos, podendo funcionar optimamente.

2.2 LiveFeeds

O *LiveFeeds* é um projecto que visa a disseminação cooperativa de informação numa rede de computadores. Tendo como âmbito um sistema Editor/Assinante, o sistema *LiveFeeds* pretende a disseminação filtrada de informação dentro de uma rede, abordando este problema de forma distribuída[MMDM09]. Cada nó tem um filtro específico de maneira a receber os conteúdos correspondentes a esse. Utilizando uma abordagem *peer-to-peer*, pretende-se a distribuição da carga total do sistema pelos seus participantes, ao encaminhar e avaliar os filtros apenas nos nós interessados nesses dados específicos.

O objectivo da difusão filtrada no *LiveFeeds* é a não obtenção de falsos positivos, mas sobretudo a não ocorrência de falsos negativos em todos os participantes do sistema. De maneira a implementar uma solução com estas características, o sistema necessita que todos os nós tenham visibilidade completa do sistema com base em [GLR03], ou seja,

conheçam todos os nós restantes e os filtros correspondentes. Para alcançar estas características de funcionamento, o *LiveFeeds* é constituído por dois algoritmos: o algoritmo de filiação de nós e o de difusão de informação. Estes serão descritos de seguida, demonstrando como e através de que mecanismos, os requisitos do sistema são alcançados.

2.2.1 Algoritmo de Filiação

Este algoritmo tem como propósito a junção de novos participantes ao sistema, dando a conhecer aos demais informações sobre estes[MMDM09]. Para atingir este propósito, o sistema tem algumas características especiais, seguindo um algoritmo muito especifico para a disseminação da informação de filiação, ambos explicados de seguida.

Começando pela organização, cada participante é denominado um nó do sistema cooperativo. Como já foi referido anteriormente, cada nó tem um filtro específico de acordo com o conteúdo que deseja receber. Cada nó tem um identificador único no sistema, um endereço específico e o filtro por si definido. Seguidamente o conjunto de identificadores de todos os nós do sistema é dividido em fatias, indicando o nó com o identificador numericamente menor como *slice leader*, sendo o responsável pela fatia em que está inserido. De notar que não existe coordenação entre *slice leaders* e que se supõe que existe conectividade entre todos os participantes.

O algoritmo de filiação baseia-se nesta organização entre os nós para desempenhar o seu propósito. Para um novo nó entrar no sistema, este tem de conhecer um participante deste, comunicando-lhe que deseja juntar-se à rede. De seguida, o nó participante comunica ao *slice leader* da fatia de identificadores correspondente à chave do novo nó, o pedido de junção por parte de um novo participante. Cada *slice leader* agrega pedidos de adesão ao sistema por um determinado tempo ou até agregar um número suficiente de pedidos. Tal é feito para reduzir o *overhead* associado aos cabeçalhos das mensagens trocadas.

Depois deste processo de pedido de adesão e agregação, começa a distribuição das novas entradas pelo sistema. Esta informação é espalhada pelo sistema através de árvores de difusão aleatórias. O *slice leader* que inicia o processo de difusão, pelo temporizador de agregação ter acabado ou por já ter 30 pedidos agregados, dividindo o espaço de identificadores em intervalos da mesma cardinalidade, ou seja, intervalos com o mesmo número de nós. Em cada intervalo é escolhido um nó aleatoriamente, enviando-lhe o evento de filiação juntamente com um sub-intervalo do qual o nó passa a ser responsável. Este trabalho é recursivo, tornando-se os intervalos cada vez mais pequenos em número de nós, chegando ate às folhas da árvore, acabando a disseminação destas entradas. Cada novo nó sabe que entrou no sistema ao receber uma mensagem da sua própria junção. Este método está exemplificado na figura 2.3.

Através destas árvores de difusão, os eventos são espalhados pelo sistema através de um método *best-effort*. Contudo podem existir falhas no sistema ou acessos concorrentes, que provocarão informação de filiação incompleta em alguns nós. Sendo pretendido que

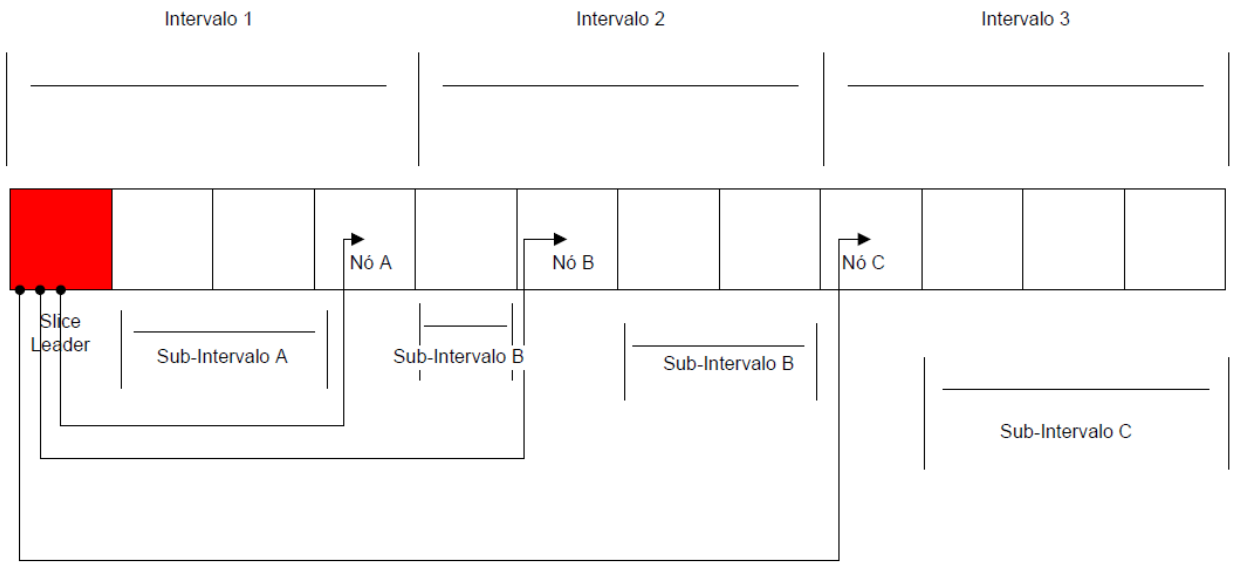


Figura 2.3: Algoritmo de filiação

todos os nós conheçam os restantes para o bom funcionamento do *LiveFeeds*, foi implementado um processo epidémico em *background* em que os participantes trocam as suas vistas sobre os eventos de filiação recebidos. Através das comparações destas mesmas vistas, um nó pode verificar que falhou um evento, adquirindo-o no momento. Estas vistas são estampilhas vectoriais, com a informação concreta dos eventos obtidos por determinado nó.

Neste sistema epidémico, os nós comunicam ponto a ponto de maneira a avaliar as diferenças entre as próprias vistas do sistema. Caso seja encontrada alguma diferença, o nó com a vista atrasada receberá os eventos de entrada em falta enviados pelo outro nó, fazendo que os nós que se contactam entre si, fiquem com as vistas do sistema iguais. A escolha do nó que será contactado é aleatória dentro dos conhecidos no sistema. Os contactos ponto a ponto por parte de um nó estão distanciados temporalmente em medidas variáveis, consoante a carga de dados que consumiram nas reparações e tempos aleatórios de distanciação dos contactos, que fazem parte do algoritmo.

Para que a característica de visibilidade completa se mantenha actualizada, providenciando maior eficácia e desempenho no algoritmo, é necessário que todos os nós presentes no sistema conheçam aqueles que sairam do mesmo. Tendo em conta que num sistema bastante dinâmico haverá muitas entradas, mas também muitas saídas de nós, torna-se imperativo um sistema de notificação de saídas que funcione tão eficazmente como a filiação de novos nós.

Desta maneira a disseminação de eventos de saída do sistema é feita de maneira similar às entradas. Quando um nó não consegue comunicar com outro, detectado que este já não está no sistema, agrega esta saída na sua lista. Após juntar um número de eventos de saída significativos, começa a disseminação destes eventos tendo por base o algoritmo descrito anteriormente para as entradas. Assim as saídas do sistema são consumidas como as entradas, podendo também os *slice leaders* enviar eventos de saída, caso tenham agregado alguns.

A única diferença entre a disseminação de eventos de entrada ou saída é que os de saída começam a ser enviados por qualquer nó, enquanto que os de entrada apenas pelos *slice leaders* correspondentes à fatia onde o novo nó se deverá inserir.

2.2.2 Algoritmo de Difusão

Estando já explicado como a informação de filiação é adquirida, construindo um sistema com visibilidade completa que o algoritmo de difusão utiliza para espalhar os conteúdos no sistema[MMDM09]. O objectivo deste algoritmo é encaminhar as informações ou notificações de actualização exclusivamente para os nós que mostraram interesse nesses dados, através dos filtros definidos. A ideia de implementação é basicamente a utilizada no algoritmo anterior, utilização de árvores de difusão e distribuição da carga, mas existem algumas alterações necessárias.

O conceito fundamental é a distribuição da carga de difusão do conteúdo na árvore, proporcionando trabalho apenas aos nós cujo filtro aceita esse mesmo conteúdo. Assim, aquando da chegada de novo conteúdo, o editor de conteúdo divide o espaço de identificadores do sistema em intervalos da mesma cardinalidade do número de nós.

De seguida, os nós de cada sub-intervalo gerado são analisados sequencialmente até que seja encontrado um que aceite o conteúdo de acordo com a avaliação do seu filtro. Esta análise é feita em cada sub-intervalo. Se um nó aceitar o conteúdo, este fica responsável pela análise do restante sub-intervalo de nós restante, repetindo-se este processo em todos os intervalos gerados, bem como recursivamente.

Como a visibilidade de cada nó pode ser incorrecta por estar atrasada temporalmente, existe uma tarefa epidémica de recuperação de eventos de filiação perdidos, de modo a evitar a comunicação de nós com vistas atrasadas. Ao aplicarmos este algoritmo, caso um nó com uma vista atrasada receba o evento e tenha que avaliar o resto do seu intervalo, poderá não ter informação de alguns nós que já estão no sistema e que receberiam esse evento. Tal situação geraria falsos negativos, ou seja, nós que não estão na vista do avaliador mas que aceitam a notificação não a receberiam.

Para colmatar este problema, quando um nó está preparado para receber um evento, o estado da sua vista é avaliado. Se a vista estiver em dia, este continua o algoritmo como foi descrito anteriormente. Caso isso não aconteça, o nó recebe o evento, mas informa a quem lho enviou que tem a vista atrasada, continuando o nó anterior o algoritmo que já estava a realizar. Esta avaliação é sempre realizada, de maneira a evitar os falsos

negativos no algoritmo de difusão.

A árvore gerada por este algoritmo é aleatória pois é inserido um deslocamento inicial ao computar os sub-intervalos de identificadores. Esta difusão de conteúdo está representada na figura 2.4, onde um intervalo é percorrido e os seus nós avaliados. Um desses nós aceita o evento e continua o posterior calculo do restante sub-intervalo.

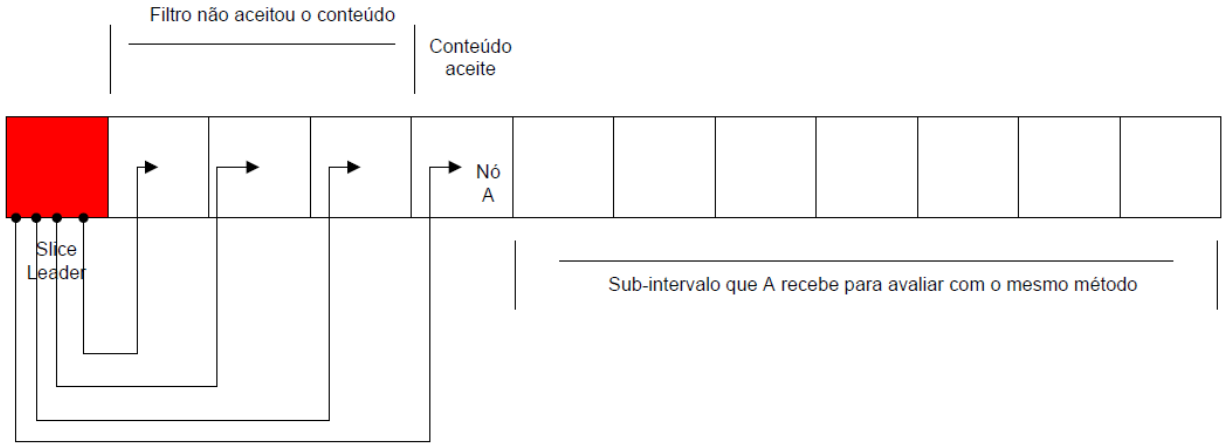


Figura 2.4: Algoritmo de difusão

Com este algoritmo de difusão, todos os nós recebem apenas o conteúdo que desejam receber, obtendo sempre a informação que é aceite pelo seu filtro, evitando os falsos positivos e, sobretudo, os falsos negativos.

Desta forma a única possibilidade de falsos negativos na entrega dos eventos surge na falha, ou saída, do nó que está a avaliar um intervalo especifico. Caso um nó avaliador falhe, todo o seu intervalo será descartado da disseminação do evento, podendo gerar um número de falsos negativos dependendo da popularidade do evento e tamanho do intervalo descartado.

2.2.3 Limitações

Apesar das boas perspectivas de funcionamento do sistema a partir destes dois algoritmos, este apresenta algumas limitações, principalmente ao nível da carga de trabalho e escalabilidade, referentes à taxa de entrada no sistema. Após análise do sistema de filiação em [MMDM09], verifica-se que os custos de largura de banda do algoritmo aumentam linearmente com o *churn*. Assim, com o aumentar da taxa média de entrada de nós no sistema, os custos de comunicação do *LiveFeeds* cresceram devido ao algoritmo de filiação utilizado para comunicar os nós recém chegados. Desta forma, o aumento de custo do sistema não depende da dimensão do mesmo, mas sim do seu dinamismo.

Esta limitação também tem em conta o tempo que cada nó está no sistema. Quanto

menor a estadia na aplicação maior será o impacto na escabilidade do algoritmo. Tal acontece pois, estando pouco tempo no sistema, a sua presença não compensará os elevados custos de filiação para o mesmo, provenientes da sua entrada. Este crescimento não é escalavel para taxas elevadas que poderão ocorrer em sistemas reais, sendo necessário procurar soluções que mitiguem estas limitações impostas pelo funcionamento dos algoritmos.

2.3 Redes Sobrepostas

Uma rede sobreposta é uma forma de organização entre vários computadores, desenvolvida e implementada em cima de uma rede já existente sem suportes físicos explícitos. O seu impacto na Internet e algumas das características que vão ser referidas de seguida estão descritas em [CLB+06]. Os nós estão ligados entre si através de ligações virtuais dependendo da rede em questão. Estas ligações são normalmente conexões ao nível de transporte, tipicamente implementadas em *TCP/IP*, sendo cada nó um processo da aplicação distribuída que se pretende implementar. Estes trabalham todos em conjunto com o intuito de cumprir os objectivos do sistema.

Sendo uma rede sobreposta implementada por cima de uma rede física existente, sustenta algumas das suas funcionalidades nessa mesma rede física, como por exemplo o encaminhamento ponto-a-ponto. Estando um sistema implementado na rede física, pode solucionar alguns problemas desta, desenvolvendo protocolos por cima da mesma, tornando possível estendê-la com diversas funcionalidades ao nível da rede sobreposta. Exemplo disto é a implementação *multicast* em grande escala através de redes sobrepostas, como utilizado no sistema Scribe[RKCD01].

Desta maneira estas redes fornecem outro tipo de funcionalidades e sistemas que a rede física não suporta, desenvolvidas com a vantagem de não alterar a rede sobre a qual está implementada[CLB+06]. Assim as limitações da rede física podem ser superadas por estes sistemas, não sendo necessários encargos extra para modificar a rede sobre a qual o sistema está implementado. Estas funcionalidades novas podem ser utilizadas através de *middlewares* exportados pelos sistemas, tornando bastante benéfica a utilização de redes sobrepostas. Exemplos destes sistemas são o Freenet[CSWH01] ou sistemas baseados na DHT[1] Chord[SMLN+03].

Porém devido ao número de participantes no sistema e a heterogeneidade entre eles (*firewalls*, utilização de NAT,etc.), a gestão destas redes tende a ser complexa. Esta heterogeneidade traz também um *overhead* à rede principalmente devido à divergência entre os caminhos na rede sobreposta e as ligações da rede física. Os nós não têm a visão total da rede, desconhecendo o melhor caminho para determinado nó, ignorando também a organização da rede física sob a qual está implementada a rede sobreposta. Isto causa o desfasamento entre as ligações das duas redes, podendo a comunicação entre dois nós adjacentes fisicamente passar por vários nós virtualmente. Isto pode implicar variadas

[1] *Distributed Hash Table*

passagens pela mesma ligação entre dois nós na entrega de apenas uma mensagem. Assim o caminho físico correspondente ao da rede virtual é muito maior, existindo uma má gestão do encaminhamento de mensagens e percorrendo cada mensagem um caminho bastante maior do que o óptimo.

Assim a latência depende do grafo formado pela rede sobreposta e a sua distanciação ao grafo da rede real. Para um melhor comportamento do sistema é necessário optimizar e assemelhar as ligações virtuais às físicas, diminuindo a latência da rede. De maneira a se compreender melhor esta problemática sugere-se a consulta da figura 2.5 onde é ilustrada não só uma rede sobreposta, mas também a diferença nítida entre as ligações físicas e virtuais, dando origem ao problema de encaminhamento descrito. A rede sobreposta está a azul escuro, e as suas mensagens a preto, e a rede física em azul claro e as mensagens correspondentes a vermelho.

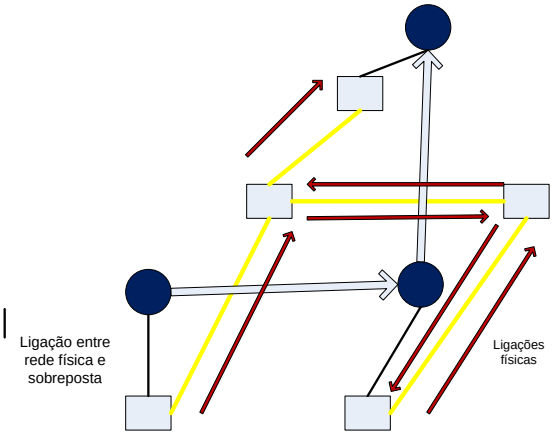


Figura 2.5: Diferença de caminhos entre rede sobreposta e rede física

Existem dois tipos de redes sobrepostas: as estruturadas e não estruturadas, sendo divididas de acordo com a existência, ou não, de uma ideia prévia de organização entre os nós dentro desta rede.

2.3.1 Não Estruturadas

2.3.1.1 Estruturação

As redes sobrepostas não estruturadas não têm qualquer critério específico na ligação entre os nós do sistema sendo estas casuais ou criadas aleatoriamente[Tar10][CCR05][LCC+02]. Desta maneira cada nó pode escolher ou alterar os seus contactos dentro do sistema aleatoriamente, não criando nenhuma organização especial entre eles. Baseando-se nisto, este tipo de redes é facilmente construído, bastando ao nó que se junta a esta a escolha de alguns contactos dentro do sistema. Tal escolha é feita aquando da sua entrada na rede, através de outro nó de que este tenha conhecimento.

Assim se percebe que a ligação entre os nós é oportunista de acordo com o momento e situação concreta do sistema em questão. Desta condição deriva a construção de um grafo aleatório para a representação dos nós e ligações entre eles, sendo estas meramente ocasionais de determinado conjunto de situações. Esta abordagem não estrutural traz vantagens e desvantagens à rede em si, discutidas em pormenor adiante.

2.3.1.2 Manutenção

Como a conexão entre os nós da rede é casual, torna-se evidente que não existe ligação entre os conteúdos geridos por cada nó dentro da rede. Desta forma a organização do sistema é flexível pois o estabelecimento de ligações entre os nós é fácil, sem custos adicionais, o mesmo acontecendo com a reparação de ligações ou alteração das mesmas. Assim torna-se simples manter o estado da rede pois geralmente não exibe uma topologia regular, podendo um nó ligar-se a qualquer outro conhecido em caso de perda de contactos.

Nesse sentido o sistema apenas garante a manutenção de um certo número de contactos para cada nó, de maneira a que a comunicação dentro da rede seja possível. Os contactos apenas são alterados em caso de falhas ou eventos de saída e entrada no sistema. A escolha de nós substitutos é trivial, não se baseando em qualquer critério, nem tendo custos significativos de procura de um nó substituto, sendo utilizado o participante mais facilmente encontrado.

Sintetizando, não é mantido qualquer estado sobre a rede sem ser as ligações necessárias para cada nó. Assim o *overhead* de manutenção do funcionamento da rede torna-se praticamente inexistente trazendo, porém, problemas ao nível de procura e encaminhamento.

2.3.1.3 Pesquisa/Encaminhamento

Contudo esta simplicidade da estruturação sem critério do sistema traz, inevitavelmente, variados problemas pela não organização da rede. Visto a informação disponível para cada nó ser apenas local sobre alguns contactos dentro do sistema, encontrar caminhos

óptimos ao nível global da rede tende a ser complexo. Assim, neste tipo de redes, os caminhos encontrados tendem a ser piores que o caminho óptimo desejado, aumentado a troca de mensagens e latência dentro do sistema. Assim conclui-se que a latência e escalabilidade destas redes dependem do grafo, tipicamente aleatório, criado pelas ligações entre os nós.

O outro grande problema destes sistemas é a pesquisa de informação dentro de uma rede não estruturada[CCR05][LCC+02]. Não havendo qualquer informação sobre a estrutura da rede, o custo de cada pesquisa tende a ser elevado, sendo necessário utilizar várias técnicas para minimizar o número de mensagens trocadas na rede. Assim é importante inserir limitações de mensagens trocadas e caminhos percorridos, de maneira a que a rede não fique bloqueada com o enorme tráfego proveniente de mensagens trocadas aquando de uma pesquisa. De notar, que por ser totalmente alheio ao nó tanto a estrutura do sistema como a existência do que procura, ou mesmo onde procurar, torna-se difícil de escolher a melhor abordagem para a resolução do problema da pesquisa.

Existem essencialmente três tipos de pesquisa[Tar10] que se podem realizar em redes sobrepostas não estruturadas, a pesquisa por inundação, os passeios aleatórios e as pesquisas informadas.

Pesquisa por Inundação[CCR05][LCC+02] Nesta pesquisa são espalhadas mensagens por toda a rede de maneira a retornar o resultado desejado da procura. Desta maneira a mensagem de pesquisa chega a todos os nós da rede, sendo necessário um *TTL*[2] e a noção do caminho mais recente da mensagem, para evitar ciclos intermináveis de mensagens. Neste tipo de pesquisa é sempre encontrado o resultado pois todos os nós são visitados, custando ao sistema uma enorme troca de mensagens entre todos os nós, aumentando em muito a latência inerente ao sistema. Pode também ser utilizada uma técnica de inundação gradual da rede, de maneira a que se o objecto for encontrado se poupe a exagerada e desnecessária troca de mensagens[LCC+02]. De seguida a imagem 2.6 esclarece como é feita uma inundação numa rede.

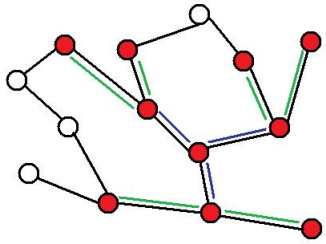


Figura 2.6: Inundação da rede

[2] *Time to live*

Passeios Aleatórios[CCR05][LCC+02] Os passeios aleatórios são um tipo de pesquisa diferente pois um nó envia a pesquisa apenas para outro, e não para todos como na inundação. Isto acontece em cada nó que recebe a pesquisa até acabar um *TTL*. Dependendo do *TTL*, a taxa de sucesso será maior ou menor, tendo em conta também o número de passeios aleatórios utilizados. Esta abordagem não garante que a pesquisa encontre o resultado pretendido dentro do sistema, pois não passa por todos os nós, garantindo, porém, um gasto muito menor de mensagens trocadas no sistema. Na imagem 2.7 aparece um exemplo deste tipo de pesquisa.

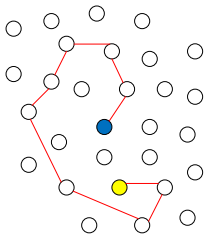


Figura 2.7: Passeios aleatórios dentro de um grupo de nós

Pesquisas Informadas As pesquisas informadas[BCMR04] consistem no envio da mensagem de pesquisa para um nó escolhido com base em alguns índices desenvolvidos sobre a localização de certos recursos. Assim é escolhido um nó que seja potencialmente bom para seguir com essa pesquisa com base nas informações guardadas. Porém este tipo de procura necessita de outro tipo de manutenção de dados do que a falada anteriormente, visto associar alguns tipos de índices de certos recursos a cada nó, aumentando a informação que necessita de estar a ser sempre gerida. Utilizam-se *bloom filters*[GH06] de maneira a guardar estes índices relevantes sobre conteúdos. Estes são compactos em termos de ocupação de espaço, utilizando técnicas de *hash* para formar conjuntos onde se possa testar a presença de determinado conteúdo, com uma baixa possibilidade de falsos positivos. Existe uma versão atenuada destes, optimizando a *performance* dos mecanismos de localização do conteúdo desejado. Estes são vectores de *bloom filters* com uma certa profundidade e tornam possível a escolha em cada nó da ligação mais adequada para o encaminhamento de cada pesquisa. O sistema Freenet é uma rede sobreposta com uma ideia muito vaga de estruturação, em que são utilizadas estas pesquisas perante os dados replicados deste sistema.

Em síntese, nota-se um claro paralelo entre o número de mensagens trocadas, latência do sistema e o sucesso de cada pesquisa. Com a maior troca de mensagens, a taxa

de sucesso aumenta, aumentando proporcionalmente o *overhead* do sistema. Através dos métodos que trocam menos mensagens há menos latência na rede, comprometendo, porém, um pouco a elevada taxa de sucesso das inundações[CCR05][LCC+02].

De notar também a grande importância da utilização de mecanismos de *TTL*[LCC+02], bem como guardar os caminhos das mensagens, de maneira a impedir ciclos infinitos dentro da rede que a iriam inundar de mensagens, aumentando em muito a latência.

2.3.1.4 Tolerância a Falhas

A simplicidade inerente à organização da rede é uma grande vantagem, facilitando a recuperação de falhas ou erros, sendo trivial retomar os pontos de partida de organização da rede[Tar10]. Aquando de uma falha, a reestruturação do sistema é imediata, sem grandes custos para a rede nem troca excessiva de mensagens dentro da mesma. Basta um mecanismo simples de troca de mensagens *keep-alive* entre os nós e os seus contactos para detectar uma falha. Esta não necessita de ser comunicada a outros participantes, fazendo com que o nó em questão a resolva apenas adicionando outro nó aleatório à sua lista de contactos.

2.3.2 Redes Sobrepostas Estruturadas

2.3.2.1 Estruturação

As redes sobrepostas estruturadas seguem uma ideia de garantir uma certa organização e estruturação da rede, de modo a tirar partido dessas características para o seu funcionamento[Tar10]. Continuam a ser um sistema criado por cima de uma rede física já existente, com o intuito de unir os utilizadores numa só aplicação. No entanto, existe uma ideia pré-definida sobre como esta rede se deve organizar e de que modo comunicarão os nós entre si. Nestas redes cada nó tem um identificador único, sendo o estabelecimento da topologia guiada a partir destes identificadores, tendo cada uma a sua função dentro do sistema. Esta divisão em identificadores é feita por determinado tipo de *hashing* de acordo com o algoritmo utilizado e o espectro de identificadores desejado. Assim as ligações entre cada nó do sistema correspondem a uma certa topologia que advém do algoritmo utilizado e identificadores definidos, de modo a manter essa estrutura consistente, funcional e eficiente de acordo com a sua utilização.

Desta forma o sistema tem uma organização topológica definida sob a qual poderá suportar algumas das suas funcionalidades. A topologia pode variar bastante, sendo normalmente utilizadas estruturas em forma de anel, árvores, hipércubos, grafos de de Bruijn, ligando-se os nós propositadamente para a formação desta. Cada nó tem um identificador único no sistema, guiando a sua organização, e proporcionando funcionalidades de pesquisa mais à frente descritas. Esta organização constante de todo o sistema traz vantagens para o desenvolvimento de aplicações sobre este tipo de redes, tomando como pressuposto o conhecimento geral de como estarão os nós organizados. Alguns exemplos de topologias adoptadas por redes estruturadas são mostrados na imagem 2.8.

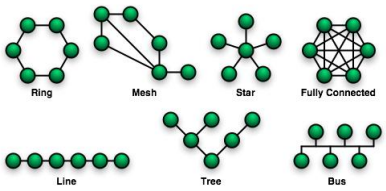


Figura 2.8: Topologias possíveis em redes estruturadas[Wik10]

Uma característica significativa destas topologias é a já falada divisão total dos nós do sistema por identificadores únicos, criando a noção de vizinhança para cada nó em relação aos nós com chaves mais próximas deste. Através desta noção de vizinhança para cada nó, poderemos desenvolver certos tipos de pesquisa informada e tolerância a falhas. Para tal, cada nó terá um sucessor e predecessor na lista de identificadores de acordo com a topologia adoptada, bem como outro tipo de contactos de longa distância dependendo do encaminhamento desejado.

As DHT's são um caso especial de redes sobrepostas estruturadas em que os conteúdos partilhados por cada nó são distribuídos inteiramente pelo sistema. Estas estruturas focam-se no problema de pesquisa e encaminhamento de informação, resolvendo-o da maneira mais simples e eficaz possível, não denegrindo a consistência da estrutura. Para a manutenção dos conteúdos no sistema é necessária a replicação de dados de maneira a obter redundância dos mesmos, aumentando a disponibilidade de conteúdos do sistema.

2.3.2.2 Manutenção

As redes sobrepostas requerem bastantes cuidados em termos de manutenção tanto de conteúdos como da estruturação. Desta maneira a informação sobre os contactos de cada nó é bastante precisa devido à topologia definida e necessita de ser verificada e actualizada em casos de falhas e entradas ou saídas da rede. Assim mantém-se o sistema sempre com as características desejadas, gastando porém alguma largura de banda para o fazer. Para o funcionamento correcto destas redes é crucial que a estrutura se mantenha consistente.

Nas DHT's é necessária a replicação dos conteúdos de cada nó pelo sistema de maneira a aumentar a disponibilidade dos mesmos no sistema. Aquando de entradas e saídas no sistema é utilizada uma política de transferência dos dados dos nós em questão, para os preservar no sistema, bem como replicação garantindo a existência de várias réplicas de cada objecto. Esta redundância dos dados mantém-nos sempre activos perante situações de desconexão ou falhas, sendo a gestão de toda a replicação dos mesmos e manutenção do grau de redundância, um gasto de mensagens significativo.

Para auxiliar estes processos anteriores são necessários também mecanismos que detectam se um nó está ligado, de modo a desencadear as acções necessárias previstas pela arquitectura adoptada de acordo com a situação de desconexão. É de notar que toda

esta manutenção é feita pelo algoritmo do sistema, não sendo necessária qualquer intervenção do utilizador da aplicação. Desta maneira a falha de um nó pode ser detectada através de *pings* aquando do contacto com o mesmo, ou por tarefas em *background* que testam, de tempo a tempo, se cada nó está ligado. Estas abordagens para detecção de falhas causam um *overhead* considerável no sistema devido ao constante teste de cada nó dentro da rede. Este aumenta, em caso de falha, devido à reestruturação do sistema de maneira a manter a topologia adoptada.

2.3.2.3 Pesquisa/Encaminhamento

Com estas topologias bem definidas, as pesquisas por identificador podem ser bastante mais fáceis de efectuar, visto cada membro ter noção da estruturação do sistema e de que maneira chegar a determinado nó[CDK05]. Este tipo de estruturas direccionadas para as pesquisas denominam-se DHT's. Com estruturação específica dos nós do sistema através dos seus identificadores, existe, à priori, uma estratégia para chegar a um dado identificador de um nó dentro da rede. Assim é possível realizar pesquisas por identificadores com um encaminhamento eficiente e determinista, tirando partido da organização dos nós e respectivos contactos. O encaminhamento é realizado através de uma reduzida troca de mensagens com a noção de vizinhança entre nós.

Contudo verifica-se que o encaminhamento das pesquisas depende, não só do conhecimento de cada nó sobre outros no sistema, mas também das características da topologia em si. Através destas é proporcionada uma maneira mais fácil, ou difícil, de chegar a determinado nó, dependendo das topologias adoptadas. As pesquisas tendem a ser bastante mais baratas, em termos de custo de mensagens, do que as referidas nas redes não estruturadas. Em relativamente poucos passos, comparando com o tamanho do sistema, uma pesquisa é normalmente satisfeita. Esta é uma clara vantagem destes sistemas que funcionam como um *middleware* entre os nós para o encaminhamento eficiente de mensagens de pesquisa. Este *middleware* disponibiliza uma API reduzida de pesquisa de conteúdos, determinando uma boa utilização por parte dos membros da aplicação e garantindo a manutenção da organização pré-definida da arquitectura da rede[CDK05].

As pesquisas dependem bastante do algoritmo implementado e grafo formado, bem como do nível de informação guardado e mantido em cada nó do sistema. Existe sempre uma relação entre pesquisas mais baratas e o custo custo mais elevado devido ao *overhead* de manutenção de dados. A cada passo do encaminhamento de uma pesquisa, esta fica mais próxima do seu nó de destino. Tal deve-se ao facto de que quando um nó recebe uma pesquisa, encaminha-a sempre para o nó que conhece mais perto do seu destino, caso este não seja o destinatário. Assim é sempre escolhido o melhor contacto conhecido para cada pesquisa, sendo este o mais perto do recurso desejado.

Sintetizando, o conhecimento total da estruturação do sistema, manutenção de informação relevante em vários nós sobre os dados e redundância destes tornam as pesquisas neste tipo de sistemas muito eficazes, rápidas e baratas.

2.3.2.4 Tolerância a Falhas

Outra das vantagens destas redes é a auto-organização e reparação destas perante falhas ou novas entradas e saídas de membros. Apesar de mais dispendioso do que nas redes não estruturadas, visto ser obrigatório corresponder a uma determinada organização, existem métodos que tornam a gestão destes eventos eficientes, sem qualquer participação dos utilizadores ou servidores. Estes exigem a escolha correcta de contactos para substituir os desconectados, bem como estruturação de tabelas de encaminhamento relacionadas com os mesmos. Toda a gestão de informação e contactos tende a ser dispendiosa para o sistema, dependendo tal facto também da dimensão do mesmo.

Através da redundância nas DHT's, existe uma prevenção contra falhas de nós que disponibilizam certos conteúdos, pois existem sempre outros que partilham essa mesma informação, não sendo esses dados perdidos com a falha do nó em questão. Devido à sua estruturação, estes sistemas normalmente recuperam bem de situações de falhas de nós, reconstruindo as tabelas de contactos necessárias para manter a topologia. Estas falhas são normalmente detectadas por troca de mensagens *keep-alive* ou pelos processos em background já referidos. Estas notícias de falha de determinado nó são disseminadas no sistema para iniciar uma reestruturação do mesmo, dependendo os métodos utilizados da arquitectura adoptada.

2.3.3 Conclusões

Através da analise anterior das redes sobrepostas nas suas vertentes com estruturação ou sem, consegue-se deduzir algumas conclusões comparativas entre ambas, bem como do seu funcionamento geral. Algumas destas conclusões estão evidenciadas no estudo de [CCR05].

Quanto às redes não estruturadas, a sua simplicidade traz vantagens óbvias em termos de organização das mesmas, comprometendo o sistema em questões como as pesquisas, latência e escalabilidade. Sendo a questão das falhas, organização e junção no sistema facilmente tratada pela aleatoriedade e falta de critério inerente à solução, o mesmo já não se verifica quando se fala de escalabilidade e pesquisas. Comparativamente, as redes estruturadas, devido à sua estruturação, trazem vantagens inerentes a essa mesma organização, bem como uma escabilidade que até agora os sistemas de redes sobrepostas não tinham. Ao acoplar-se a uma topologia específica, a rede será sempre escalável enquanto a manutenção desta não se tornar muito dispendiosa. Contudo ambas as abordagens partilham do mesmo problema, ao ser uma rede virtual implementada numa física, a latência tende a depender do grafo formado e da sua distanciação para a rede física normal.

Em relação às pesquisas dentro da rede, as redes não estruturadas têm essencialmente três soluções possíveis. Uma inundação total da rede, muito dispendiosa mas que garante resultados, uma abordagem de passeios aleatórios, diminuindo significativamente as mensagens enviadas e baixando a taxa de sucesso, ou uma pesquisa informada que

necessita de outros custos de manutenção, garantindo bons resultados. A troca exagerada de mensagens em algumas destas abordagens leva ao aumento da latência do sistema, podendo comprometer a sua escalabilidade. Nas redes estruturadas, as pesquisas tendem a ser bastante eficientes e baratas em termos de troca de mensagens, não aumentando a latência. Existe também uma grande tolerância a falhas e mecanismos de recuperação, tanto de conteúdos como de topologia, que tornam o sistema muito mais eficaz do que nas redes não estruturadas. Porém todas estas vantagens baseiam-se na manutenção feita pelo algoritmo que pode trazer um grande *overhead* ao sistema. Este *overhead* e a informação que é necessária manter no sistema depende do tamanho do mesmo e da topologia adoptada.

Através desta análise percebe-se também o problema existente de escalabilidade nas redes não estruturadas inerente às muitas mensagens trocadas e latência do sistema. Assim as redes sobrepostas não estruturadas são uma boa alternativa a sistemas de pequena escala, mas menos eficientes globalmente. Pela estruturação, facilidade de ligação e recuperação de falhas o sistema escala bastante bem, falhando neste ponto exclusivamente pelas pesquisas dentro do mesmo. No caso das redes estruturadas tendem a não ter estes problemas anteriores referidos, à custa de possíveis *overheads*. Sendo uma boa alternativa às redes não estruturadas em termos de pesquisas e encaminhamento, existem variados tipos de redes estruturadas, cada uma com a sua abordagem em cada uma destas questões. De seguida serão analisadas algumas destas devido à sua importância dentro deste tema.

2.3.4 Primeira Geração

De seguida irão ser brevemente descritas algumas DHT's de primeira geração relevantes para este trabalho, que trazem conceitos importantes para o desenvolvimento do algoritmo *LiveFeeds*. A sua motivação é direccionada às pesquisas distribuídas e encaminhamento dentro de sistemas descentralizados *peer-to-peer*. De notar que será referenciado por n o número de elementos presentes no sistema.

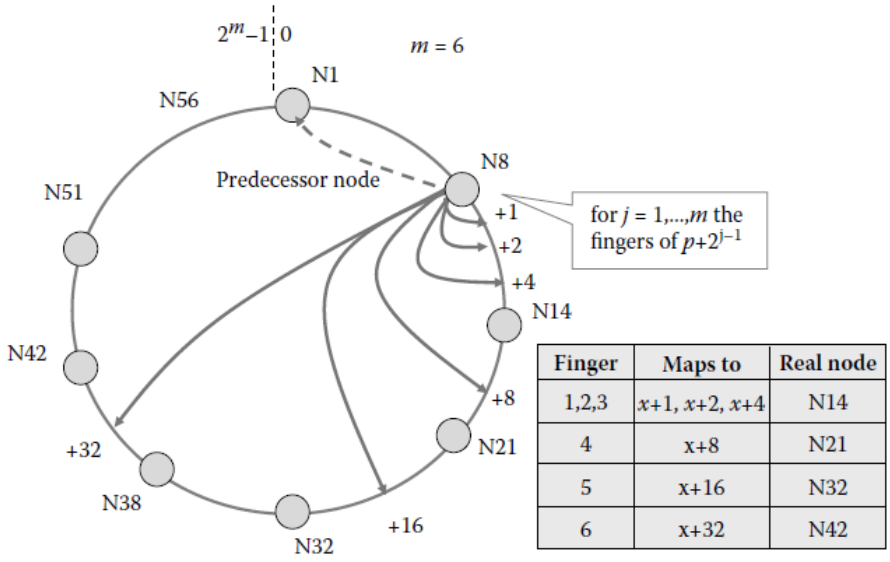
2.3.4.1 Chord

O Chord[SMLN+03] tem uma organização de nós em anel, sendo os dados espalhados por cada interveniente do sistema, ao qual é atribuído um identificador único neste. Esta divisão em identificadores do anel é feita através de um método de *hashing* consistente, sendo mapeadas chaves em nós. Devido à organização em anel cada nó tem um nó sucessor e um predecessor, de forma a estruturar o sistema, sendo o nó responsável pelas chaves entre o predecessor e o mesmo. É guardada informação de *logN* contactos em locais específicos do anel, de modo a encaminhar para identificadores mais longe em menor número de passos.

Para esta estrutura ser mantida, são necessárias algumas actualizações aquando de

entradas ou saídas de nós do sistema. Ao entrar no sistema, um nó procura o seu sucessor no anel, sendo que algumas das chaves pertencentes ao sucessor passam para o nó em questão, havendo reorganização de chaves e ligações de sucessor e predecessor, bem como dos contactos mais longínquos. Quando um nó sai do anel, informa tanto o predecessor como o sucessor desta saída e transfere as suas chaves para a responsabilidade do sucessor. Quanto ao encaminhamento, este é feito em $logN$ passos, sendo sempre verificado se a chave em questão é gerida pelo nó, se não o for a pesquisa é enviada para o contacto com identificador mais perto da chave desejada. De maneira a tolerar falhas são guardadas listas de predecessores e sucessores de modo a continuar uma execução normal perante erros inesperados.

As falhas são toleradas num ambiente dinâmico em que o sistema se adapta facilmente tanto à entrada, como à saída de nós. Tal não custa mais de $log_2 N$ de mensagens a atingir. As principais características que tornam o Chord interessante são a sua simplicidade, balanceamento de carga, correcção e *performance* que existem se os dados estiverem guardados segundo a sua estrutura definida. Desta maneira o sistema tem uma eficácia de quase 100 por cento aquando da falha colectiva de 45 por cento dos nós do sistema, mostrando uma grande tolerância a falhas. De seguida a imagem 2.9 mostra um anel Chord com um exemplo de uma tabela de encaminhamento, evidenciando as ligações entre os nós.



Finger	Maps to	Real node
1,2,3	$x+1, x+2, x+4$	N14
4	x+8	N21
5	x+16	N32
6	x+32	N42

Figura 2.9: Anel Chord[Tar10]

2.3.4.2 Pastry

A DHT Patry[RD01] é estruturada em forma de anel de identificadores, sendo referenciados a cada nó estampilhas aleatórias dentro do espaço de chaves do sistema. É utilizada uma função de *hash* consistente para este efeito, sendo importantes os prefixos dos identificadores, visto que a pesquisa ser feita através destes. Cada nó guarda informação sobre

outros nós em termos de prefixos próximos, nós com os quais tem uma baixa latência e os nós mais próximos numericamente.

Aquando dos eventos de entrada é espalhada uma mensagem especial de ajuntamento ao sistema que, chegada ao novo nó, transporta as tabelas de encaminhamento dos nós por onde passou de maneira ao novo popular a sua. Ao sair um nó do sistema, é escolhido o seu melhor substituto dentro de um grupo de chaves candidatas. Nesta estrutura o encaminhamento é feito na complexidade do logaritmo de n passos, estando gradualmente num prefixo mais perto do nó desejado a cada passo. É retornado o nó mais perto da chave procurada, sendo utilizados os contactos de prefixos mais perto, ou numericamente mais chegados de acordo com cada situação, tirando partido de métricas de baixa latência que tornam o Pastry uma DHT com esta característica.

Neste sistema as falhas podem criar partições auto-organizadas do anel do Pastry devido à constante actualização e reparação do sistema. Para ultrapassar esse problema são utilizadas técnicas de *multicast* limitado de modo a reintegrar possíveis partições no anel. Seguidamente aparecerá uma figura, 2.10, que esclarece a pesquisa por prefixos desenvolvida por esta DHT, de maneira a encontrar dados de acordo com algum tipo de prefixo relacionado com estes.

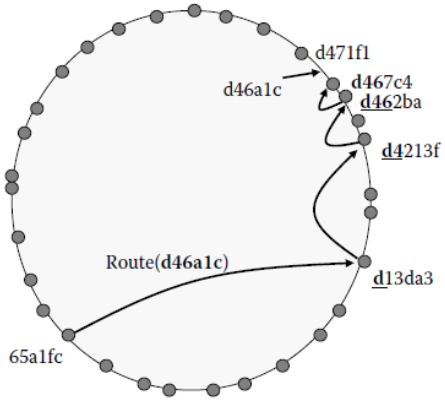


Figura 2.10: Anel Pastry com prefixos[Tar10]

2.3.5 Segunda Geração

Segue-se a descrição de algumas DHT's de segunda geração com importância relevante para a elaboração do algoritmo *LiveFeeds*. A motivação destas tem como base o encaminhamento e pesquisas com distintas características fulcrais em cada uma que serão abordadas e discutidas, mostrando o que de mais marcante acontece em cada caso. De notar que será referenciado por n o número de elementos presentes no sistema.

2.3.5.1 Koorde

O Koorde[KK03] é uma DHT que junta a teoria dos grafos de de Bruijn com as características mais marcantes do Chord que foram previamente referenciadas. A ideia é um

grafo de de Bruijn ser embebido no anel de identificadores, havendo um por cada número binário de b bits. Cada chave pertence ao nó sucessor da mesma, estando estas uniformemente espalhadas pelo sistema tal como os nós participantes. Esta é uma DHT que utiliza grau constante, ou seja, cada nó está ligado a um mesmo número de nós, dependendo a sua *performance* do grau escolhido para a implementar.

Utilizando grau 2 nesta DHT, cada nó tem dois contactos: ao retirar-se o bit mais alto do nó em questão, este aponta para os dois criados com a adição do bit mais baixo. Desta maneira o encaminhamento é feito através do descarte do bit mais significante fazendo um *shift* de um bit novo para o menor, correspondendo cada *hop* a um destes *shifts*. Como todos os nós não estão ligados em simultâneo, há nós de de Bruijn imaginários para simular o algoritmo sendo que no final o nó em questão que é obtido responde se a chave procurada existe ou não.

A tolerância a falhas é atingida pela utilização dos métodos do Chord, ou seja, a lista de sucessores e predecessores e toda a auto-reparação e organização inerente. Com grau 2 as pesquisas têm no máximo $logN$ passos, sendo que com grau log o encaminhamento pode ser feito em $\frac{logN}{loglogN}$ *hops*. Esta DHT tem todas as vantagens e simplicidade do Chord, aliando essas ao encaminhamento eficaz através dos grafos baseados em bits, atingindo um balanceamento entre o número de passos de uma pesquisa e o respectivo tamanho da tabela de encaminhamento.

2.3.5.2 Viceroy

A DHT Viceroy[MNR02] baseia-se numa técnica de *hashing* consistente mantendo o grau constante em cada nó e o diâmetro logarítmico do grafo da rede. A topologia baseia-se num anel de identificadores como os já estudados anteriormente com sucessor e predecessor, numa estrutura de $logN$ níveis e contactos de longa distância. Cada nó guarda 5 ligações para além do anel de identificadores, sendo uma para o nível acima e outra para o debaixo e 3 contactos de longa distância: dois direccionados para baixo no grafo e um para cima.

Aquando de uma junção no sistema é dado um identificador aleatório ao novo nó, sendo escolhido o seu nível e populados os seus contactos na rede. Quando um nó parte este avisa os nós que o conhecem, sendo a responsabilidade das suas chaves passadas ao sucessor. Esta organização proporciona um encaminhamento em $logN$ passos que é hierárquico sendo primeiro utilizados as ligações para os níveis debaixo e só no final o anel de sucessores. Através desta topologia consegue-se um encaminhamento e pesquisa eficazes mantendo o grau do grafo entre os nós constante. O Viceroy consegue o mesmo custo de encaminhamento como Chord ou outras DHT tendo cada nó simplesmente 7 contactos com uma latência média de comunicações entre nós.

2.3.5.3 One-Hop

Este tipo de encaminhamento é utilizado num sistema *peer-to-peer* em que cada membro tem conhecimento total dos demais nós presentes na aplicação, como descrito em [GLR03]. As tabelas de encaminhamento completas são guardadas e espalhadas para cada nó, para que o encaminhamento seja feito num só passo, mantendo os gastos de comunicação baixos. Esta abordagem só é possível se a informação dos membros ligados ao sistema seja correctamente e totalmente disseminada dentro deste, podendo também tolerar falhas de nós presentes no sistema.

Através deste sistema conseguem-se pesquisas num só *hop* com alta probabilidade de sucesso, sendo o custo para manutenção de toda a estrutura de encaminhamento baixo e viável para qualquer tipo de rede normal. Evitando a redundância e agregando eventos tornam o sistema mais eficiente com um *overhead* menor, sendo a tolerância a falhas atingida pela troca de mensagens *keep-alive*, substituição de nós preponderantes no sistema e repetição de pesquisas. Comparativamente com as DHT's anteriores, este sistema encaminha em muito menos passos, sendo discutido quando tal poderá ser feito no artigo [RB04].

2.3.5.4 Two-Hop

Tendo por base a ideia e discussão de um sistema que suporta procuras em um só passo, foi concebida uma alternativa que realiza essas procuras em dois *hops* de maneira a colmatar as limitações existentes do sistema supracitado[GLR04]. A ideia base da solução é a mesma, aparecendo esta para sistemas de grandes dimensões em que guardar informação de todos os membros na memória de cada nó deixa de ser viável e escalável.

Esta solução apresenta resultados igualmente bons, colmatando as falhas a qual foi direccionado. Assim é guardada informação total dentro de cada fatia do sistema, havendo contactos para as demais fatias escolhidos a partir de métricas servindo o primeiro *hop* para localizar a fatia correcta e o segundo para o contacto com o nó desejado. Todos os mecanismos de tolerância de falhas e encaminhamento têm por base a abordagem das pesquisas por um passo, sendo a diferença que neste tipo de rede são guardados menos contactos podendo usufruir-se de uma escalabilidade aliada a pesquisas rápidas.

2.3.5.5 Kelips

O Kelips[GBL+03] é um sistema *peer-to-peer* com o intuito de guardar ficheiros dispersos por todos os nós da aplicação. Contudo esta DHT tem duas características díspares das DHT's normais: no Kelips é guardada mais informação em cada nó e são usadas técnicas epidémicas para espalhar e replicar informação relevante. Através destas premissas o Kelips fornece uma procura constante em tempo e espaço, guardando relativamente pouca informação em cada nó e mantendo todos os dados do sistema coerentes e interligados.

O funcionamento deste sistema baseia-se na ideia de pesquisas e inserções de ficheiros constantes. Para tal, o sistema é dividido em grupos de afinidade, havendo conhecimento total dentro deste com contactos para os demais. Assim os grupos estão interligados à distância de um *hop*, custando raíz n em espaço de memória por cada nó. Para manter esta informação são utilizados algoritmos epidémicos de inserção e actualização dos dados. Estas tarefas correm em background inicialmente dentro de cada grupo, e espalhando-se para os demais tendo em conta *rtt's* entre nós. As entradas no sistema são mantidas desta maneira, como a actualização de dados sobre ficheiros e contactos utilizando vários protocolos de epidemia.

Esta DHT suporta procuras e inserções constantes através dos mecanismos referidos anteriormente. No entanto a memória utilizada em cada nó não é muito elevada ficando-se pelos poucos *MB's* para milhões de ficheiros, sendo este número irrisório comparando com as melhorias que produz. É tolerante a falhas, estabilizando o sistema rapidamente através da *stream* de epidemia, podendo as procuras ser feitas em vários *hops*, reduzindo a taxa de insucesso a nulo. Os dados são actualizados e inseridos à custa de um *overhead* constante no sistema que aumenta a latência, dependendo do número de nós e grupos. Este pode aumentar quando existem mais nós e ficheiros no sistema, podendo ser prejudicial para este. Em casos de falha os custos de execução aumentam, porém garantem sempre a resolução correcta da pesquisa.

3

Desenho da Solução

3.1 Introdução

Este capítulo descreve o trabalho efectuado, explicando as razões da evolução deste a partir do sistema já existente. Inicialmente haverá uma descrição detalhada do estado do sistema antes da implementação desta solução, servindo como base para o desenvolvimento deste trabalho.

De seguida serão explicados os pressupostos sobre os quais este trabalho se suporta, de maneira a evoluir e melhorar a versão do sistema já existente. Haverá uma reflexão sobre vantagens e desvantagens da utilização dos mesmos, explicando a motivação deste trabalho.

Depois de explicitada a motivação e vantagens inerentes à nova solução, será descrito o desenho do novo sistema. Este terá várias fases de desenvolvimento distintas, descritas cada uma delas na sua secção.

Estas fases estão divididas entre o algoritmo novo base e alguns melhoramentos a este, existindo também mecanismos que ajudam à evolução do sistema e mitigam possíveis limitações que poderiam aparecer da sua utilização.

3.2 LiveFeeds

O sistema base utilizado no desenho da nova arquitectura é o *LiveFeeds* e segue o funcionamento geral dos sistemas Editor/Assinante. O propósito deste é a disseminação filtrada de conteúdo, onde cada nó define um filtro sobre o conteúdo que deseja receber, sendo utilizada uma estruturação da rede segundo o paradigma *peer-to-peer*. O sistema

tira proveito desta estrutura para motivos de encaminhamento de conteúdo e distribuição de carga.

O *LiveFeeds* é composto por dois algoritmos, um de filiação e outro de disseminação de conteúdo. O primeiro é utilizado para controlar a junção e saída de nós do sistema, mantendo a estrutura desejada. O segundo pretende uma disseminação de conteúdo sem falsos negativos nem positivos, recebendo apenas a informação, os nós que declararam interesse na mesma. Este sistema foi totalmente descrito em 2.2.

3.3 Redundância de Filtros

Num sistema de grande escala com inúmeros nós, é de prever que exista também um grande montante de conteúdos dependente do número de participantes e do que estes partilham uns com os outros. Assim, devido ao elevado fluxo de dados no sistema, é importante que cada participante defina exactamente o que quer receber com o intuito de tornar a partilha de informação eficaz. Para que tal aconteça, cada nó escolhe as informações e notificações que deseja obter através de um filtro que define o conteúdo desejado.

Com o avolumar de conteúdos partilhados e havendo informações bastante populares, que toda a gente quer receber, é simples inferir que muitos participantes quererão receber os mesmos conteúdos. Desta forma, para os obter, irão definir filtros iguais entre si, criando uma gama de filtros repetidos na rede. Nestes sistemas de publicação e subscrição de informação, existem muitos filtros repetidos pelas razões explicadas anteriormente. Assim, existindo esta repetição de conteúdos definidos para cada nó diz-se que há uma redundância de filtros no sistema.

Os algoritmos anteriormente descritos têm um bom funcionamento e custo razoável para sistemas grandes e dinâmicos. Ao melhorar este sistema pretende-se um funcionamento mais eficaz, reduzindo os custos ao mitigar o *churn* de entrada, mantendo as garantias de não existência de falsos negativos e positivos na disseminação filtrada.

Neste trabalho pretende-se utilizar a referida redundância de filtros para melhorar o sistema e diminuir os custos dos algoritmos presentes neste. Desta forma pretende-se desenvolver uma arquitectura para o sistema baseada nessa repetição em que os participantes se ligam entre si, de acordo com o seu filtro. Esta abordagem prevê a aglomeração dos nós com o mesmo filtro no sistema, estando ligados a um responsável, denominado *super nó*.

Através da arquitectura que liga os participantes, esperam-se melhorias significativas, diminuindo a largura de banda utilizada pelo algoritmo ao reduzir o *churn* de entrada. Tal acontece pois apenas os *super nós* utilizam o algoritmo de filiação antigo, sendo estes uma parcela dos nós totais a entrar no sistema. Esta abordagem pode, também, trazer algumas desvantagens de funcionamento. Para garantir a estruturação atrás enunciada, serão necessários mecanismos de estruturação e promoção, em caso de falhas, mais complexos.

Tais assuntos referentes a vantagens e desvantagens, e como foram utilizadas ou mitigadas, serão discutidos posteriormente nas fases de desenho respectivas.

3.3.1 Ideia Base

O sistema *LiveFeeds* original apenas contemplava uma ideia de filtros muito abstracta, visto a sua definição concreta não ser necessária para o funcionamento do algoritmo. Por isso, os filtros funcionavam como uma espécie de "caixa preta", ou seja, apenas forneciam a informação se cada evento era aceite ou não, sem ter como base nenhuma definição. Com isto, funcionavam apenas como uma função que retorna *true* ou *false*, consoante o evento a filtrar.

Para o desenvolvimento desta nova arquitectura, esta solução era claramente insuficiente. Havia a necessidade de implementar uma noção específica de filtro, de maneira a que, durante a filiação, os nós se pudessem agregar de acordo com o seu filtro definido. Assim, tinha que ser desenvolvido um método que verificasse a igualdade dos filtros, de modo a ter uma base para estabelecer as relações entre participantes.

3.4 Fases da Solução

Este novo sistema, que tem por base os algoritmos anteriores, com uma arquitectura suportada na redundância de filtros existente, foi dividido em quatro fases de desenvolvimento, cada uma delas fornecendo características e funcionalidades únicas.

Inicialmente, ao nível da filiação, foi desenvolvida uma arquitectura do sistema com nós e *super nós*, representando as ligações entre filtros iguais. De seguida foi desenvolvido um mecanismo de tolerância a falhas, disponibilidade e replicação de responsabilidades. Posteriormente foi estudado e criado um mecanismo que gere o tamanho do sistema de acordo com a largura de banda utilizada. Para finalizar, em relação à disseminação filtrada, desenvolveu-se um mecanismo de recuperação de falhas na recepção de eventos.

De modo a facilitar a interpretação dos novos algoritmos foi incluído na explicação o pseudo código dos mesmos, sendo este referenciado aquando da descrição de cada algoritmo desenhado.

3.5 Arquitectura de *Super Nós*

3.5.1 Estruturação

O *LiveFeeds* tem como objectivo uma disseminação filtrada sem falsos positivos e negativos, sendo que para tal, todos os nós se conhecem entre si, bem como os seus filtros. Por conseguinte, baseando-se nesta arquitectura, se um nó não estiver presente na visibilidade completa e surgir um evento que lhe interessa, acontecerá um falso negativo. Para reduzir os custos deste sistema, é necessário desenvolver uma organização de rede que,

continue a garantir as premissas necessárias ao funcionamento do *LiveFeeds*, reduzindo a largura de banda utilizada na filiação.

3.5.1.1 Problema

A redundância de filtros existente dentro da rede junta uma nova característica ao sistema, que pode ser utilizada de maneira a este se montar e funcionar de maneira divergente da anterior. Como existem filtros repetidos, pode-se pensar na disseminação filtrada através de um método diferente, garantido à mesma a não existência de falsos negativos e positivos. Como cada participante define um filtro para o conteúdo que deseja obter, é lógico inferir que as notificações no sistema serão encaminhadas para os filtros que as aceitam. Assim a característica de visibilidade completa do sistema que evita os falsos positivos, reduzindo os falsos negativos, pode ser pensada em função de filtros e não de nós. Ou seja, conhecendo-se todos os filtros de um sistema, é garantido que existe uma entrega sem falsos negativos ou positivos, na ausência de falhas.

A partir desta nova noção de visibilidade completa, pode-se pensar numa arquitectura em que todos os nós apenas conhecem um nó de cada filtro distinto. Assim os demais nós com filtros repetidos estarão ligados ao representante do filtro, não entrando nos algoritmos de filiação e difusão. Esta abordagem manteria as características pretendidas para a difusão filtrada, devido à acção dos nós representantes de cada filtro. Estes, estando conectados a todos os participantes com filtros iguais ao seu, disseminam as notificações que recebem para todos os nós com o seu filtro, não sendo necessário que estejam todos presentes no sistema de filiação.

Utilizando um representante por filtro, o número de nós total a ser conhecido pelos representantes diminui. Com isto, a taxa de entrada de representantes também é reduzida, em relação a taxa de entrada de nós no sistema. Através desta redução do *churn* de filiação, o custos dos algoritmos decrescem, podendo-se suportar sistemas maiores e mais dinâmicos.

3.5.1.2 Solução

Nesta nova arquitectura os nós que representam o seu filtro dentro do sistema são denominados de *super nós*, enquanto que os participantes que se ligam ao *super nó* que possui um filtro idêntico ao seu são os *sub nós*. Os *sub nós* estão presentes no sistema, mas não participam no algoritmo de filiação visto não ser necessário que estes conheçam outros participantes. Esta diferenciação acontece no momento de entrada do nó na rede. O pseudo código do algoritmo de estruturação é apresentado na listagem 1.

Em (1), ao entrar no *LiveFeeds*, um participante verifica junto dos *super nós* do sistema se existe um representante para o seu filtro (3). Caso exista, estabelece uma ligação com ele (13-25). Se tal não acontecer este torna-se *super nó* e representante do seu filtro, entrando no sistema de visibilidade completa e passando todos os outros *super nós* a conhecê-lo (27). Esta ligação está exemplificada na figura 3.1. A entrada de um *super*

Listagem 1 Algoritmo de entrada no sistema

```
1   -join()
2       seed ← randomSuperNode()
3       send FilterCheck(self.filter) to seed
4           onFailure
5               repeat
6           onReply(FilterReply fr)
7               exists ← fr.superExists
8               if exists
9                   joinNode(fr.endpoint)
10              else
11                  joinSuperNodeNetwork()
12
13  -joinNode( Endpoint )
14      send JoinNode() to Endpoint
15          onFailure
16              joinSuperNodeNetwork()
17          onReply(SuperNodeAccept fr)
18              initSubnode()
19
20  -onReceive( JoinNode )
21      subnodes ← self.subnodes
22      if subnodes.size = 0
23          initPromotions()
24      subnodes.add(JoinNode.endpoint)
25      send SuperNodeAccept() to JoinNode.endpoint
26
27  -JoinSuperNodeNetwork()
28      Original System Join Method
29      ...
```

nó neste novo sistema utiliza o mesmo protocolo que qualquer participante na versão anterior.

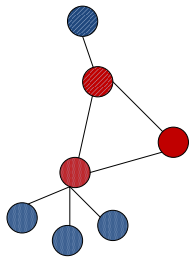


Figura 3.1: Estruturação - *Super nós* a vermelho e filhos a azul. Padrões representam filtros.

Deste modo consegue-se uma estrutura funcional em que todos os nós são tidos em conta, não necessitando todos de participar activamente nos algoritmos, conhecendo-se apenas os *super nós* entre si e aos seus filhos. Ao ligar os nós de um mesmo filtro a um *super nó*, não necessitando de passar pelo algoritmo de filiação, uma fracção das entradas do sistema é evitada. Assim se mitiga o problema do *churn* existente no sistema anterior. Como existem vários *super nós*, as entradas de *sub nós* são espalhadas pelos seus filtros, não sobrecarregando cada *super nó*. Tal acontece pois as entradas, sem ser de novos filtros, são tratadas localmente ao nível dos seus representantes.

De notar que, havendo dois tipos de nós diferenciados, cada um deles deverá possuir bases de dados distintas de acordo com o seu papel no sistema. Estas deverão guardar toda a informação relevante para o funcionamento do sistema, adicionando-lhe dados sobre irmãos, nós filhos e nó pai, e outro tipo de informação que facilite e seja necessária para a execução do novo algoritmo.

3.5.2 Concorrência

Contudo, como em todos os sistemas distribuídas e redes *peer-to-peer*, existem eventos concorrentes na rede que podem gerar situações de excepção. Ao avaliar que situações concorrentes podem afectar o sistema, surge uma bastante pertinente. Num sistema com uma elevada taxa de entrada, podem entrar dois nós com o mesmo filtro simultaneamente, gerando uma entrada concorrente em relação a um filtro específico. Ao tentarem entrar concorrentemente e não havendo um *super nó* para esse filtro, o sistema processa-os como duas entradas, ficando ambos como *super nó* do seu filtro.

Este caso da concorrência pode ser um problema, visto que não garante que exista apenas um *super nó* por filtro. A solução poderia passar por um protocolo mais complicado, que procurasse *super nós* com filtros repetidos, eliminando-os. Porém, em redes

peer-to-peer, os protocolos mais complexos normalmente geram mais complicações e maiores custos, podendo esta questão da concorrência ser benéfica para o sistema. Tal acontecerá por vários motivos. Esta repetição de *super nós* é esperada mínima pois refere-se ao primeiro momento de entrada de nós de um filtro, gerando poucos *super nós* desse mesmo filtro e afectando um número baixo de filtros. Sendo poucos, os custos do algoritmo não aumentarão significativamente, podendo estes nós serem utilizados como mecanismos de tolerância a falhas e melhor disponibilidade ao seu filtro. Assim, o que poderia ser um potencial problema, transforma-se num ponto forte desta nova arquitectura. O impacto deste problema foi avaliado no capítulo 5 de resultados mais à frente.

3.5.3 Promoções

Até agora, com esta nova arquitectura para o sistema *LiveFeeds*, temos cada um dos filtros representados por um *super nó*, ao qual estão conectados os demais nós com esse filtro. De notar que, devido ao algoritmo adoptado, o *super nó* responsável por um filtro é sempre o primeiro nó a entrar no sistema com essa especificação. Num sistema deste tipo, seria melhor para o seu funcionamento que o *super nó* se portasse como um servidor, nunca saindo do sistema. Porém, nestes sistemas *peer-to-peer* é muito comum estadias curtas na rede, nunca se mantendo infinitamente neste.

3.5.3.1 Problema

Tomando como argumento que os tempos de estadia são limitados e impossíveis de prever, observam-se duas características: os *super nós* acabam por sair do sistema e que, sendo impossível prever o tempo de participação, é impossível escolher um representante que optimize tal substituição. Analisando a primeira destas conclusões, percebe-se que o algoritmo até agora desenvolvido tem o problema de quando o *super nó* sai do sistema. Quando tal acontece, deixa de haver representante desse filtro, perdendo os *sub nós* ligados a ele a ligação ao pai e, consequentemente, ao sistema. Este problema traria grandes debilidades ao sistema, perdendo a ligação aos nós anteriormente conectados ao pai e consequente indisponibilidade desse filtro.

De maneira a manter o sistema funcional, enquanto existirem nós com um determinado filtro na rede, é obrigatório que esteja presente neste um *super nó* para essa especificação de conteúdo. Desta forma, é necessário um algoritmo que consiga substituir o representante de um filtro, aquando da sua saída do sistema.

Assim, cada vez que um *super nó* de determinado filtro sair do sistema poderão acontecer dois casos. Caso o *super nó* que sai do sistema não tenha qualquer *sub nó* ligado a este, significa que não deixará nenhum participante do sistema sem ligação à aplicação, não sendo necessário desenvolver nenhuma acção. Assim, o próximo nó com esse filtro que entrar no sistema será *super nó*. Deste modo, percebe-se que quando o *super nó* cessante não tem filhos, nenhuma acção de estruturação é necessária.

Contudo, quando o *super nó* que sai do sistema tem filhos, é necessário um tratamento

especial para que estes não percam a ligação ao sistema, de modo a garantir que para cada filtro existe um *super nó*. Pretende-se que a estrutura se mantenha a mesma, ou seja, um *super nó* com os restantes participantes ligados a este. De notar que os outros *super nós* recebem este evento de saída, mas como não conhecem os filhos de cada *super nó*, nada podem reparar. Deste modo entende-se que a solução terá que partir dos próprios nós filhos.

Assim, a ideia para resolver este problema seria promover um dos *sub nós* filhos, ligando os restantes filhos do *super nó* cessante ao recém promovido *super nó*. Para que esta abordagem seja bem executada é necessário que algumas pré condições sejam satisfeitas. A primeira cinge-se com o facto de que, para que um filho seja promovido e tenha informação dos outros participantes ligados ao seu pai, este tem que ter conhecimento dos seus irmãos dentro do sistema. Outra relaciona-se com o método de detecção da falha ou saída do *super nó* pai.

3.5.3.2 Solução

Em relação ao conhecimento dos irmãos entre si, este é necessário para não se perder o serviço a nenhum nó, ou para não se promoverem todos simultaneamente. Existem várias maneiras de endereçar esta gestão com avisos de chegada por parte do *super nó* aos filhos, ou agregação de chegadas. De modo a diminuir a largura de banda utilizada inserimos este mecanismo juntamente com o algoritmo de promoção, não havendo trocas adicionais de mensagens.

Para se detectar que um nó saiu do sistema ou falhou, é necessário que a ligação a este por parte de outro falhe. Sendo desejado que um dos filhos seja promovido, estes têm que detectar, de alguma maneira, a falha ou saída do *super nó* pai. Para esta situação existem, por exemplo, dois métodos distintos para a detecção da falha. Um deles tem como base a ideia de os filhos contactarem o pai, verificando se este está operacional. O outro mecanismo é basicamente o contrário, ou seja, o *super nó* vai alertando os filhos de que está *online* e, caso não exista um aviso atempado, os filhos sabem que o pai saiu do sistema.

A detecção é feita como descrito no segundo mecanismo falado, pois a verificação por parte dos filhos exigiria uma recepção enorme de mensagens por parte do pai. Na figura 3.2 demonstram-se, de maneira simples, os dois métodos distintos de detecção de falhas.

Estando definido o modo de detecção da falha, é necessário desenvolver um algoritmo que o utilize de forma a substituir o *super nó* cessante. Na listagem 2 será apresentado o algoritmo simplificado do mecanismo das promoções, que acompanhará a explicação do desenho do mesmo.

Assim, o pai iniciará periodicamente a disseminação de uma mensagem, mostrando que está *online* (30-45). Esta mensagem será entregue a um filho e encaminhada, um a um, para os restantes, de maneira a obterem a informação de que o pai continua no sistema. Ao detectar a falha de recepção da mensagem por parte do pai dentro do intervalo de

Listagem 2 Algoritmo de promoções simplificado

```
30 -initPromotions()
31     promotion ← self.subnodes
32     childPromotion( promotion )
33     sleep( PROMOTION − INTERVAL )
34     initPromotions()
35
36 -childPromotion( List<Endpoints>)
37     if Endpoints.size > 0
38         endpoint ← Endpoints.first
39         send Promotion( Endpoints ) to endpoint
40             onFailure
41                 self.subnodes.remove(endpoint)
42                 Endpoints.remove(endpoint)
43                 repeat
44     else
45         stopPromotions()
46
47 -subnodePromotion()
48     sleepSubnode ( PROMOTION − INTERVAL )
49     promote()
50     send Promoted() to self.brothers
51
52 onReceive( Promotion<Endpoints> )
53     self.brothers.add(Endpoints)
54     sleepSubnode ( PROMOTION − INTERVAL + eta )
55     if self ≠ Endpoints.last
56         send Promotion( Endpoints ) to endpoint
57             onFailure
58                 self.brothers.remove(endpoint)
59                 Endpoints.remove(endpoint)
60                 repeat
61     else
62         parent ← self.parent
63         send LiveNodes( Endpoints ) to parent.endpoint
64
65 onReceive( LiveNodes<Endpoints> )
66     self.subnodes.set(Endpoints)
67     if Endpoints.size = 0
68         stopPromotions()
```

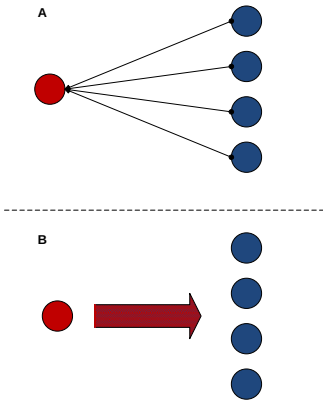


Figura 3.2: A - Filhos verificam se o pai está vivo. B - Pai avisa os filhos que está *online*.

tempo estipulado, um nó aperceber-se-à que o seu *super nó* saiu da aplicação (47).

Em (50), quando um filho detecta a saída do *super nó* pai, avisa os seus irmãos conhecidos que se vai promover. De seguida entra no sistema como um *super nó* normal, tendo já como nós ligados a ele os seus antigos irmãos (49). Desta forma o sistema mantém a arquitectura, existindo um *super nó* para o filtro. Para que não se promovam vários filhos ao mesmo tempo, existe uma hierarquia na mensagem disseminada, referida em (52-60), significando a permanência do pai no sistema. Esta hierarquia previne a promoção ao mesmo tempo de vários filhos, o que poderia gerar grande número de *super nós* com o mesmo filtro, tornando esta arquitectura ineficaz. Este algoritmo está exemplificado na figura 3.3.

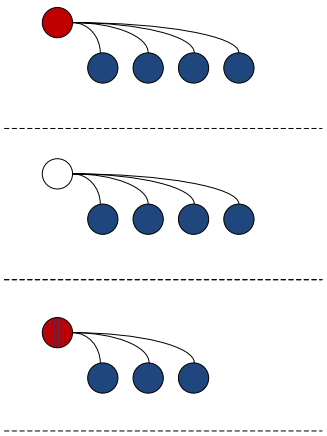


Figura 3.3: Saída de um *super nó* e sua substituição

Desta maneira, a saída de nós do sistema é tratada pelo mecanismo de promoção, mantendo a nova arquitectura. As saídas de participantes que apenas estão ligados a *super nós* são também tratadas por este algoritmo, removendo estes nós das listas de irmãos ou filhos. Tal é feito aquando da disseminação da mensagem por parte do pai. Com estes mecanismos, o sistema mantém a nova arquitectura ao longo do tempo, podendo ser

desenvolvidas novas funcionalidades e uma disseminação filtrada menos dispendiosa.

3.5.3.3 Conhecimento dos irmãos e filhos - Hipóteses

Anteriormente foi referido que era necessário os *sub nós* conhecerem todos os outros que estão ligados ao mesmo *super nó* de maneira a que, aquando da promoção de um deles, nenhum fique sem pai. Existem variadas maneiras de fomentar este conhecimento entre os nós do sistema. Uma delas seria o *super nó* alertar os seus filhos sempre que um nó entrasse, não devendo aglomerar entradas para não deixar nós sem serviço. Outra ideia seria o pai entregar a cada filtro uma percentagem dos contactos dos seus filhos, contactando-se eles posteriormente de maneira a completar a base de dados de nós. Estas comunicações poderiam ser epidémicas.

Inicialmente foi desenvolvido um sistema em que o *super nó* fornecia uma percentagem reduzida de contactos ao novo nó. Este contactava um deles, informando-o da sua junção e pedindo o contacto dos irmãos. O nó contactado, informava os restantes da junção de um novo participante e facultava ao novo nó a lista de irmãos. O novo nó recebia essa lista e contactava os restantes nós enviados pelo pai, de forma a verificar se algum contacto lhe escapava. Desta maneira era garantido que todos os filhos se conheciam entre si a uma taxa de 100 %, não deixando nenhum irmão sem ligação ao pai caso algum deles se promovesse.

Contudo, esta abordagem foi abandonada pois gastaria largura de banda desnecessariamente. Isto deve-se ao facto de, ao se juntar o conhecimento de irmãos com o esquema de promoção, a solução torna-se mais barata, em termos de largura de banda gasta. Assim, tornou-se a promoção um bocado mais dispendiosa em termos do tamanho das mensagens, beneficiando com a não utilização de dois mecanismos separados.

3.5.3.4 Conhecimento dos irmãos e filhos - Solução

Com o esquema de promoções, o conhecimento dos filhos é garantido pela permanência dos nós já visitados na mensagem de promoção passada desde o *super nó* até os *sub nós*. Este conhecimento pode ter nós que já não estão no sistema devido à mensagem inicial enviar o conhecimento anterior do pai, podendo o estado dos seus filhos já estar alterado. Porém,este caso não é um problema para o sistema. Sempre que se tenta contactar um nó e essa comunicação falha, é sempre feita outra tentativa com outro nó, até não haverem filhos.

Este algoritmo também trata um terceiro problema. A questão do conhecimento do pai sobre os filhos que estão *online*, e os que já saíram, é resolvida com a mensagem final enviada para o *super nó*. Apesar de não ser preocupante o pai ter informação de filhos que já não estão no sistema, é mais eficiente se a sua lista estiver actualizada, caso que este método engloba e resolve apenas com o envio de uma mensagem por aviso do *super nó* que está operacional.

Com o algoritmo anterior, este problema era tratado através de um aviso, tanto aos

pais, como aos irmãos da detecção de um nó que já havia saído do sistema. Este método obtinha uma grande eficiência na actualidade da informação do pai, a custo de alguma largura de banda. Na nova abordagem , este custo é menor, justificando-se a alteração do mecanismo antigo implementado, que foi explicado.

De notar que, quando entra um novo nó no sistema e se liga a um representante, este adianta o seu aviso de *keep-alive* de modo a que o novo nó passe a ser conhecido. Tal acontece pois, se o pai saísse antes de fazer esse alerta, esse *sub nó* não seria conhecido pelos irmãos e acabaria por se promover sozinho.

3.6 Tolerância a Falhas ao nível da Filiação

Numa rede que funciona segundo o paradigma *peer-to-peer*, a disponibilidade e tolerância a falhas do sistema são características importantes para que este funcione da maneira desejada, satisfazendo os participantes. Assim, pretende-se garantir estas características para o sistema até agora descrito, colmatando as suas possíveis limitações em relação a estes aspectos.

Tolerância a falhas neste sistema significa a capacidade deste que, perante uma falha de um ou mais participantes, se regenere e consiga que a sua execução permaneça normal sem perda de funcionalidades. A disponibilidade é demonstrada a partir da total manutenção da interacção entre participantes e o sistema. Com estas duas características bem desenvolvidas, um sistema torna-se robusto contra falhas e problemas, melhorando significativamente o seu desempenho.

3.6.1 Problema

Até agora foi descrito um novo sistema de filiação que implementa uma arquitectura de *super nós* com visibilidade completa dos filtros, com promoção dos filhos aquando da falha dos pais. Segundo esta arquitectura, cada filtro estará sempre acessível através do seu respectivo *super nó*, quer para entradas de novos participantes, quer para a disseminação filtrada de conteúdo. Contudo, se analisarmos ao detalhe o funcionamento das promoções, existe um período de tempo de indisponibilidade.

Esta indisponibilidade advém da falha de um nó quando tem filhos, ficando o filtro sem *super nó* até tal ser colmatado por uma promoção. Assim, o problema principal é a geração de um período momentâneo sem *super nó* antes da promoção de um novo participante. O período de tempo em causa é entre a detecção da falha e o *download* de filtros e *endpoints* para a entrada do *super nó* substituto. Para mitigar este problema é necessário um acréscimo de disponibilidade e tolerância a falhas, de modo a melhorar o sistema.

Durante esta janela temporal, os filhos do *super nó* cessante ficam momentaneamente sem receber conteúdos. Outro caso, vindo da indisponibilidade, seria a entrada de um

participante aquando da falha, ainda não tratada, de um *super nó*. Caso exista um representante concorrente não é gerado qualquer problema por esta situação. Se esse *super nó* concorrente não existir, não conseguindo o novo participante aceder ao representante do seu filtro, podendo ser-lhe informado pelos restantes que este já saiu, este novo participante poderá entrar como *super nó*, trazendo uma anomalia ao sistema.

Este não será um problema de maior para a filiação como já foi analisado quanto às entradas concorrentes. De qualquer modo, será sempre benéfico eliminar esta situação, juntando alguma replicação de *super nós*, de maneira a tolerar completamente todo o tipo de falhas do sistema, aumentando o mais possível a disponibilidade a determinado filtro.

3.6.2 Solução

Ao analisarmos esta nova arquitectura baseada em *super nós*, percebe-se que, teoricamente, o custo da filiação diminuirá com a redução da taxa de entrada de nós para o sistema de visibilidade completa. Caso se aumente o número de *super nós* de maneira controlada, mantendo o *churn* destes abaixo da taxa de entrada de nós total no sistema, haverá sempre uma redução de custo face à solução base.

Tomando como ponto de partida as conclusões anteriores, a funcionalidade seguinte foi desenvolvida para garantir uma melhor tolerância a falhas e disponibilidade. O mecanismo consiste em aumentar o número de *super nós* por filtro presentes no sistema, evitando os problemas supracitados de tolerância a falhas e disponibilidade, à custa de uma maior largura de banda utilizada. O algoritmo de funcionamento deste mecanismo surge da alteração de algumas funcionalidades de 1. Essas alterações na listagem 3, de modo a facilitar o entendimento do algoritmo descrito seguidamente.

Com este mecanismo tendem a existir, pelo menos, K *super nós* desse filtro presentes no sistema, como se percebe pela análise do código em (69 - 94). Tal é feito pela avaliação, aquando da entrada de novos nós, de quantos representantes do filtro são conhecidos (84). Partindo deste novo pressuposto, quando um participante entra, tem sempre variados *super nós* que pode contactar, mitigando o problema da entrada aquando do tempo de detecção da falha e promoção. Porém, estes K *super nós* não terão todos as mesmas funções dentro da rede. Isto acontece pois, resolvendo o problema da entrada de participantes, cria algum particionamento entre os filhos com esse filtro. Essa divisão entre os *super nós* significaria a continuação do problema da disponibilidade aos filhos aquando da falha.

Para se ultrapassar esta situação, o mecanismo contempla a ideia de que existem *super nós* no sistema, filhos de um *super nó* representante. Ou seja, estes entram na filiação do sistema de visibilidade completa, mas não podem ter filhos nem responsabilidades, transferindo-as para o seu pai. Assim, aquando de uma falha, mal esta seja detectada, um desses filhos substitui o pai, não perdendo o tempo de entrar no sistema, visto já ser *super nó*. Desta forma a disponibilidade é imediata após a detecção da falha, sendo o problema referido mitigado.

Listagem 3 Alterações ao algoritmo de entrada no sistema para mecanismo de K nós

```
69  -joinNode( Endpoint )
70      send JoinNode() to Endpoint
71          onFailure
72              joinSuperNodeNetwork()
73          onReply(SuperNodeAccept fr)
74              initSubnode()
75          onReply(KNodeAccept fr)
76              initSubnode()
77              joinSuperNodeNetwork()
78          onReply(ParentNode pn)
79              joiNode(pn.endpoint)
80
81  -onReceive( JoinNode )
82      if self.mainSuperNode
83          subnodes ← self.subnodes
84          n ← sameFilterNodes(JoinNode.filter)
85          if subnodes.size = 0
86              initPromotions()
87          subnodes.add(JoinNode.endpoint)
88          if K ≥ n
89              send SuperNodeAccept() to JoinNode.endpoint
90          else
91              send KNodeAccept() to JoinNode.endpoint
92      else
93          parent ← self.parent
94          send ParentNode(parent.endpoint) to JoinNode.endpoint
```

Estes *super nós* filhos em número igual ou superior a K, contando este K com alguma entrada concorrente que poderá ter acontecido, garantem uma disponibilidade muito maior ao algoritmo, sendo um forte mecanismo de tolerância a falhas. Este método está exemplificado na figura 3.4.

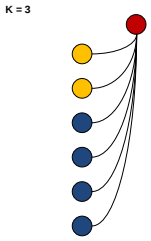


Figura 3.4: Estruturação com K = 3 *super nós*. *super nós* filhos a amarelo.

De notar que o funcionamento destes novos nós K, que são *super nós* e filhos simultaneamente, fazem aparecer um terceiro estado no funcionamento do sistema. Assim, para além de existirem os estados básicos de *super nós* e *sub nós*, passa a existir este terceiro, que junta um pouco dos dois. Este estado terá comportamentos distintos dos anteriores, como já foi descrito, introduzindo maior complexidade no sistema e na sua gestão.

Esta funcionalidade deverá ser utilizada em redes que necessitem uma disponibilidade muito elevada, com grande tolerância a falhas, aumentando os custos de largura de banda utilizada.

3.7 Dimensionador do Sistema

Com as funcionalidades descritas até este momento, temos uma nova arquitectura para o sistema baseada em *super nós*, com promoções de forma a substituí-los em caso de saída do sistema. Existe também um mecanismo que fornece uma grande tolerância a falhas, garantido uma maior disponibilidade aos serviços do sistema.

3.7.1 Ideia Base - Dimensionador

Todos estes mecanismos têm por base a redundância de filtros esperada num sistema deste tipo. Utilizar sempre este parâmetro como base pode ser restritivo para um sistema, apesar de a redundância de filtros esperada ser alta. Com esta limitação, o tamanho do sistema fica limitado ao número de filtros existentes neste e às taxas de concorrência e tolerância a falhas. De modo a mitigar esta limitação, pretende-se o desenvolvimento de uma funcionalidade que relaxe o tamanho do sistema, tendo em conta as possibilidades e limitações deste.

Ao dimensionar o sistema através de outra metodologia, não tendo por base os filtros, a rede pode ser estruturada de acordo com as suas necessidades ou possibilidades de largura de banda. Tal é positivo para o sistema pois adapta-o ao que realmente é

necessário, não restringindo em demasiado a sua estrutura, sobrecarregando menos os *super nós* presentes na aplicação. Este mecanismo terá como base explorar um eventual excedente da capacidade de *upload* definida, crescendo o sistema se esta quantidade de largura de banda não for utilizada na sua totalidade. Uma vantagem deste mecanismo, é que este dimensionador poderá funcionar com qualquer estrutura de *super nós*, podendo as ligações entre nós serem feitas de acordo com outro critério que não seja a igualdade de filtros.

Assim, a ideia por detrás do funcionamento deste mecanismo é bastante simples: se o sistema tiver capacidade de largura de banda excedentária para o *upload* estabelecido, sem limitar a execução da aplicação na rede, ele evolui para uma rede maior de *super nós*. Este mecanismo é apenas utilizado para a filiação, visto o tamanho do sistema estar dependente das entradas e saídas do mesmo.

3.7.2 Fases da Solução do Dimensionador

3.7.2.1 Avaliação do Sistema e Cálculo de Evolução

Os participantes da rede são constantemente avaliados em relação à largura de banda gasta, sendo inferido, a partir desses dados e o excedente definido, como deverá evoluir o sistema para que o funcionamento seja óptimo. Esta análise é feita periodicamente, conseguindo inferir a evolução do sistema e de que modo poderá este dimensionar-se futuramente.

De maneira a analisar a rede, este mecanismo analisa, não só a largura de banda gasta, mas também o número de entradas no sistema. Este número de entradas refere-se às entradas de *super nós*. Comparando a largura de banda gasta, com a entrada de *super nós*, consegue-se estimar o comportamento do sistema, ao nível da filiação, e dimensioná-lo de forma a torná-la mais eficaz.

Os participantes que efectuam esta avaliação da rede e cálculo da evolução são os sequenciadores. Tal deve-se ao facto de serem estes a iniciarem a disseminação de informação de filiação, tendo uma melhor perspectiva do funcionamento geral do sistema e partindo destes uma posterior disseminação de *slots*. Assim, são calculados, nestes, todas as estatísticas necessárias para inferir a melhor evolução da rede. Tendo por base o limite de *upload* e o restante que se poderá verificar numa análise, podem-se estimar o número de promoções que fariam o sistema evoluir naturalmente, para cada *slice* do sistema.

3.7.2.2 Obtenção e Geração de *Slots*

Estando já o sistema analisado, falta definir como são obtidos os *slots* de promoção por parte do sequenciador, e como é que estes serão distribuídos pelo sistema, de maneira a promover *sub nós*. Começando a disseminação de entradas pelo sequenciador, faz sentido que seja este também a coleccionar os *slots*, para posterior disseminação dos mesmos, simultaneamente à das entradas. Assim, a obtenção de *slots* é desfasada temporalmente das análises periódicas feitas no sequenciador.

Existe um desfasamento temporal entre a recolha dos *slots* e a sua análise visto serem acções separadas no algoritmo e com periodicidades não compatíveis. Por causa destas, é importante delinear um método em que os *slots* obtidos sejam só os computados, independentemente das vezes que um sequenciador tenta obter *slots* entre análises. Para evitar que sejam obtidos várias vezes os mesmos *slots*, desenvolveu-se um mecanismo em que estes são gerados de acordo com uma taxa de geração. Desta maneira, quando um sequenciador tenta coleccionar os *slots* produzidos, só irá obter os que já foram criados desde a última análise.

Juntamente com a idealização deste método de criação de *slots*, foram desenvolvidas algumas medidas para que a evolução do sistema seja feita dentro dos limites estabelecidos, respeitando as análises feitas. Através destas estima-se a taxa a que o sistema deverá gerar *slots* para se observar o tamanho indicado. Porém, esta evolução poderá ser muito abrupta, se o sistema estiver a gastar pouco, criando muitas promoções. De modo a que a rede cresça racionalmente sem entradas em massa, que podem provocar sobrecarga nas comunicações no momento do consumo dos *slots*, desenvolveu-se um método em que a taxa de geração é uma média calculada entre a taxa desse momento e a anterior. Tal fará com que o sistema evolua, passo a passo, até ao tamanho calculado, não entrando os nós abruptamente, sendo o crescimento da rede evolutivo.

Outra medida utilizada é a inutilização dos *slots* não consumidos aquando da geração de uma nova taxa, pois esta indicará, provavelmente, outro tipo de evolução. Assim, o sistema terá de se comportar de maneira diferente, pois a evolução segundo as promoções anteriores iria degradar a nova estimativa, sendo estes *slots* desnecessários para o crescimento pretendido à nova taxa. Existe, também, a impossibilidade de um gerador criar mais *slots* do que os que foram estimados inicialmente, de maneira a não aumentar o sistema para tamanhos inadequados.

Estas medidas que são tomadas são cautelosas, nivelando por baixo o número de *slots* gerados tendo em conta o excedente de *upload*. Assim entende-se que esta abordagem é conservadora pois pretende, em caso algum, ultrapassar os limites estabelecidos. Como são descartados *slots* e produzidas médias evolutivas, deve-se observar a não utilização total do excedente, sendo a preocupação principal nunca o ultrapassar.

3.7.2.3 Disseminação de *Slots*

Definido o esquema de geração e obtenção de *slots* para o sequenciador, resta explicar como é que, a partir destes, certo número de nós se irá promover. Na actual situação, o sequenciador vai buscar o número de promoções criadas ao seu gerador, aquando do envio das novas chegadas agregadas. Estes *slots* de promoção têm que ser disseminados para o sistema equilibradamente, de maneira a dar a hipótese de promoção a nós de vários filtros, mantendo a estrutura do sistema e fazendo com que o número de *super nós* de cada filtro não diferisse exageradamente.

De notar que a disseminação da informação de filiação é suposta chegar a todos os

nós, podendo ser utilizada para distribuir os *slots* pelos *super nós* contactados. Assim, quando houver a possibilidade de promover nós, esta informação será adicionada aos dados partilhados na filiação, com o intuito de espalhar os *slots* de promoção pelos *super nós* na rede.

Ao receber estes dados com o número de promoções disponível, um *super nó* irá utilizar uma para si e iniciar o algoritmo de divisão de *slots* pelas restantes mensagens de filiação que vai enviar. Caso não existam mais promoções disponíveis, o algoritmo de disseminação será utilizado sem alterações daí em diante. De seguida existem dois casos distintos, o nó pode enviar informações para participantes folha na árvore de disseminação, ou que não o sejam. Para cada um dos casos existe uma definição diferente da promoção dos nós.

Caso este não envie para nós folha, e tenha *slots* disponíveis, irá , primeiramente, verificar se tem maior ou igual número destes comparado com o *fanout* calculado. O *fanout* é o número de nós a quem este participante irá enviar mensagens de filiação. Na situação de os *slots* serem em número maior ou igual, é enviada uma mensagem com promoções para todos os nós. Caso contrário, é apenas enviado para os primeiros nós quem comunica, equivalentes ao número de promoções disponíveis. Se existirem ainda mais *slots* que o número de nós com quem se irá comunicar, esses serão contabilizados como nós extra, e distribuídos pelo número de mensagens a enviar.

Se, por outro lado, o nó que recebe estes dados for enviá-los para participantes folha na árvore, o método de promoção será distinto do referido anteriormente. Neste caso, enquanto houver promoções disponíveis, estas são incorporadas no envio da informação de filiação. Assim, ao enviar os dados dos novos nós que entraram, é trocada também a informação que existe um *slot* disponível para esse *super nó*. Ao receber essa informação, o nó é alertado que pode promover um dos seus filhos.

De notar que nesta fase final do algoritmo são ignoradas as promoções em excesso. Tal deve-se ao facto de, chegando *slots* às folhas, é uma indicação de que todos os *super nós* vão utilizar uma promoção, provocando o aumento para o dobro do tamanho. De forma a limitar este aumento, todos os restantes *slots* são ignorados, de maneira a manter a estabilidade e não sobrecarregar o sistema. Esta abordagem vai de encontro ao que foi referido anteriormente, designando este mecanismo de conservador de modo a nunca exceder o limite imposto.

3.7.2.4 Promoção de um Filho

Ao receber a informação que pode promover um dos seus filhos, o *super nó* em causa inicia o processo de alerta de um dos seus *sub nós* para a sua promoção. Desta maneira se conclui o funcionamento do algoritmo, sendo promovidos os *sub nós* relativos ao número de *slots* coleccionadas pelo sequenciador.

3.8 Disseminação Filtrada

Estando completo o novo algoritmo de filiação com base em *super nós*, é agora necessário adaptar a disseminação filtrada a esta arquitectura. De modo a obter uma disseminação sem falsos positivos ou negativos, o algoritmo que distribui os conteúdos na rede terá que ser modificado, visto este só trabalhar dentro da visibilidade completa definida na rede. Assim, a entrega aos *super nós* continua garantida, falhando, porém, a disseminação de eventos para os seus filhos, que não participam na filiação da rede.

3.8.1 Entrega com *Super Nós*

3.8.1.1 Problema

Como já foi referido, o algoritmo de disseminação de conteúdos anterior, funcionava apenas para os nós dentro da visibilidade completa do sistema. Porém, com esta nova arquitectura de *super nós*, todos os participantes que apenas ficam ligados a um *super nó* representante não fazem parte desta visibilidade completa da rede. Assim, os filhos dos *super nós*, não receberiam qualquer evento, tendo em conta o algoritmo anterior. Por esta razão, este teve que ser desenvolvido, de forma a garantir as propriedades necessárias neste sistema: ausência de falsos positivos e, especialmente, negativos.

Esta arquitectura de *super nós*, sendo cada um representante de um filtro, foi adoptada para melhorar a filiação, mitigando o problema do *churn* e diminuindo os custos, mas também por facilitar a disseminação filtrada. Esta é facilitada pois, ao invés de percorrer um sistema linearmente com todos os nós, são analisados apenas os *super nós*, estando os restantes participantes com determinado filtro a cargo desses mesmos representantes. Assim, os intervalos para análise são mais curtos, tendo todos os nós que irão receber o conteúdo aglomerados, e não espalhados pelo intervalo de identificadores do sistema.

Tomando como princípio que os *sub nós* com o mesmo filtro estão aglomerados, ligando-se ao *super nó* em questão, a não chegada dos eventos aos filhos pode-se resolver de maneira simples. Partindo deste pressuposto pode-se estender o algoritmo anterior, de maneira a que cada representante informe os seus filhos do conteúdo por ele recebido, para além de efectuar o seu trabalho normal na disseminação filtrada.

3.8.1.2 Solução

Como, com a nova arquitectura, uma significativa parte dos nós não está presente na visibilidade completa, ficando dependente de um representante, é necessário introduzir métodos de disseminação de conteúdo entre *super nós* e filhos. Assim, para garantir a inexistência tanto de falsos positivos como negativos, é necessário desenvolver um método de entrega de eventos aos nós filhos, que garanta que os eventos sejam obtidos por todos os nós que os desejam receber.

Assim, o *super nó* representante que recebe um evento, tem como função distribuí-lo pelos participantes dependentes dele. Existem vários métodos de envio possíveis para o

Listagem 4 Entrega de eventos a *sub nós*

```
94  onReceive( TurmoilPayload<...>)
95      ...
96      sendToSubnodes ( TurmoilPayload )
97
98  -sendToSubnodes( TurmoilPayload<...>)
99      endpoint ← self.subnodes.removeRandom()
100     Endpoints ← self.subnodes
101     send TurmoilSubnode<TurmoilPayload<...>,Endpoints> to endpoint
102         onFailure
103             repeat
104
105 onReceive( TurmoilSubnode<TurmoilPayload<...>,Endpoints>)
106     acceptEvent(TurmoilPayload<...>)
107     if Endpoints.size > 0
108         endpoint ← Endpoints.removeRandom()
109         send TurmoilSubnode<TurmoilPayload<...>,Endpoints> to endpoint
110             onFailure
111                 repeat
```

pai notificar os filhos do evento recebido. Como estes têm o mesmo filtro, não é necessário verificar os nós que aceitam o conteúdo pois todos têm as mesmas definições de aceitação que o pai.

Os métodos que poderiam ser utilizados são variados. O pai poderia enviar os dados a cada filho, contudo esta abordagem gastaria muita largura de banda e aumentaria a carga de trabalho do *super nó*. Outra abordagem seria os filhos pedirem, de tempo a tempo, os eventos que chegaram ao pai. Dependesse do número da filhos a contactá-lo, este método poderia gastar largura de banda em excesso, tendo também o problema de não receber os eventos na altura da sua publicação. De seguida é referido o algoritmo desta entrega na listagem 4, de maneira a facilitar o entendimento do desenho descrito.

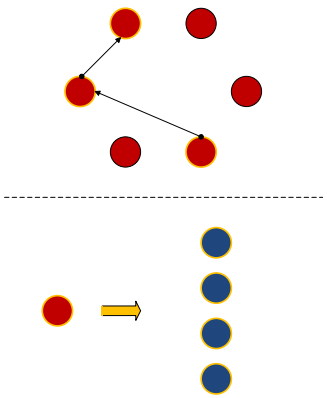


Figura 3.5: Disseminação entre *super nós* e posterior envio aos filhos

Assim, de maneira a não sobrecarregar o *super nó*, este enviará o evento unicamente

para um dos seus filhos, encarregando-o de continuar a disseminação do conteúdo para os restantes (94-103). Aquando da recepção de um evento, este *sub nó* também tem os métodos anteriormente descritos como possibilidades para a disseminação do conteúdo. Com o intuito de distribuir ao máximo a carga entre nós, optámos pela mesma política de disseminação de conteúdo, ou seja, cada *sub nó* entrega a outro. Tal acontece sucessivamente, até já não haverem mais *sub nós* desse representante.

Porém, existe um detalhe por explicar na entrega entre os *super nós*. Se não for referido a quem entregar ou a quem já foi entregue, estes estariam constantemente a trocar eventos às cegas, gerando falhas, uma baixa eficácia e gastos de largura de banda desnecessários. Assim, haveria duas metodologias distintas a adoptar: o *super nó* enviar na mensagem os nós a contactar, ou adicionando os nós já contactados à mensagem. O segundo método é possível de utilizar pois existe um conhecimento total entre os filhos do mesmo *super nó*.

Após analisar os dois casos, chega-se à conclusão que a largura de banda utilizada será a mesma visto, num método a lista vai diminuindo, enquanto que no outro crescendo. Basicamente um é o contrário de outro. Assim, o segundo parece mais viável por não expor tanto o pai a gastar *upload*. Porém, existe uma situação que contraria esta decisão: os nós só têm conhecimento de todos os outros filhos quando o *super nó* avisa que está vivo. Ou seja, caso um participante se junte ao *super nó* depois deste comunicar que está *online* e exista uma disseminação de conteúdo antes de outro aviso, os *sub nós* desconheceram este participante.

Por causa deste pormenor, e de maneira a garantir que todos recebem o conteúdo que definiram aceitar, escolhe-se a primeira abordagem, criando o *super nó* uma lista de entregas, de onde retira o nó a quem enviou o evento (101).

Desta forma, ao aceitar um conteúdo, um *super nó* irá informar um dos seus filhos da chegada desse mesmo evento. A entrega da notificação aos restantes filhos é delegada para o *sub nó* que recebeu o evento. Este fará passar o conteúdo a outro *sub nó*, delegando-lhe o intervalo restante de irmãos. O método será recursivo e em cadeia, de modo a todos os filhos receberem o evento em causa (105-111). A metodologia adoptada permite, também, uma distribuição da carga deste mecanismo por todos os seus intervenientes. De notar que os nós não receberão eventos duplicados devido à ordem do intervalo de *sub nós* imposta. A figura 3.5 demonstra o método de disseminação de conteúdo desde a recepção por parte do pai, até à delegação do evento a um filho.

3.8.2 Reparação de Falhas

3.8.2.1 Problema

Tanto a nova funcionalidade anterior, ao nível da disseminação filtrada, como o sistema antigo são *best-effort*, podendo existir falhas na entrega do conteúdo. Tal acontece se o algoritmo experienciar a falha de um dos nós intervenientes no mecanismo, quer na disseminação entre os *super nós*, quer no envio aos filhos dos eventos. Em ambos os casos

essa falha seria prejudicial para o sistema, induzindo falsos negativos indesejáveis para o sistema.

Especificamente, podem acontecer dois casos em que as falhas trazem problemas à aplicação. O primeiro, refere-se à falha de um *super nó* que recebeu um evento, e está encarregado de avaliar certo intervalo de identificadores de *super nós* do sistema. Com a saída deste representante, esse intervalo deixa de ser analisado e os possíveis nós presentes neste que receberiam o evento, não o recebem. Assim podem ser gerados falsos negativos, bastantes prejudiciais para os intuitos desta aplicação.

O segundo ponto crítico assinalável gera um problema idêntico ao anterior. Este acontece aquando da disseminação de um evento para e entre os filhos de um *super nó*. Caso o nó que está a efectuar a distribuição do conteúdo, dentro do seu filtro, saia ou experiencie alguma falha, observa-se a falta de recepção de conteúdo de um ou mais participantes desse filtro. Com isto, são gerados falsos negativos no grupo de nós do mesmo filtro, ou seja, os filhos do *super nó* representante.

Gerando estas duas situações falsos negativos, torna-se imperativo montar um mecanismo que tolere estas falhas, de modo a evitar tais falhanços na recepção de conteúdo. É relevante perceber que o primeiro caso é mais grave pois, não recebendo um *super nó* determinado evento, os seus filhos também não o receberão havendo uma taxa muito maior de falsos negativos.

3.8.2.2 Solução

Para resolver este primeiro, e mais grave, problema na disseminação filtrada, idealizou-se um mecanismo que repare as falhas. Uma falha, na disseminação filtrada, é a ausência de recepção de determinado conteúdo que o filtro do nó aceitava, ou seja, uma falha é um falso negativo. De notar que, como a saída de nós ou *super nós* é pouco provável, estas falhas também serão pouco significativas, como será avaliado posteriormente. Atendendo a este facto, adopta-se um modelo de reparação menos exaustivo, ou seja, um modelo epidémico. Tal deverá ser suficiente pois, através da troca de eventos com alguns nós já conhecidos, os eventos são restaurados.

Assim, cada nó irá efectuar, de tempo a tempo, uma troca de informação ponto a ponto dos conteúdos recebidos. Para desenvolver este mecanismo de reparação é necessário que cada nó do sistema guarde um histórico dos eventos recebidos, de maneira a compará-lo e inferir os conteúdos que perdeu. Este histórico guarda os identificadores dos eventos recebidos, de maneira a facilitar trocas de históricos. O algoritmo de reparação está simplificado na listagem 5, de modo a acompanhar a seguinte descrição detalhada.

Cada *super nó* irá contactar outro periodicamente, de modo a efectuar esta reparação (112-115). Contudo, se esta for feita para nós aleatórios no sistema, as recuperações de eventos poderão demorar a aparecer, visto que grande parte dos nós poderá ter filtros completamente contrários. Para minimizar os contactos em que nada é recuperado,

Listagem 5 Reparação de eventos nos *super nós*

```
112 repairTurmoil()
113     if DB.loadedEndpoints
114         endpoint ← self.neighbours.random()
115         send RepairTurmoil<self.history > to endpoint
116             onFailure
117                 repeat
118             onReply( TurmoilRepairReply<Differences,Events> )
119                 for ( e : Events)
120                     self.history.add(e)
121                     sendToSubnodes( TurmoilPayload< e > )
122                 diffs ← calculateDifferences( Differences )
123                 if diffs.size > 0
124                     events ← getEvents( diffs )
125                     reply TurmoilRepairReply< null, events >
126                 repairSleep( REPAIR − INTERVAL )
127
128 onReceive( RepairTurmoil< History >)
129     differences ← getDifferences( History )
130     if differences.size > 0
131         events ← getEvents( differences.this )
132         reply TurmoilRepairReply< differences, events > to endpoint
133             onReply( TurmoilRepairReply<Differences,Events> )
134                 for ( e : Events)
135                     self.history.add(e)
136                     sendToSubnodes( TurmoilPayload< e >)
137                 repairSleep( REPAIR − INTERVAL )
```

instaurou-se um método em que cada *super nó* apenas contacta aqueles com quem já comunicou anteriormente, na disseminação filtrada de eventos. Adicionalmente, são também introduzidos todos os nós conhecidos pelo *super nó* em questão, que tenham um filtro igual ao previamente guardado. Assim se constrói uma base de dados de nós com que cada *super nó* tem mais afinidade, de maneira a aumentar a probabilidade de recuperação de eventos, no contacto restrito com estes nós. Ao comunicar com os nós desta base de dados, a probabilidade de terem eventos que faltam uns aos outros é significativamente maior, visto já comunicarem entre si e terem filtros que se abrangem.

Ao receber uma comunicação de reparação com um histórico, um *super nó* avalia as diferenças entre o seu histórico e o recebido (129). Com isto, este infere quais eventos lhe faltam que o outro nó possui, e vice-versa. Isto é feito em duas fases: inicialmente o nó calcula que identificadores de eventos faltam a ambos os intervenientes na reparação. Seguidamente, o nó receptor do contacto, analisa a lista de identificadores de eventos que faltam ao emissor, verificando quais é que o filtro deste aceita. O receptor pode fazer esta análise pois: tem todos os eventos da lista que faltam ao emissor e conhece o filtro deste, avaliando quais dos eventos diferentes entre os dois é que ele aceita. Com isto, o nó receptor fica com vários dados interessantes: a lista de eventos que tem de enviar ao emissor e a lista de identificadores que diferencia a sua história da do outro nó.

Na posse destes dados, o receptor do primeiro contacto de reparação, envia-os numa comunicação ao outro nó. A mensagem trocada só é enviada se houver alguma diferença entre os históricos, caso contrário não é necessário mais trocas pois ambos têm os mesmos eventos (130-132).

O nó que iniciou a reparação, ao receber estes novos dados, consome os eventos que lhe faltam, juntando-os ao seu histórico e enviando-os para os seus filhos (118-121). Seguidamente, se a lista de identificadores em falta por parte do outro nó for maior que zero, este *super nó* verifica quais destes eventos é que são aceites pelo outro, de maneira idêntica à referida anteriormente (122). Caso falte ao outro nó algum evento, é-lhe enviada a informação de resposta idêntica à anterior, mas sem uma lista de identificadores e diferenças (123-125). Ao receber esta resposta, o outro *super nó* consome os eventos da mesma maneira já indicada (133-136).

Através deste mecanismo os eventos perdidos pelos *super nós* são recuperados e posteriormente disseminados para os seus filhos, resolvendo a questão mais grave. Como as trocas são epidémicas, a recuperação de eventos perdidos deverá tender para 100%, através de um número diminuto de partilha ponto a ponto. Endereçado este problema, surge o seguinte: a falha na entrega entre os filhos.

A solução para este problema é em tudo semelhante à anterior. Aliás, o método é igual, trocas epidémicas entre os participantes com os históricos dos eventos recebidos. Porém, estas comunicações serão só feitas ao nível dos filhos, e quanto muito do pai, de forma a receber imediatamente as informações necessárias. Neste método, todos os nós com quem se comunica deverão ter o mesmo histórico, e portanto será mais rápida a obtenção de eventos perdidos e mais fácil encontrá-los.

Assim, a recuperação de eventos periódica de um *sub nó* inicia-se com a verificação se este continua *sub nó*. Tal é necessário pois, devido a promoções, poderiam ser gerados casos de trocas epidémicas entre *super nós* e *sub nós*, que se revelariam desnecessárias. Seguidamente, é escolhido ao acaso um *sub nó* irmão e inicia-se o contacto enviando o histórico, como referido anteriormente.

Ao receber esta informação, um *sub nó* verifica as diferenças entre os históricos de identificadores. Assim, este computa os dados de que eventos que um tem, faltam ao outro. Caso falte algum evento a algum dos intervenientes, é trocada a informação que possui os identificadores que faltam ao receptor da primeira mensagem, e os eventos que faltam ao seu emissor.

Quando é alertado deste caso, o *sub nó* em causa consome os eventos e verifica se o outro necessita de algum evento que ele possui. Se tal acontecer, é desenvolvido um novo contacto, apenas com os eventos pretendidos.

Para que estas reparações, tanto ao nível de *super nó* como de *sub nó*, sejam correctamente desenvolvidas, necessitam de duas características adicionais. A primeira é o tempo definido entre o início de tarefas de reparação. Num sistema real, faria sentido uma adaptação dinâmica deste tempo tendo em conta os vizinhos, número de *sub nós* irmãos, número de eventos no sistema, número total de *super nós* e eventos consumidos em reparações recentes. Todos estes dados deveriam influenciar uma mais rápida ou lenta recuperação de eventos. Outra solução seria definir um espaço temporal único configurável, obrigando a reparações temporais fixas. Este foi o método adoptado pela sua simplicidade.

Outro detalhe é o tamanho do histórico enviado. Ao longo do tempo, este há-de crescer indefinidamente, tomando tamanhos incomportáveis para enviar em mensagens. Assim, foi implementado um mecanismo de truncagem, em que só é trocado o histórico mais recente. Este, está dependente da taxa de entrada de eventos no sistema. Para sistemas com muitos eventos a chegar ao mesmo tempo, deverá existir a troca de um histórico maior, de maneira a efectuar as reparações necessárias. Porém, este mecanismo terá um custo cada vez maior, com o aumento do histórico trocado. Esta relação seria interessante de estudar e avaliar.

De notar que, através destes mecanismos de trocas epidémicas, ganha-se outra funcionalidade extra. Esta é, nós que ainda não estavam no sistema e são contactados por outros já presentes, podem receber eventos passados detectados como falhas. Assim podem-se obter conteúdos de eventos antigos, trazendo este facto uma mais valia importante ao sistema de disseminação filtrada. A figura 3.6 demonstra uma reparação.

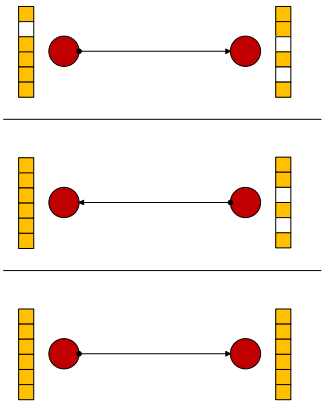


Figura 3.6: Comunicação ponto a ponto de modo a reparar o histórico

4

Implementação

No capítulo anterior foi detalhado o desenho da nova arquitectura de *super nós* para o *LiveFeeds*, sendo explicado e fundamentado o funcionamento geral das suas novas funcionalidades. Com o intuito de demonstrar como as opções de desenho foram postas em prática, são explicitados os detalhes de implementação da nova arquitectura. Assim, explica-se com que métodos é que as características pretendidas no desenho da aplicação foram cumpridas.

De modo a descrever a implementação com uma metodologia evolutiva e sistemática, as funcionalidades desenvolvidas foram separadas de acordo com a ordem de desenvolvimento. Desta maneira, conseguir-se-á entender a evolução dos mecanismos. As funcionalidades estão divididas em quatro fases gerais, como já aconteceu no capítulo anterior, cada uma com as suas características próprias.

Toda a implementação do sistema foi desenvolvida num simulador de eventos discretos. Este simula com realismo o comportamento de redes de computadores através de tarefas, que correspondem a fios de execução concorrentes. Estas permitem simular a existência de vários nós, troca de mensagens entre estes e saída e entrada do sistema dos mesmos. Este simulador permite o agendamento de tipos de mensagens diferentes, trocadas entre participantes. O algoritmo *LiveFeeds* anterior também estava implementado neste simulador.

De notar que o simuladores permite criar variados tipos de mensagens que interagem no sistema. Num sistema real, estas mensagens seriam acrescidas de um custo de tipificação, de modo a se distinguirem aquando da recepção. Na descrição da implementação são referidos os tamanhos das mensagens implementados no simulador, devendo ser acrescidos ao tamanho da tipificação da mensagem.

4.1 Fases de Implementação

Como na explicação relativa ao desenho desta nova solução, existem quatro fases de implementação. Cada uma delas pode ter mais do que uma funcionalidade, dependendo do que foi implementado no âmbito desse mecanismo. As fases são as seguintes: Arquitectura de *Super Nós*, Tolerância a Falhas na Filiação, Dimensionador do Sistema e Disseminação Filtrada.

4.2 Arquitectura de *Super Nós*

Neste tópico iremos referir os principais detalhes de implementação desenvolvidos de modo a obter a nova arquitectura de *super nós* funcional. Assim, será explicado como é simulada a redundância de filtros no sistema, a ligação dos *super nós* entre e si e, finalmente, a metodologia de implementação do mecanismo de promoções.

Os detalhes de implementação começam a ser explicados pelo início, ou seja, como se simula a redundância de filtros. Progressivamente se descreverá os demais mecanismos, tendo uns como base a implementação dos anteriores.

4.2.1 Filtros

A definição de filtro presente em cada utilizador do sistema foi implementada segundo um *long*, em que cada valor diferente representará, necessariamente, um filtro diferente. Com esta definição, um nó ao entrar no sistema define o seu filtro segundo um número. Como o algoritmo funciona num simulador, esse filtro é lhe correspondido na criação do nó.

Para simular esta atribuição de filtros aos novos participantes, de acordo com uma redundância de filtros, pensou-se que cada um gera um número aleatório, aquando da sua entrada, sendo essa a definição do seu filtro. Esta definição, contudo, tem que ser contida dentro de um espectro limitado de filtros, como já foi explicado anteriormente. Este intervalo de filtros foi desenvolvido segundo três constantes do sistema: tempo médio de estadia, taxa de redundância de filtros e taxa de entrada de nós no sistema.

Através do tempo médio de estadia e taxa de entrada obtém-se uma estimativa da dimensão do sistema que, multiplicada por 1 menos taxa de redundância de filtros, resulta no número de *super nós* estimado que o sistema deverá ter. Assim, limita-se o número de filtros disponíveis ao número de *super nós* estimado, sendo este valor guardado e utilizado para a geração de um novo número (filtro) entre 0 e o valor calculado. A atribuição do filtro é feita ao se inicializar um novo nó do sistema. A informação de que nós existem por filtro é guardada para motivos de funcionamento do mesmo.

4.2.2 Redundância de Filtros

A limitação da implementação do sistema desenvolvido até então, proveniente da falta de definição de filtros, era a não utilização de uma taxa de redundância para os mesmos. Esta não podia ser definida pois não existia qualquer método de igualar filtros, segundo uma taxa esperada para o sistema. Com a definição de filtro, teve que se montar um mecanismo que, segundo uma taxa de redundância de filtros, espelha a mesma no sistema, em termos dos filtros que cada nó possui.

4.2.2.1 Hipóteses

Inicialmente, para desenhar a redundância de filtros, foi desenvolvido um método estatístico bastante simples. Dada uma taxa de redundância de filtros definida para o sistema, quando um novo nó se junta é sorteado um número aleatório de 0 a 1. Variando a taxa de redundância do sistema de 0 a 100 %, ou seja, de 0,0 a 1, pode-se fazer corresponder o número aleatório gerado com a taxa. Ao se corresponder o número aleatório à taxa, insere-se uma condição de que, se o número for menor ou igual à taxa de redundância, esse nó tem um filtro novo gerado. Caso contrário, o nó teria um filtro já existente no sistema, sendo-lhe atribuído um aleatoriamente.

Através deste método, os filtros eram gerados segundo a taxa de redundância, correspondendo a totalidade dos filtros à taxa de redundância configurada. Apesar dos bons resultados a gerar correctamente os filtros segundo uma taxa, este método trouxe um problema em relação à sua adaptação a um sistema real. Esta simulação obtinha bons resultados na geração de filtros, mas falhava ao simular a entrada de nós do sistema correctamente. Detalhando, num sistema deste tipo são esperadas entradas concorrentes de nós do mesmo filtro, coisa que, com este algoritmo, era evitado devido à atribuição de um filtro já existente ao nó suposto concorrente.

Assim, este mecanismo teve que ser alterado, de forma a simular melhor a realidade, tendo em conta o caso das entradas concorrentes esperadas numa rede *peer-to-peer*. De notar, que não se pretende com este sistema definir uma taxa de concorrência à partida, quer-se medir esta taxa após o funcionamento do algoritmo. Desta maneira, pensou-se num método em que os nós entrassem no sistema já com determinado filtro, respeitando a taxa de redundância.

4.2.2.2 Solução

Desta forma, de maneira a simular com maior coerência a entrada de nós no sistema, foi desenvolvido um método distinto para a simulação da redundância de filtros. A ideia deste novo método é simples, é gerado um número limitado de filtros distintos em forma de um intervalo, só podendo existir, nó máximo, esse número de filtros.

Assim, para corresponder correctamente aos valores esperados de *super nós* e *sub nós* tendo em conta a redundância de filtros, é calculado o número de *super nós* estimado para a redundância pretendida e o intervalo é gerado de acordo com esse valor. Quando

um novo nó entra na aplicação, este define o filtro aleatoriamente dentro do intervalo calculado, simulando assim as entradas concorrentes como seria pretendido.

O correcto funcionamento deste mecanismo é exemplificado no capítulo 5, de resultados.

4.2.3 Estabelecimento da Arquitectura de *Super Nós*

Para montar a nova arquitectura, o algoritmo de filiação foi incrementado, em algumas comunicações extra, mantendo o método de disseminação de entradas e saídas. Como já foi referido, só entram para a visibilidade completa do algoritmo de filiação os *super nós*, ficando dependentes de um representante. Assim, existem dois tipos de nós, e portanto, dois tipos de bases dados diferentes que necessitam de ser implementados.

Os dois tipos de bases de dados necessários são implementados em classes diferentes, hierarquicamente, de modo a que uma base de dados *sub nó* possa ser também uma de *super nó* (caso dos nós K). Assim permite, também, uma evolução simples de uma para a outra, aquando de uma promoção. As informações adicionais necessárias como listas de filhos e outros dados, são guardadas em listas dentro das bases de dados respectivas.

Tomando como ponto de partida as novas bases de dados e os dois tipos de nós distintos, foi desenvolvido um novo protocolo de entrada na aplicação. Assim, ao querer entrar para a rede, o novo nó envia uma mensagem a um *super nó* aleatório do sistema. Essa mensagem contém, apenas, o seu filtro, sendo o espaço gasto por este (506 bytes), o tamanho dessa mensagem. Caso a entrega desta seja falhada, o novo nó tenta contactar outro *super nó*. Ao receber uma mensagem deste tipo, um *super nó* avalia a sua visibilidade completa do sistema e verifica que representantes conhece com essa definição de filtro. Se não encontrar nenhum, envia uma mensagem vazia do tipo *super nó*. Caso tenha na sua base de dados um, ou mais, nós com esse filtro, este *super nó* envia ao participante que quer entrar uma nova e diferente mensagem.

Essa mensagem deverá conter o *endpoint* de um dos *super nós*, com aquele filtro, presentes no sistema. O tamanho do *endpoint* foi estipulado em 6 bytes, 4 para o *IP* e 2 para o porto. Esse é o tamanho da mensagem enviada. O novo participante, ao receber esta nova mensagem, inicia um contacto com o *super nó* que lhe foi referenciado. Este contacto é desenvolvido através do envio de uma mensagem vazia do tipo de junção ao pai. Caso o contacto com o *super nó* falhe, o novo participante volta ao início, contactando outro *super nó* aleatório. Por outro lado, se o contacto for bem sucedido, o novo nó espera uma resposta à sua mensagem.

O *super nó*, ao receber a mensagem anterior, adiciona o novo participante como seu filho, respondendo-lhe com uma mensagem vazia de aceitação. Ao receber essa resposta, o novo participante do sistema define aquele *super nó* como pai.

Através deste protocolo de trocas de mensagens, entre nós que se querem juntar ao sistema e *super nós* que já pertencem a este, consegue-se estabelecer uma arquitectura segundo o que foi descrito na fase de desenho.

4.2.4 Promoções

Através da implementação anterior monta-se a nova arquitectura do sistema. Porém, para manter a organização de nós funcional, é imperativo implementar o mecanismo de promoções descrito na fase de desenho, devido às saídas e entradas de novos participantes. Como já foi referido, foi escolhido um método de verificação da disponibilidade do pai, em que o próprio representante se anuncia aos seus filhos. De notar que este mecanismo serve, também, para informar cada filho de quais os outros que partilham essa condição.

Começando pela promoção de nós, esta é baseada em duas tarefas periódicas, uma executada no pai e outra em cada filho. Estas tarefas são iniciadas aquando da existência de um ou mais filhos de *super nó* e quando os nós se juntam a um representante, respectivamente. Visam desenvolver o método de detecção da falha, ou saída, de um *super nó*, iniciando imediatamente a promoção de um dos seus filhos.

4.2.4.1 Detecção da saída

Tendo em conta o método adoptado para a detecção de falhas, é necessário que o *super nó* avise os filhos que está *online*. A tarefa periódica do *super nó* consiste precisamente nisso, periodicamente avisa os nós dependentes deste que continua activo no sistema. Porém, o aviso a todos os seus filhos pode-se tornar muito dispendioso para um *super nó*, dependendo do número destes e da redundância de filtros existente. Assim, a implementação desta tarefa inclui uma hierarquia de envio entre os filhos, de modo a que o *super nó* apenas envie uma mensagem, reportando que está *online*.

Detalhando, periodicamente, o *super nó* envia a um filho uma mensagem contendo os *endpoints* de todos os irmãos. Essa mensagem é passada entre os irmãos que enviam a mensagem para o nó seguinte da lista, até serem todos contactados. Ao contactar os irmãos, se algum tiver falhado, este é removido da lista e a mensagem é enviada a outro participante, caso exista. Quando um nó se apercebe que é o último da lista, reenvia ao pai uma lista de todos os nós contactados, ou seja, todos os que estão *online* no sistema.

Ao receber a lista de nós final, o pai tem a informação de quais dos seus nós estão vivos, actualizando a sua base de dados. Os irmãos também vão recebendo esta informação, parcialmente tratada pelos *sub nós* já contactados. Esta hierarquia foi criada com o intuito de poupar o pai a grandes gastos em termos de largura de banda utilizada em comunicações. O tamanho das mensagens são variáveis, contendo sempre listas com *endpoints*. Assim, ocupando cada um 6 bytes, a mensagem custa 6 bytes vezes o número de filhos presentes na lista, em determinada altura.

A escolha, por parte do *super nó*, a que nó deve enviar a mensagem para iniciar a sua disseminação é feita com base em uma de duas políticas. Não se podendo prever quanto tempo os nós ficarão no sistema, é impossível escolher qual será o melhor nó a promover para diminuir o número de promoções, aumentando a disponibilidade. Das políticas implementadas uma delas é FIFO, sendo enviada uma lista de acordo com a ordem de

junção dos nós. A outra é LIFO em que a lista é composta pela ordem inversa. Não foi desenvolvida uma política aleatória pois o algoritmo pode tornar-se inconsistente com tal caso. Isto acontece quando a ordem da lista é aleatória de cada vez, podendo um nó que recebeu a mensagem em primeiro na comunicação passada, recebê-la agora em último acabando a sua tarefa de verificação antes. Tal geraria entradas de filhos, mesmo quando o pai estaria *online*. Uma possibilidade de tornar esta política viável seria relaxar bastante os tempos de comunicações e verificações, causando, porém, maior indisponibilidade ao filtro em causa.

4.2.4.2 Promoção de *sub nó*

Quanto aos *sub nós*, estes implementam uma tarefa periódica, durante a qual esperam ser contactados de forma a inferir que o *super nó* continua em funções. Ou seja, se a mensagem descrita anteriormente chegar antes da tarefa acabar, esta é marcada para a próxima janela temporal de acordo com o período. Se tal não acontecer, esse *sub nó* terá a informação que o pai falhou e irá promover-se. Os *sub nós* têm as suas tarefas separadas com um curto intervalo de tempo relativo ao *download* da mensagem, de maneira a impedir que a tarefa de verificação acabe em dois nós e se promovam ambos, causando inconsistências no sistema. De notar que quando um nó se junta a um representante, esta tarefa será atrasada um período de modo a que, juntando-se logo após da comunicação, o novo nó não entre imediatamente.

Quando um *sub nó* detecta que o pai falhou, alerta os seus irmãos de que se vai promover. Para isto envia uma mensagem de promoção, vazia, a todos os nós que estavam dependentes do *super nó* cessante. Ao receberem esta mensagem, esses nós alteram a informação de quem é o seu representante e reprogramam a sua tarefa de promoção. Após enviar estas mensagens, o *sub nó* promove-se, alterando o seu estatuto no sistema, preenchendo a sua base de dados de *super nó*, cancelando a tarefa de promoção e entrando directamente no sistema como *super nó*, após fazer o *download* de filtros e *endpoints*. A partir deste momento a estrutura está novamente reposta, funcionando o sistema sempre dentro destes padrões descritos.

4.3 Tolerância a Falhas e Disponibilidade

De modo a implementar a solução desenhada para este mecanismo, descrita detalhadamente na fase de desenho, foi necessário desenvolver e tornar mais complexo o método inicial de junção de nós ao sistema e estabelecimento da arquitectura. Assim, iremos descrever as alterações necessárias à implementação da estruturação da rede, de modo a desenvolver esta funcionalidade, tendo em conta o desenho já referido.

O primeiro contacto do novo participante com um *super nó* aleatório, mantém-se inalterado, sendo-lhe enviada a informação de um *super nó* com o seu filtro, ou juntando-se o novo participante como *super nó*. Ao contactar o *super nó* do seu filtro, caso exista, este

irá avaliar o número de *super nós* (contando com o próprio) que tem na base dados com o seu filtro. Estes *super nós* poderão ser do tipo *super nó* filho, ou *super nó* concorrente. Se no total tiver menos *super nós* na visibilidade completa que um K definido, o comportamento do sistema será alterado, caso contrário o funcionamento é igual ao mecanismo descrito na secção 3.5.1.

A alteração realizada no caso de ter menos de K *super nós* é simples, em vez de o *super nó* enviar uma mensagem de aceitação ao filho, partilha uma mensagem de aceitação e junção ao à rede de *super nós*. Com isto o *super nó* torna esse novo participante seu filho, entrando na tarefa das promoções e dependendo do pai para obter o serviço dado pela aplicação. Por seu lado, o filho ao receber esta mensagem torna-se *sub nó* desse representante, implementando uma base de dados referente a tal. Ao mesmo tempo este entra no sistema de *super nós*, de visibilidade completa, fazendo o *download* de *endpoints* e filtros, sendo conhecido pelos outros *super nós* e implementando, portanto, uma base de dados também de *super nó*.

Assim, entra um *sub nó* com características de *super nó*, o novo estado que havíamos referido. Este apresenta todos os comportamentos normais de *sub nó*, não sendo possível, contudo, desempenhar funções iguais a *super nós*. A sua actividade como *super nó* está bastante limitada: não pode ter filhos pois geraria uma árvore de filhos complexa, sendo prejudicial ao algoritmo e não necessita de avisar que está vivo a nenhum nó, funcionando como uma espécie de *super nó* inactivo.

Desta forma, o algoritmo de junção tem que ser alterado para os casos em que algum nó contacta um *super nó* inactivo para se acoplar a ele no sistema. Ao ser contactado para albergar um nó como seu filho, um *super nó* deste tipo reenvia uma mensagem a esse participante com o *endpoint* do seu pai. O tamanho dessa mensagem seria o tamanho de um *endpoint*, 6 bytes mais o *hash* do filtro de 16 bytes e o , sempre presente, tamanho da tipificação da mensagem. Ao receber essa mensagem, o nó que quer entrar no sistema contacta o devido *super nó* referido, juntando-se ao mesmo utilizando os métodos já descritos.

Através deste redireccionamento consegue-se que estes *super nós* inactivos nunca tenham filhos, a não ser que sejam promovidos, mantendo a estrutura do sistema a mais simples possível. De notar que, se estes aceitassem filhos, a complexidade do sistema iria aumentar bastante, tal como a dos algoritmos. É de relembrar que em redes *peer-to-peer* normalmente pretende-se simplicidade para uma maior eficácia da aplicação.

4.4 Dimensionador do Sistema

A implementação deste mecanismo reside, principalmente, nos sequenciadores que iniciam a disseminação de novas entradas no sistema, sendo esperado que sejam os nós mais trabalhadores e que gastam mais largura de banda.

Desta forma, os sequenciadores irão ser os nós que executam este mecanismo. Para além de avaliarem o sistema em termos de promoções, e consequentes gastos próprios

de *upload*, estimarão que evolução a rede poderá apresentar com o intuito de executar de forma óptima, dentro do limite de largura de banda previamente definido. A implementação deste mecanismo está dividida em três fases: a avaliação do sistema, a geração e obtenção de *slots* e a disseminação dos mesmos. Os detalhes de implementação destas fases serão descritos seguidamente.

4.4.1 Avaliação do Sistema e Cálculo da Evolução

Começando pelos princípios iniciais do algoritmo, é necessário avaliar os nós. Esta avaliação tem duas componentes: a largura de banda gasta em *upload*,e o número de promoções detectadas pelo nó, no sistema. Consideramos uma promoção uma entrada normal de um *super nó* de filtro novo, ou a promoção de um nó cujo representante saiu do sistema. Assim, cada nó sequenciador do sistema guardará estas estatísticas, tanto da sua utilização de *upload*, como do número de nós que entraram na visibilidade completa de maneira a poder analisar o sistema e inferir a sua evolução óptima. Estas são guardadas nos objectos de tráfico por um *double* e um inteiro, respectivamente.

Para efectuar uma avaliação periódica da rede por parte de cada sequenciador, foi utilizada uma tarefa periódica com um intervalo de 5 minutos. Inicialmente, essa tarefa verifica se o nó ainda é sequenciador, visto estes mudarem durante a execução do algoritmo, impedindo assim outros nós de efectuarem alterações ao sistema. Seguidamente são obtidas as estatísticas de promoções e *upload* desde a última execução da tarefa. Com isto, iniciam-se os cálculos com o intuito de inferir a possível evolução da rede.

Primeiro, é calculado o *upload* não utilizado, subtraindo ao limite de *upload* definido na configuração, o que foi utilizado desde a última análise. Como o custo do sistema em termos de filiação é baseia-se nas entradas e saídas, o *upload* gasto é dividido pelo número de promoções desde a última análise, de modo a encontrar o custo de cada promoção para o sequenciador. Com estes dois dados estatísticos, consegue-se estimar, aproximadamente, com o *upload* restante, quantos *super nós* poderiam ter entrado a mais dentro dos limites estabelecidos, dividindo este excesso pelo custo de promoção. Assim temos o número de nós excedentários que o sistema suportaria dentro do *budjet* definido, sem apresentar custos demasiados para a rede.

Contudo, esta avaliação não está concluída. Desta maneira foi calculado o número de participantes que poderiam entrar para a rede de *super nós* mas, tendo em conta que poderão existir vários sequenciadores a avaliar o sistema, cada um apenas introduzirá alterações no sistema dentro da sua responsabilidade. Assim, o número de *slots* de promoção calculado anteriormente é dividido pelo número de *slices* do sistema, pois existe apenas um nó sequenciador em cada. Com isto, teremos o número de nós que deverão entrar no sistema, ou *slots* de promoção, da responsabilidade de determinado sequenciador.

4.4.2 Geração e Obtenção de *Slots*

Após a recolha das estatísticas e posterior cálculo do número de *slots* que o sistema deverá utilizar para ter uma evolução calculada, é necessário gerar estes *slots* de modo a que os sequenciadores os possam coleccionar.

Para gerar os *slots* ao longo do tempo, em vez de os criar logo após o cálculo, é estimada uma taxa de entrada em termos da periodicidade da tarefa de avaliação da rede, 5 minutos, e o número de *slots* calculado. Essa taxa de entrada é utilizada como argumento para uma distribuição exponencial, que gera intervalos de tempo para as entradas dos nós, de acordo com a taxa fornecida.

Estes intervalos de tempo são combinados com uma tarefa do sistema. Esta é programada para correr tendo em conta o próximo intervalo de tempo gerado. Quando a tarefa é executada, é adicionado um *slot* ao número destes já gerados, sendo gerado outro intervalo de tempo e agendada a execução da tarefa para o final dessa janela temporal. Assim as entradas são geradas ao longo do tempo e não automaticamente na avaliação do sistema por parte dos sequenciadores.

A obtenção destes *slots* para posterior disseminação dos mesmos pelo sistema é feita pelos sequenciadores, aquando da execução da tarefa de disseminação das entradas e saídas agregadas pelos mesmos, dentro da sua fatia. Nessa altura é verificado o número de *slots* criados, sendo estes coleccionados por parte do sequenciador e apagados da tarefa de geração dos mesmos.

De notar que quando é calculada uma nova taxa de entrada de *slots*, todos os que foram gerados mas não consumidos, são eliminados, de maneira a manter a evolução equilibrada do sistema. Aquando do cálculo de uma nova taxa, é criada uma nova distribuição exponencial que é agregada à tarefa periódica de geração de *slots*.

4.4.3 Disseminação de Slots

Após serem consumidos os *slots* por parte do sequenciador, este necessita de os espalhar pelo sistema de modo a criar promoções segundo o intuito deste mecanismo. Para tal foi criada uma nova mensagem idêntica à de filiação, à qual de junta a informação de quantos *slots* existem por disseminar e estão disponíveis para ser recolhidos. Assim, o tamanho desta mensagem será basicamente o da geral do sistema *LiveFeeds* mais 4 bytes para o inteiro referente aos lugares de promoção.

De seguida esta nova mensagem é disseminada segundo o algoritmo descrito no capítulo de desenho, que tem por base o algoritmo de filiação, sendo adaptado de acordo com a existência, ou não, de *slots* de promoção. Assim, quando não existe lugar a nenhuma promoção por parte deste mecanismo, é utilizado o algoritmo de filiação anterior, substituindo a nova mensagem pela anterior.

4.4.3.1 Promoção de um filho

Ao consumir um *slot* na disseminação atrás explicada, um *super nó* inicia uma tarefa que visa a promoção de um dos seus filhos. Este escolherá, aleatoriamente, um filho da sua lista de *sub nós*, enviando-lhe uma mensagem de promoção do tamanho de 4 bytes. Caso o envio falhe, é escolhido outro filho até a mensagem ser enviada a um filho *online*, terminando quando não existirem *sub nós* desse representante.

Um *sub nó*, ao receber esta mensagem de promoção, cancela a sua tarefa para se auto promover caso o pai falhasse, muda o seu estado para *super nó* e entra no sistema como mais um representante do filtro. Se este apresentar o estado de *super nó* inactivo, ele apenas deixa a ligação pelo pai, iniciando as actividades de *super nó*, entrando automaticamente no sistema. Caso contrário, este tem que fazer o *download* de *endpoints*, filtros e anunciar-se ao sequenciador da sua *slice* de maneira aos outros *super nós* o conhecerem.

4.5 Disseminação Filtrada

Após a implementação de todo o sistema e mecanismos auxiliares da filiação no sistema, será agora desenvolvido o método de disseminação de conteúdo pretendido para este. Como já foi descrito na fase de desenho, foi dividido em duas fases distintas. Primeiramente, é necessário garantir a entrega a todos os nós do sistema, visto que a nova arquitectura impossibilitava tal facto. Seguidamente, pretende-se implementar mecanismos de tolerância a falhas, tornando a entrega fiável e eficaz.

O algoritmo anterior contemplava a análise dos nós pertencentes à visibilidade completa, entregando o desenvolvimento do algoritmo e o evento aos participantes cujos filtros aceitassem esse conteúdo. Como a arquitectura do sistema foi alterada, são necessários mecanismos que completem este algoritmo, de modo a se obter os resultados pretendidos.

Para aproximar a nova definição a um sistema real, foram desenvolvidos novos filtros. Até agora, os filtros definidos eram simples, um número distinto de todos os outros existentes. De maneira a simular de maneira mais real a disseminação de conteúdos, foram utilizados definições que se podem englobar umas às outras. Ou seja, tecnicamente dois nós com um filtro podem receber o mesmo conteúdo, caso a definição de um esteja englobada noutro. Este comportamento é o esperado em sistemas deste tipo, visto as definições de filtro não serem, naturalmente todas disjuntas. Este método também reflecte mais a normalidade de funcionamento de sistema deste tipo pois, com esta definição, um nó pode receber conteúdos diversos e diferentes, o que não era permitido pela anterior.

De seguida, as listagens 1 e 2 exemplificam a implementação de um filtro abstracto e de um evento em concreto neste sistema. Através da análise destas, entende-se melhor o funcionamento o algoritmo em relação à aceitação de eventos.

Listing 4.1: Classe de Filtro abstracta

```
1
```

```
2  abstract class Filter implements Serializable {
3
4  private static final long serialVersionUID = 1L;
5  public long filter;
6
7  public long getFilter() {
8    return filter;
9  }
10
11 public Filter() {
12 }
13 public Filter(long filter) {
14   this.filter = filter;
15 }
16
17 final double mask = getFilter() ;
18 public boolean accepts( Event e ) {
19   return ((Event1)e).value < mask ;
20 }
21
22 @Override
23 public boolean equals(Object obj) {
24   if (this == obj)
25     return true;
26   if (obj == null)
27     return false;
28   if (getClass() != obj.getClass())
29     return false;
30   Filter other = (Filter) obj;
31   if (filter != other.filter)
32     return false;
33   return true;
34   }
35 }
```

Listing 4.2: Classe de Evento abstracta

```
1
2  abstract public class Event implements Serializable {
3    private static final long serialVersionUID = 1L;
4    public final long src ;
5    public final int index;
6
7    public double value = rg.nextLong() ;
8
9    public Event( long srcKey, int index ) {
10     this.index = index;
11     this.src = srcKey ;
12   }
13
14   private static int g_serial = 0 ;
```

```
15    public final int serial = g_serial++;
16
17    public String toString() {
18      return "" + value ;
19    }
20  }
```

4.5.1 Entrega a *Sub Nós*

Aquando da recepção de um novo evento, o *super nó* em causa tem de enviar esta informação aos *sub nós* dependentes dele. Para isto foi implementada uma tarefa que, após a recepção de um evento por parte de um *super nó*, escolhe aleatoriamente um filho deste para lhe comunicar o evento. Para tal, cria uma mensagem com a informação relevante desse evento para os seus filhos, que é composta pela lista de *endpoints* dos *sub nós* restantes e o evento em causa. O tamanho desta mensagem é o tamanho do evento definido por configuração do sistema, somando 6 bytes por cada *endpoint*. Esta mensagem é enviada ao *sub nó* escolhido.

Caso o envio falhe, este processo é repetido até não haver filhos, sendo os *endpoints* dos nós falhados removidos. Ao receber a mensagem, o *sub nó* faz exactamente o mesmo que o *super nó*, criando uma nova tarefa e uma nova mensagem, removendo da lista de *endpoints* o novo *sub nó* escolhido aleatoriamente. De seguida envia a mensagem para o nó escolhido. caso falhe, a tarefa é corrida novamente, como no caso do *super nó*. Este processo decorre até a lista de entregas estar vazia.

4.5.2 Recuperação de Eventos

Como já foi referido no capítulo de desenho, existem alguns tipos de falha possíveis no algoritmo de difusão, sendo necessários mecanismos para recuperar eventos perdidos. Estas metodologias terão que ser aplicadas tanto a *super nós*, como a *sub nós*. Contudo, as abordagens irão diferir em alguns pontos, devido à diferença arquitectural entre os dois tipos de falhas. A ideia da recuperação de eventos é simples: os nós periodicamente comunicam com outros, de maneira a verificar que eventos faltam a ambos e posteriormente consumi-los.

Como base para este mecanismo de reparação, aparecem as informações de histórico de eventos e nós vizinhos. O histórico é implementado segundo uma lista de inteiros, sendo adicionado o inteiro correspondente ao identificador do evento recebido. Os nós vizinhos são, também, implementados segundo uma lista. Esta lista guardará os *endpoints* contactados e *super nós* conhecidos desse mesmo filtro. Ao adicionar um nó contactado, o conhecimento dos *super nós* existentes é avaliada, inferindo a necessidade de adicionar mais *endpoints* de *super nós* conhecidos com essa definição de filtro.

O contacto epidémico entre *super nós* é implementado segundo uma tarefa periódica com intervalos de tempo configuráveis. Na execução desta tarefa é escolhido um nó aleatório da lista de vizinhos para estabelecer o contacto de reparação. Escolhido o receptor,

é criada uma mensagem com o histórico, sendo-lhe enviada essa informação. O tamanho da mensagem depende do tamanho do histórico, gastando 10 bytes por cada identificador presente. Este consiste em 6 do *endpoint* mais 4 do número de sequência.

Ao receber esta mensagem, um nó inicia uma tarefa onde avalia as diferenças de identificadores através de remoção de listas umas das outras. Este também avalia quais dos identificadores restantes, que possui e que faltam ao outro participante, é que esse deve receber. Tal é feito por sujeitar os eventos que tem e que faltam ao outro nó com quem comunicou, ao filtro desse mesmo nó, conhecido pela visibilidade completa. Os eventos aceites pelo filtro, e em falta, são adicionados a uma lista.

Nesta tarefa é ainda criada uma mensagem de resposta que contém essa lista de eventos e a lista de identificadores que o primeiro nó tem que este não possui. Essa mensagem é enviada como resposta. O tamanho desta mensagem tem em consideração o número de identificadores em falta, multiplicando esse factor por 10 bytes, e ainda o tamanho do evento, configurável pela aplicação, multiplicado pelo número de eventos enviados.

Ao receber esta mensagem, o nó que iniciou o contacto consome os eventos em falta pelo método normal, e expõe os eventos dos identificadores recebidos ao filtro do nó oposto. Assim é construída uma lista com os eventos que o outro nó irá receber. A tarefa de seguida envia uma mensagem igual à anterior, mas sem identificadores, caso exista algum evento por receber. Ao receber essa suposta mensagem, o nó consome os eventos pelo método normal, acabando a reparação e sendo agendada novo contacto periódico através de uma tarefa.

Estando implementado o contacto entre *super nós*, a reparação entre irmãos é simplificada. O método é exactamente o mesmo, excepto a escolha dos nós a comunicar por parte da tarefa. Neste caso, o contacto é feito para um *sub nó* irmão aleatório. Existe uma verificação na tarefa de reparação que verifica se o *sub nó* já foi promovido. Caso tal tenha acontecido, a tarefa de *sub nó* é acabada e iniciada uma de *super nó*.

De notar que se algum destes contactos falhar inicialmente, a tarefa é agendada para correr no momento seguinte, escolhendo outro nó aleatório da lista que está em questão.

5

Avaliação Experimental

Neste capítulo será apresentada a avaliação experimental do trabalho realizado que foi descrito anteriormente. De modo a efectuar esta avaliação, serão desenvolvidas simulações do funcionamento dos mecanismos do sistema, sendo os resultados recolhidos e analisados. A partir destas métricas serão tecidas conclusões acerca do funcionamento do trabalho.

Esta avaliação será feita faseadamente, de acordo com os mecanismos desenvolvidos. Primeiro será avaliado o funcionamento do novo método de filiação do sistema, baseado em *super nós* e na redundância de filtros. Seguidamente, a avaliação efectuada recai sobre o mecanismo de tolerância a falhas, coexistindo até *K super nós* para o mesmo filtro. Posteriormente será avaliado o mecanismo dimensionador do sistema, que relaxa o número de *super nós* através de um mecanismo de *budjet*. Finalmente a avaliação experimental recairá sobre o mecanismo de recuperação de falhas ao nível da filiação.

5.1 Solução de Referência

De modo a avaliar o custo do novo sistema, é necessário recordar o gasto, em termos de largura de banda, da solução base adoptada. Isto deve-se à dependência do funcionamento base do sistema *LiveFeeds*, por parte da nova arquitectura. A arquitectura base é utilizada apenas no funcionamento dos *super nós* da nova solução, sendo, desta maneira, uma parte bastante significativa dos custos da nova arquitectura.

Inicialmente pretende-se, neste trabalho, avaliar o funcionamento do novo sistema de filiação desenvolvido. Para tal, dever-se-à comparar os seus resultados experimentais aos obtidos pelo sistema base, de forma a entender os melhoramentos proporcionados pelo mesmo. Assim, inicialmente, serão demonstrados os custos do sistema base, e a

Parâmetros	
Taxas de Entrada	1.4 nós/s / 1.05 nós/s / 0.7 nós/s
Tempo médio de sessão	4 horas
Tempo máximo de sessão	8 horas
Endpoint	6 bytes
Filtro	506 bytes
Tempo de simulação	23 horas
Warmup	8 horas

Tabela 5.1: Parâmetros de simulação

sua composição. De forma a estabelecer uma comparação, foram recolhidas métricas relativas ao funcionamento do algoritmo de filiação base, para três taxas de entradas de nós diferentes. Os parâmetros destas simulações estão presentes na tabela 5.1.

A taxa de entrada representa o ritmo a que novos nós entram no sistema, para cada simulação efectuada. O tempo médio e máximo de sessão são os parâmetros utilizados para definir o tempo de sessão de cada nó, de acordo com uma distribuição estatística. Esta garante que estas definições são cumpridas, enquanto gera tempos aleatórios de sessão, dentro dessa distribuição, para os novos nós. O espaço gasto para referenciar um *endpoint* são 6 bytes, sendo que o espaço gasto para um filtro são 506. O tempo de simulação refere-se a quanto tempo o sistema estará activo, durante o qual entrarão e sairão nós. O tempo de *warmup* é utilizado logo na inicialização do sistema, durante a qual este estabilizará. No final desta janela temporal, inicia-se a recolha de métricas.

Efectuadas estas simulações para o sistema base, foi obtida a figura 5.1 relativamente à largura de banda utilizada.

Ao analisar a figura 5.1 entende-se que a largura de banda utilizada é aproximadamente proporcional à taxa de entrada de nós no sistema, para a mesma dimensão dos eventos utilizados. Denota-se, também, que o gasto de *upload* e *download* tende a ser igual para sessões mais longas, sendo que nestas o custo médio é similar. Para as sessões mais baixas, os gastos são mais altos por duas razões.

Primeiro, os nós que estão pouco tempo no sistema não têm tempo para diluir o seu custo de entrada ao longo do tempo de sessão. Assim, os gastos médios para estes são significativamente superiores aos outros. A outra razão prende-se com o facto de o sistema *Catadupa* compensar os gastos dos nós com maiores sessão, proporcionando trabalho aos que estão à pouco tempo na rede. Tal é feito pelo ajuste dinâmico na filiação das árvores de disseminação por parte dos *super nós*, tendo em conta os seus gastos, fazendo os nós recentes trabalhar mais na árvore. Outra razão surge com a ideia de que os nós mais recentes com racio desfavorável (por serem recentes), em termos de *upload*/*download*, servem *endpoints* e filtros com maior probabilidade . Esta metodologia é adoptada para premiar os nós que estão mais tempo na aplicação, beneficiando esta com este facto. Assim, nós que entrem, e pouco depois saiam do sistema são prejudicados com grande carga de trabalho, demonstrada no aumento das curvas verificado para sessões curtas.

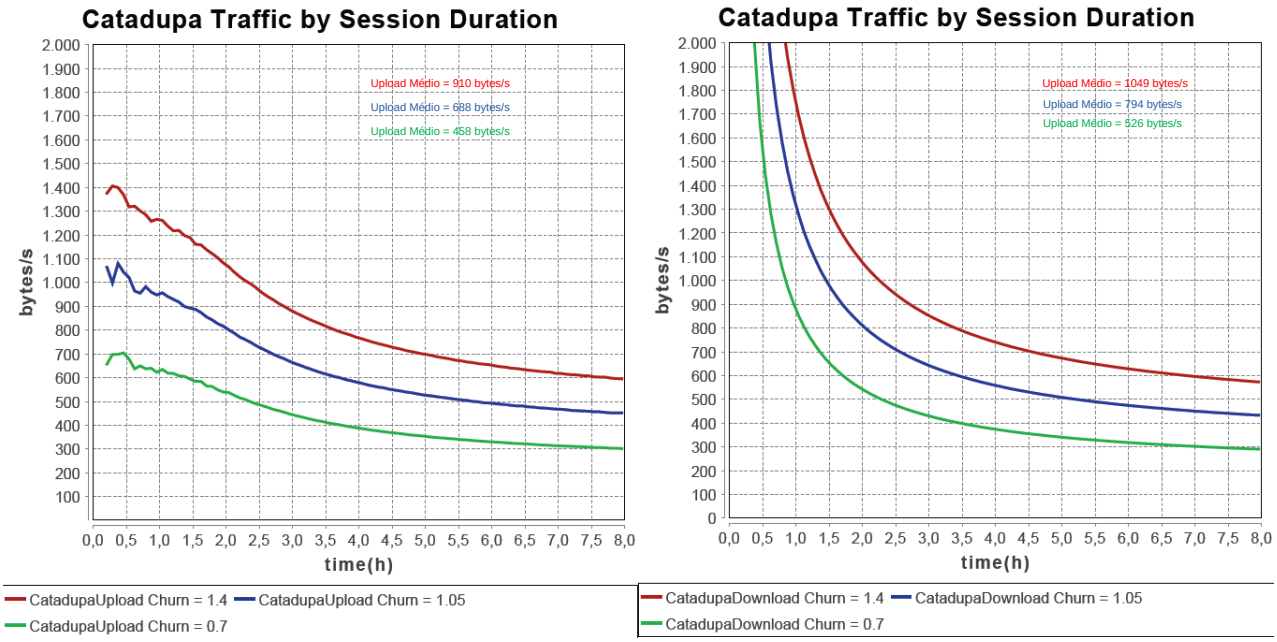


Figura 5.1: No primeiro gráfico o *upload* médio para várias taxas de entrada e no segundo o *download*

Será interessante, também, verificar a composição do custo do sistema base. Esse está estratificado na figura 5.2 para o sistema com taxa de entrada de 1,4 nós/s. De notar que o custo está expresso em valores exponenciais para estas figuras que mostram o tráfego no sistema.

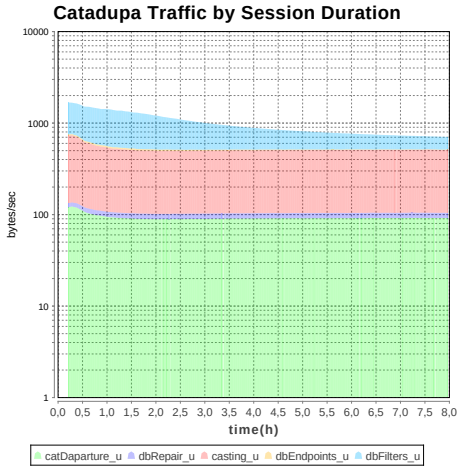


Figura 5.2: Custo de *upload* estratificado para os parâmetros da 5.1

Ao analisar a figura 5.2, verifica-se que o custo reside principalmente na disseminação de entradas e saídas do sistema e *upload* dos filtros. Os dois primeiros estão directamente relacionados com o *churn*, mostrando que a grande parcela do custo está relacionada com a taxa de entrada de nós. A epidemia desenvolvida para a reparação tem

Parâmetros - Nova Arquitectura	
Tempo médio de sessão	4 horas
Tempo máximo de sessão	8 horas
Endpoint	6 bytes
Filtro	506 bytes
Keep-Alive	45 em 45 segundos
Tempo de Simulação	23 horas
Warmup	8 horas

Tabela 5.2: Parâmetros de simulação - Nova Arquitectura

um custo menos significativo, como verificado na figura. Ao verificar estes resultados obtidos, denota-se que o custo nunca depende da dimensão do sistema, mas sim, principalmente, da taxa de entrada de nós.

5.2 Resultados Experimentais da solução baseada em *Super Nós*

Terminada a análise ao funcionamento do sistema base *LiveFeeds*, foram desenvolvidos variados testes de maneira avaliar o funcionamento da nova arquitectura. Os resultados serão expostos e analisados, de maneira a compreender como é que a solução se comporta num sistema simulado. As métricas recolhidas permitirão elaborar conclusões sobre a viabilidade e funcionamento desta nova arquitectura. Em cada fase, as métricas relevantes serão analisadas e comparadas.

5.2.1 Fase Inicial - Nova Estrutura de *Super Nós*

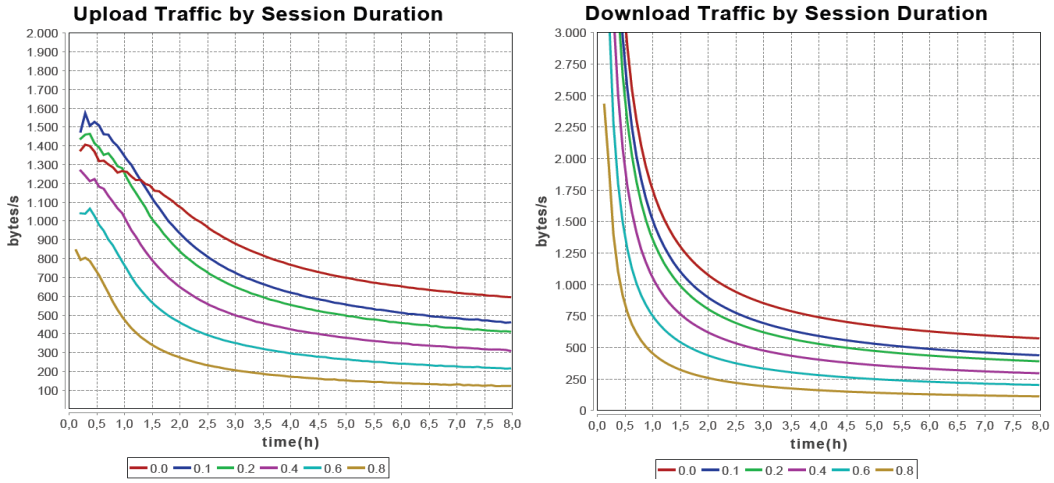
Iniciaremos a análise de resultados pela primeira parte do projecto, a estruturação da nova arquitectura baseada em *super nós*. Nesta secção, a análise terá como base a execução e resultados do sistema original, servindo este como base de comparação de modo a inferir os possíveis melhoramentos da nova solução. Seguidamente a redundância de filtros será variada, para o novo sistema, e as várias métricas medidas analisadas. Serão desenvolvidas duas simulações base, avaliando o novo sistema comparativamente a ambas, cada uma delas diferenciadas pela taxa de entrada e dinamismo do sistema.

Assim, as simulações para a nova arquitectura terão as taxas de entrada de 1,4 nós/s e 1,05 nós/s, sendo o tamanho estimado dos sistemas 20 mil e 15 mil nós, respectivamente. Para estas taxas de entrada foi variada a redundância de filtros, utilizando os parâmetros definidos na tabela 5.2.

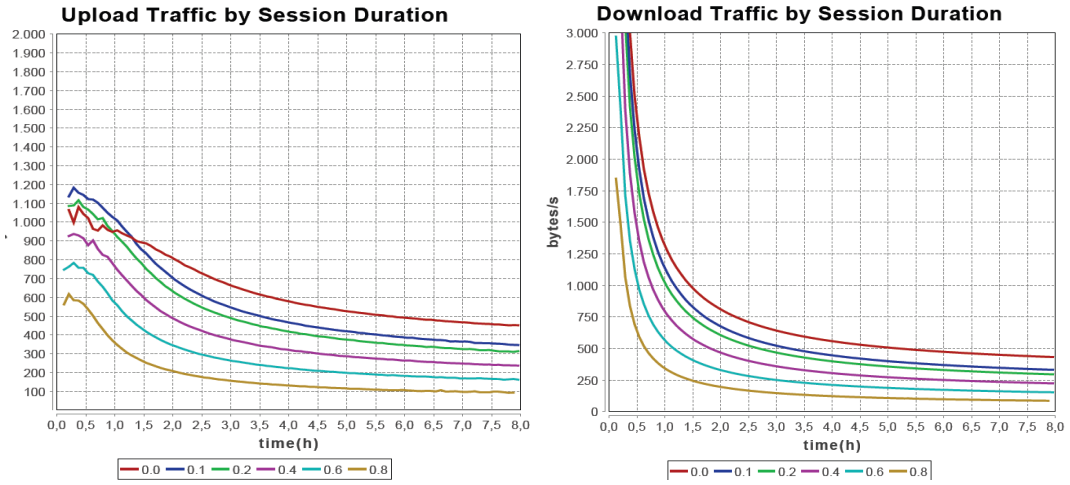
Nesta tabela 5.2 aparece um parâmetro novo em relação à simulação anterior, a periodicidade dos *keep-alive*. Este é o espaço temporal definido entre a comunicação por parte do pai que está *online*, alertando um dos seus filhos, suportando o mecanismo de promoções do sistema.

5.2.1.1 Análise de Custo

Na figura 5.3 são apresentados os gastos de *upload* e *download* para as várias redundâncias de filtros. Estes dados foram recolhidos para as duas taxas de entrada previamente referidas.



(a) *Churn* = 1.4



(b) *Churn* = 1.05

Figura 5.3: *Upload* e *download* por tempo de sessão, para diversos rácios de redundância de filtros (cada cor corresponde a um rácio distinto).

Ao analisarmos a figura 5.3, denota-se que, com o aumento da redundância de filtros, o custo médio por sessão baixa. Tal acontece para ambas as taxas de entrada testadas, demonstrando que a nova arquitectura reduz a largura de banda utilizada no sistema. Ao analisar as curvas referentes aos gastos do sistema com a nova arquitectura, realça-se que a diferença entre as redundâncias e os gastos na curva sugerem uma variação aproximadamente linear.

Estes resultados são equivalentes para o *upload* e *download* em ambos os *churn* simulados. Nota-se, também, uma proximidade dos gastos de *upload* e *download* para tempos

de sessão mais altos, sugerindo que estes gastos devem ser idênticos. Assim, pode-se concluir que a nova arquitectura supera os problemas que se tinha proposto resolver, diminuindo significativamente a largura de banda utilizada na filiação.

Verifica-se, porém, que para tempos de sessão muito baixos, as taxas de redundância mais baixas gastam mais largura de banda do que a solução base. Este comportamento dos nós com sessões curtas irá influenciar a largura de banda média gasta por nó, independentemente da sessão. Contudo, não deverá ter grande relevância pois, como já foi explicado, o custo dos nós com sessões mais pequenas é aumentado propositadamente, para favorecer nós com sessões maiores, que são ideais ao funcionamento do sistema.

Assim sugere-se que a nova arquitectura consegue suplantar o seu propósito, reduzindo os custos de largura de banda na filiação. Quanto maior a redundância de filtros do sistema, melhores os resultados obtidos por esta. Porém, para redundâncias muito baixas, o funcionamento é satisfatório e mais barato comparando com a solução base, excepto para nós com sessões muito pequenas e redundância abaixo de 0,2. É de notar que esta melhoria parece manter-se independentemente da taxa de entrada de nós no sistema, mostrando que o dinamismo de entrada é controlado pela arquitectura.

De forma a analisar estes resultados de uma forma mais global, foi avaliada a relação entre o gasto médio dos nós do sistema, com a redundância da rede onde estão a actuar, para as duas taxas de entrada testadas, tal como se apresenta na figura 5.4.

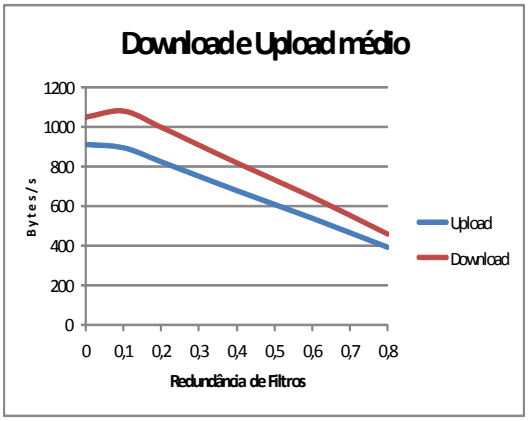
Analisando as figuras 5.4 de largura de banda média gasta por nó, confirmamos a tendência descrita anteriormente. Os custos do sistema baixam com o aumento da redundância, trazendo ganhos significativos. Excepção feita à redundância de 0,1, em que o peso dos nós com sessões mais baixas faz com que os gastos médios sejam ligeiramente mais altos do que a solução base.

Nota-se, também, para ambas as taxas de entrada, a proximidade do *upload* e *download*, tendo curvas idênticas. Um pormenor importante verificado é a similaridade da inclinação das curvas de largura de banda gasta, entre as duas taxas de entrada utilizadas.

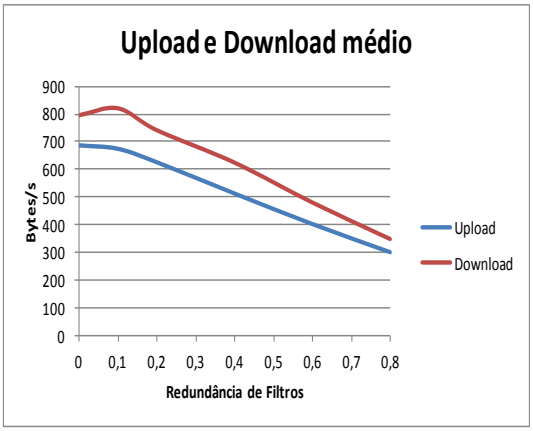
Como havia sido descrito, o custo do sistema anterior dependia directamente do *churn* de entrada de nós do sistema. De forma a verificar tal facto para a nova arquitectura, foi desenvolvida a figura 5.5, onde se podem observar as taxas de entradas de *super nós*, de acordo com a redundância de filtros.

Ao analisar a figura 5.5, verifica-se o esperado: os gastos médios do sistema variam exactamente segundo a taxa de entrada de *super nós* no sistema, tendo em conta a redundância de filtros. Ou seja, a taxa de entrada de *super nós* desce com o aumento da redundância, para ambas as taxas de entrada de nós na rede. Denota-se, também, que a inclinação das duas curvas parece igual, sendo também semelhante às dos gastos de largura de banda média.

Outra conclusão interessante que se pode tirar desta figura 5.5, é a de que a taxa de entrada de *super nós* no sistema será maior que a percentagem de nós deste tipo na rede, a cada momento. Teoricamente, existindo uma certa redundância de filtros, o seu



(a) *Churn* = 1.4



(b) *Churn* = 1.05

Figura 5.4: *Upload* e *download* médio por nó, variando a redundância de filtros, para duas taxas de entrada distintas

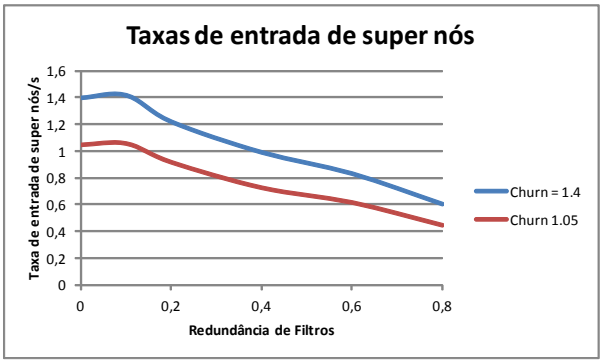


Figura 5.5: Taxa de entrada de *super nós* variando a redundância de filtros, para dois *churn* diferentes

complementar será a percentagem de *super nós*. Assim, assiste-se a uma divergência entre estes dados e o valor teórico. Tal dever-se-à ao funcionamento do mecanismo de promoção, fazendo com que, durante o seu tempo de vida, um *super nó* conheça os nós iniciais mais os promovidos. Estes dados serão escrutinados mais a fundo na secção

5.2.1.2.

De forma a estudar esta igualdade nas inclinações das curvas, foi desenvolvido um estudo comparativo entre a taxa de entrada inicial e a taxa de entrada de *super nós* verificada. Esta relação está evidenciada na figura 5.6. Estas taxas estão apresentadas em nós/s.

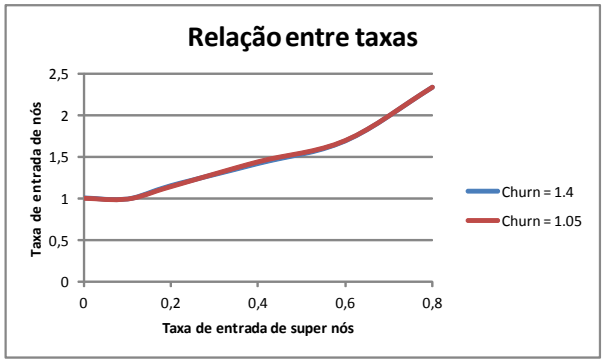


Figura 5.6: Relação entre taxa de entrada de nós com a de *super nós*

O estudo a figura 5.6 sugere uma extrapolação bastante interessante para o funcionamento do sistema. Significa que, para cada redundância de filtros utilizada pela nova arquitectura, a relação entre estas taxas é constante. Com este resultado pode-se prever a taxa de entrada de sistemas com outras dimensões, estimando também a ordem do custo. Assim, conclui-se que a taxa de entrada de *super nós* não depende da taxa de entrada de nós no sistema, dependendo apenas da redundância de filtros dentro do mesmo.

Para terminar a análise do custo da nova arquitectura, falta referir a largura de banda gasta pelos mecanismos de estruturação do sistema e promoção de nós. Para ambas as taxas de entrada testadas, verificou-se que os custos eram praticamente desprezáveis, na ordem de valores unitários. Outra conclusão tirada foi que estes custos apenas eram dependentes da redundância de filtros e não de qualquer taxa de entrada no sistema. Tal verificou-se pois os gastos eram semelhantes para cada redundância, com as diferentes taxas de entrada.

5.2.1.2 Composição da nova arquitectura

Após a análise comparativa dos custos da nova solução, surge a questão de como é composta a rede com a utilização da nova arquitectura. Ou seja, que percentagens de nós haverá de cada tipo, para as variadas redundâncias testadas.

Analisando teoricamente esta questão, facilmente se entende que a percentagem de *super nós* no sistema deverá ser igual à percentagem de filtros diferentes, ou seja, o complementar da redundância de filtros presente no sistema. Tendo isto, a percentagem de *sub nós* será o complementar dos *super nós*, ou seja, igual à taxa de redundância de filtros no sistema. Com base nestas conclusões, pode-se determinar o número de filhos esperado para cada *super nó*, dependendo da redundância de filtros na rede. Definindo

Redundância	SN estimado	Filhos estimado	SN	Filhos	P(SN)
0,2	0,80	0,25	0,75	0,31	0,77
0,4	0,60	0,66	0,59	0,68	0,67
0,6	0,40	1,50	0,41	1,51	0,59
0,8	0,20	4,00	0,23	3,55	0,42

Tabela 5.3: Composição do sistema para *Churn* = 1,4, variando a redundância de filtros

sn como a taxa de *super nós* no sistema, *sb* como a de *sub nós* e *R* para a redundância, o número de filhos *fil* será:

$$fil = \frac{sb}{sn} = \frac{R}{1-R} \tag{5.1}$$

É interessante verificar que o número de filhos por *super nó* não dependerá da taxa de entrada de nós do sistema, nem do número de nós presentes neste. Esta apenas varia dependendo da redundância de filtros, de acordo com a expressão atrás obtida. Tal traz uma estabilidade ao sistema tendo em conta a sua estruturação, mostrando que independentemente de taxas de entrada, em média, este, para cada redundância, comportar-se-à da mesma maneira.

De maneira a comprovar estes resultados teóricos, estabelecemos uma comparação com os resultados experimentais obtidos, na tabela 5.3. O último resultado é a probabilidade de um nó vir a ser *super nós*, durante todo o seu tempo de sessão. Estes dados foram recolhidos das simulações com taxa de entrada de 1,4 nós/s.

Ao analisar os resultados verificamos que estão perto dos resultados teóricos obtidos. As flutuações para cima da taxa de *super nós*, com consequente diminuição de número médio de filhos, devem-se à taxa de nós concorrentes que entraram no sistema. Essa métrica será analisada na secção 5.2.1.3. A diminuição da taxa de *super nós* para redundâncias mais pequenas deve-se à aleatoriedade da escolha do filtro, que, havendo muitos filtros para escolher, podem alguns ficar de fora de maneira aleatória. De notar também, que a métrica recolhida refere-se a uma média, havendo valores superiores e inferiores.

A percentagem de nós da rede que, em alguma altura da sua presença no sistema, foram *super nós* é interessante, demonstrando o funcionamento das promoções do sistema. Sugere, também, que cada nó não conhecerá apenas a percentagem de *super nós* na rede a cada momento, mas sim um número superior de participantes, devido à promoção de novos nós aquando da saída de representantes. Ao ser promovido o *super nó* conhecerá automaticamente a percentagem de *super nós* presentes nesse momento no sistema, à qual serão adicionados todos os *sub nós* que se promovem durante o tempo de sessão deste.

Multiplicando as taxas de *super nós* pelo custo da solução base, verifica-se que o gasto de largura de banda não coincide com essas percentagens. Assim, é sugerido que a taxa de entrada de *super nós* não evolui linearmente com a percentagem destes no sistema. A métrica mais perto deste resultado é a percentagem de *super nós* que estiveram no sistema e que ajudam a explicar o facto, já discutido, da diferenciação das taxas de entrada de

super nós.

Como foi explicado, um facto que poderá fazer variar a percentagem de *super nós* no sistema é a taxa de entrada de nós concorrentes. Com esta poderão ser adicionados nós suplementares por filtro, variando o número e taxa de entrada de *super nós*. Tornando-se esta métrica importante para o funcionamento do sistema, será analisada seguidamente.

5.2.1.3 Taxa de Concorrência

Nesta parte da dissertação, serão avaliadas as entradas concorrentes de *super nós* do mesmo filtro, provenientes do grande dinamismo característico de uma rede *peer-to-peer*. Para cada redundância de filtros no sistema, foi recolhida a taxa de entradas concorrentes medida em relação ao número de *super nós* total. As duas experiências foram juntas na figura 5.7, de forma a aglomerar a informação relevante.

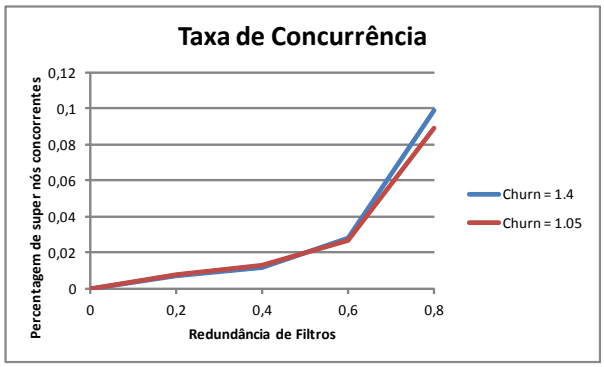


Figura 5.7: Taxas de concorrência de *super nós*

Analisando a figura 5.7, verificamos que as taxas de entrada de nós concorrentes são maiores consoante o aumento da redundância de filtros dentro do sistema. Até uma redundância de 0,5, esta taxa é bastante reduzida. Tal era esperado, devido à existência de uma percentagem grande de filtros diferentes dentro do sistema, diminuindo em muito a probabilidade de dois nós tentarem entrar com o mesmo filtro ao mesmo tempo.

Para redundâncias iguais ou superiores a 0,6, existe uma gama menor de filtros, aumentando a probabilidade de entradas concorrentes. É precisamente este comportamento que é verificado na figura 5.7 para ambas as taxas de entrada testadas.

Outra conclusão que se pode retirar é que, com o aumento da taxa de entrada, a probabilidade de entrarem vários nós ao mesmo tempo é maior. Assim, para redundâncias grandes, a probabilidade de entrada concorrente aumenta mais. Este facto é demonstrado também no gráfico, ao se observar uma maior taxa de concorrência para o *churn* mais elevado.

5.2.1.4 Análise de Tempos

Ao efectuar as experiências com a nova arquitectura, foram guardadas algumas métricas relativas a tempos de entrada no sistema, tempos para a promoção de um nó enquanto o

Redundância	0,2	0,4	0,6	0,8
Churn 1,4	2,80 horas	2,65 horas	2,39 horas	1,96 horas
Churn 1,05	2,78 horas	2,64 horas	2,42 horas	1,95 horas

Tabela 5.4: Tempo médio de estadia no sistema por redundância de filtros, para duas taxas de entrada

pai já saiu do sistema e tempos médios de estadia como *super nó*.

Este último será interessante de escrutinar, de forma a perceber algo mais sobre a taxa de entrada de *super nós* já descrita. Os outros dois tempos servirão para avaliar a evolução do sistema base para um sistema com K nós, sendo esses resultados descritos mais à frente. O tempo de entrada após uma promoção é a soma do tempo até ser detectada a saída do pai, e o de entrada no sistema de visibilidade completa.

Para ambas as taxas de entrada testadas e para todas as redundâncias, este tempo médio não varia, sendo igual a 59 segundos. Assim, percebe-se que o tempo até um nó estar disponível não depende nem da taxa de entrada de nós, nem da redundância do sistema. Deste modo, verifica-se que este valor deverá variar de acordo com a periodicidade das mensagens de *keep-alive*, definidas para 45 segundos nestas simulações. É importante ter em conta que o tempo médio para um nó entrar como *super nó*, necessitando de descarregar *endpoints* e filtros, tem um maior peso neste resultado obtido.

De seguida iremos analisar o tempo médio de estadia de um *super nó* no sistema. Estes resultados serão mostrados na tabela 5.4, variando a redundância de filtros para as duas taxas de entrada testadas. De notar que os tempos de estadia estão em horas, para analisar mais facilmente a tabela.

Através do estudo desta tabela 5.4 conclui-se que o tempo médio de estadia como *super nó* é mais reduzido do que o tempo médio de estadia no sistema global. Tal é óbvio pois, em situações normais, um nó entra no sistema como *sub nó*, sendo posteriormente promovido, ficando menos tempo como *super nó*. Nota-se, também, uma diminuição deste tempo médio relativamente ao aumento da redundância. Este facto é explicado pela existência de mais *sub nós*, demorando mais a promoção de cada um destes na rede.

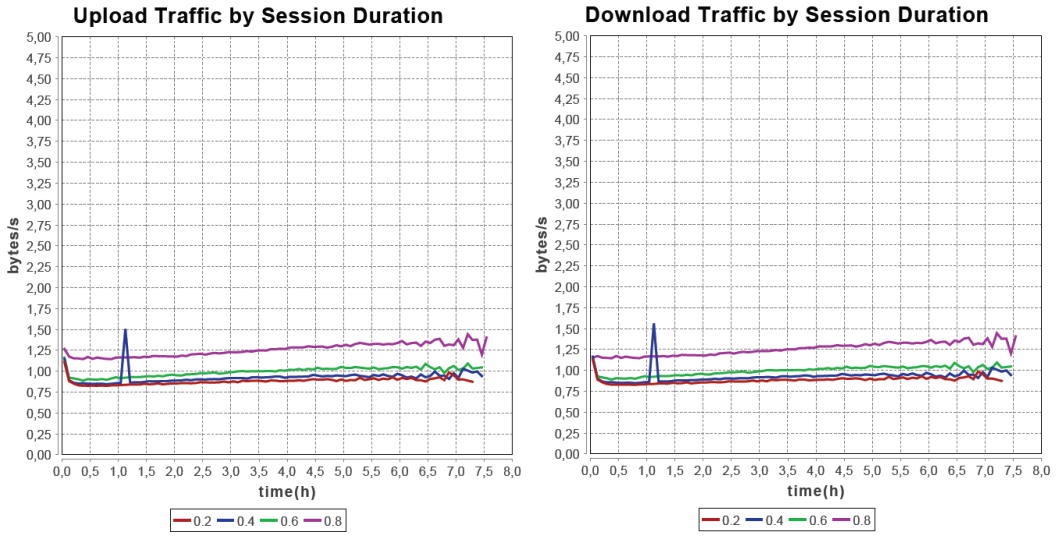
Este tempo médio de estadia como *super nó*, está intimamente ligado ao número de promoções. Estando um *super nó* mais tempo no sistema, haverá menos promoções para esse filtro. Desta maneira, percebe-se que, adoptando uma política que escolha melhor os nós, de acordo com os tempos de sessão, se pode minorar o número de promoções. Assim, baixaria a taxa de entrada de *super nós* do sistema, reduzindo os custos deste.

Apesar de o tempo de estadia dos *super nós* ser, para todas as redundâncias, menor do que o do sistema base, tal é compensado pela não entrada de muitos dos *sub nós* no sistema. Assim justifica-se que, apesar da diminuição do tempo médio no sistema, os custos desta nova arquitectura sejam menores.

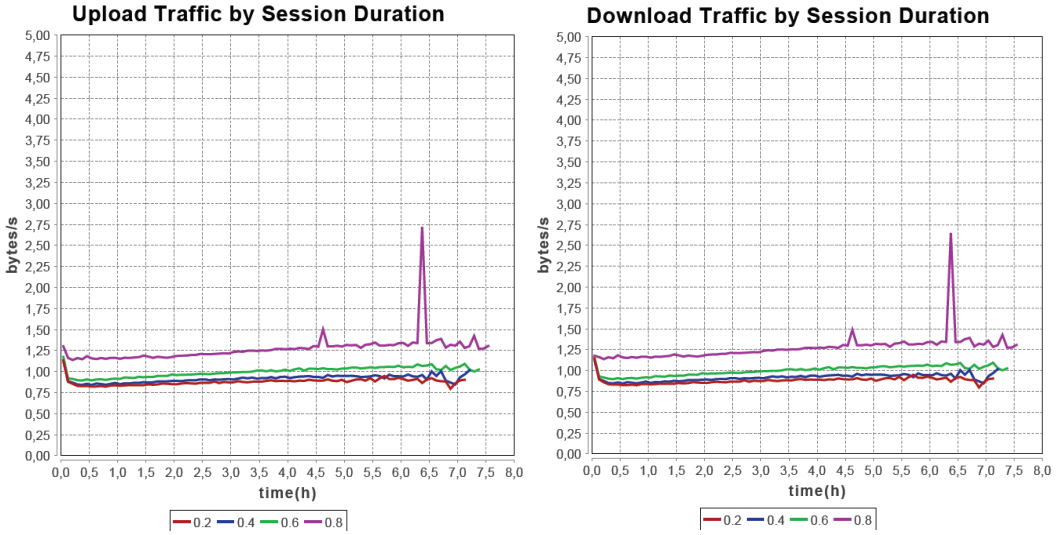
5.2.1.5 *Sub nós*

Após a avaliação do funcionamento do novo sistema para os *super nós* estar completa, serão expostos alguns resultados referentes à execução dos *sub nós* na rede. As métricas interessantes a avaliar no caso dos *sub nós*, serão os custos que o seu funcionamento suporta e a composição, em termos de fases, destes gastos de largura de banda.

Inicialmente será avaliado o custo, em termos de largura de banda gasta, para variadas taxas de redundância de filtros. Este estudo será efectuado para as duas taxas de entrada de referência. A figura 5.8 demonstra o *upload* e *download* gastos, por tempo de sessão, por parte dos *sub nós*.



(a) *Churn* = 1.4



(b) *Churn* = 1.05

Figura 5.8: *Upload* e *download* médio por *sub nó*, variando a redundância de filtros, para duas taxas de entrada distintas

Analisando a figura 5.8, verifica-se que os custos quase não variam durante o tempo

Redundância	0,2	0,4	0,6	0,8
Churn 1,4 Upload em bytes/s	0,86	0,91	0,96	1,22
Churn 1,05 Upload em bytes/s	0,86	0,89	0,97	1,23
Churn 1,4 Download em bytes/s	0,86	0,91	0,97	1,22
Churn 1,05 Download em bytes/s	0,86	0,9	0,97	1,23

Tabela 5.5: Gastos de largura de banda dos *sub nós*

de sessão, tendo as redundâncias mais altas um custo acrescido. Tal faz sentido pois, em redundâncias mais altas, os *sub nós* têm receber e enviar listas maiores de irmãos, relativamente à promoção. Tal deve-se ao aumento do número médio de filhos, já comprovado, com o aumento da redundância no sistema.

Outra característica a ter em conta é a igualdade de gastos entre *upload* e *download*. Esta igualdade faz sentido, devido ao funcionamento do sistema de promoção de nós, em que tudo o que os *sub nós* recebem, voltam a enviar. De notar que os gastos vão subindo, ligeiramente, ao longo da sessão. Isto acontece pois, quanto mais tempo estiver no sistema, mais vezes o *sub nó* terá que entrar na troca de mensagens do mecanismo de promoção, aumentando o custo.

Na figura, os gráficos exibem alguns picos estranhos ao funcionamento do mecanismo. Uma das explicações para os picos nas sessões mais altas é simples: sendo o tempo médio de um nó no sistema 4 horas, e havendo uma probabilidade de promoção visto o número de filhos estimados para as redundâncias nunca ser muito elevado, entende-se que poucos nós chegaram a tempos e sessão altos. Assim, a média perde a sua referência devido aos poucos nós que chegaram a esse estado.

Porém, existem picos (menos acentuados) em sessões mais curtas. Como os *sub nós* apenas gastam largura de banda no mecanismo da promoção e no contacto inicial, sendo este quase desprezável, entende-se que este pico será por algo referente às promoções. Como o custo das promoções é determinado pelo número de filhos, depreende-se que estes picos são devidos a algum representante que terá tido um número anormal e não espectável de filhos. Tal seria pouco provável, mas, baseando-se a escolha do filtro em métodos meramente estatísticos, tal poderá acontecer. Num sistema real isto também acontece para filtros de conteúdos muito desejados, não sendo a distribuição de nós pelos filtros igual.

De modo a avaliar melhor estes gastos de largura de banda, foram expostos os resultados médios obtidos na tabela 5.5.

Ao analisar os valores médios observados, entende-se que existe, de facto, uma quase igualdade dos custos de *download* e *upload*. Outra característica verificada é a igualdade dos valores, mesmo variando a taxa de entrada, mostrando que o custo não depende desta, mas sim apenas da redundância de filtros e número de filhos no sistema.

Os gastos crescem com a redundância de filtros devido ao mecanismo de promoção. Tal é evidente pois, aumentando a redundância, o número esperado de filhos é

maior. Com isto, a mensagem de promoção terá que guardar informação de mais *end-points*, sendo maior em termos de espaço ocupado. Assim, para maiores redundâncias, os nós partilham mensagens maiores, contribuindo para um gasto de largura de banda incremental entre redundâncias.

A conclusão mais óbvia a tirar desta avaliação de custos, é que os gastos do sistema para os *sub nós* são muito pequenos, na ordem dos poucos bytes/s, mesmo para redes grandes e dinâmicas. Tal justifica-se por estes, simplesmente, não efectuarem trabalho. Assim, fará sentido que os *super nós*, com carga bastante maior, deleguem trabalho nos seus *sub nós*, de maneira a distribuir mais equitativamente a carga do sistema. Com esta delegação, esta arquitectura passará ainda a ser mais vantajosa, com um gasto de largura de banda ainda menor.

Com esta avaliação termina a apresentação dos resultados experimentais para o mecanismo base desta nova arquitectura. Seguir-se-ão no documento resultados e avaliações desenvolvidas aos restantes mecanismos desenvolvidos para este trabalho.

5.2.2 Tolerância a Falhas - K nós

Nesta secção será avaliado o mecanismo de tolerância a falhas desenvolvido, onde existem até *K super nós* do mesmo filtro. Estes providenciam tolerância a falhas e disponibilidade do filtro, quando o pai falha. Para avaliar o total funcionamento deste mecanismo, foram realizadas quatro simulações específicas: para uma taxa de entrada de 1,4 nós/s e uma redundância de 0,8, este mecanismo foi utilizado com K = 2, 3 e 4. Ou seja, foi fixada a taxa de entrada e definido como sistema base uma simulação com 0,8 de redundância que não utiliza o mecanismo dos nós K. Esta é comparada às simulações com os variados valores para K que foram efectuadas, de modo a observar o funcionamento do mecanismo e estudar as suas características.

5.2.2.1 Número de *Super nós*

A pré-condição em que se baseia este mecanismo é muito simples, para cada filtro procura-se que existam *K super nós*. Tal não acontecerá quando apenas existem menos do que *K* participantes com esse filtro, ou quando existem mais do que esse valor devido a entradas concorrentes.

Neste mecanismo será interessante verificar como variam as taxas de *super nós* no sistema e a composição dessa mesma taxa. Ou seja, seria importante analisar a percentagem de nós *K* dentro do total de *super nós*. Outra métrica interessante é verificar quantos nós *K* existem por nó, em média. Estes dados estão representados na tabela 5.6.

Ao estudar a tabela 5.6, verifica-se o aumento, já esperado, da taxa de *super nós* no sistema. Desse aumento da taxa, as proporções de nós *K* são referidas, estando perto dos valores teóricos esperados. Assim verifica-se, também, que o número de nós *K* por *super nó* activo está perto do estimado. Em alguns casos, haverá um número menor destes nós pois, devido a promoções e número de nós filhos de cada representante, alguns filtros

Experiência	% de *Super nós*	% K	% K esperada	K/Super
Sem K	0,23	0	0	0
K=2	0,39	0,43	0,50	0,74
K=3	0,53	0,62	0,66	1,70
K=4	0,66	0,70	0,75	2,34

Tabela 5.6: Composição do sistema - Mecanismo *K*

poderão não satisfazer a pré-condição estabelecida. Tal denota-se particularmente com um número *K* elevado, sendo cada vez mais baixa a probabilidade de um representante ter esse número de filhos. Esta diferenciação do resultado é também explicada por este ser uma média dos valores obtidos, influenciando os nós que ainda não têm *K* nós, os resultados.

De qualquer modo, os dados recolhidos estão semelhantes ao esperado, confirmando o funcionamento teórico para este mecanismo.

5.2.2.2 Gastos de largura de banda

Um dos resultados interessante de se avaliar é a evolução do custo variando o *K* do mecanismo. Assim, a figura 5.9 apresenta o *upload* e *download* médio destas duas simulações, comparando com a solução base, com redundância de 0, e a nova solução com 0,8 de redundância, tendo em conta os parâmetros de referência.

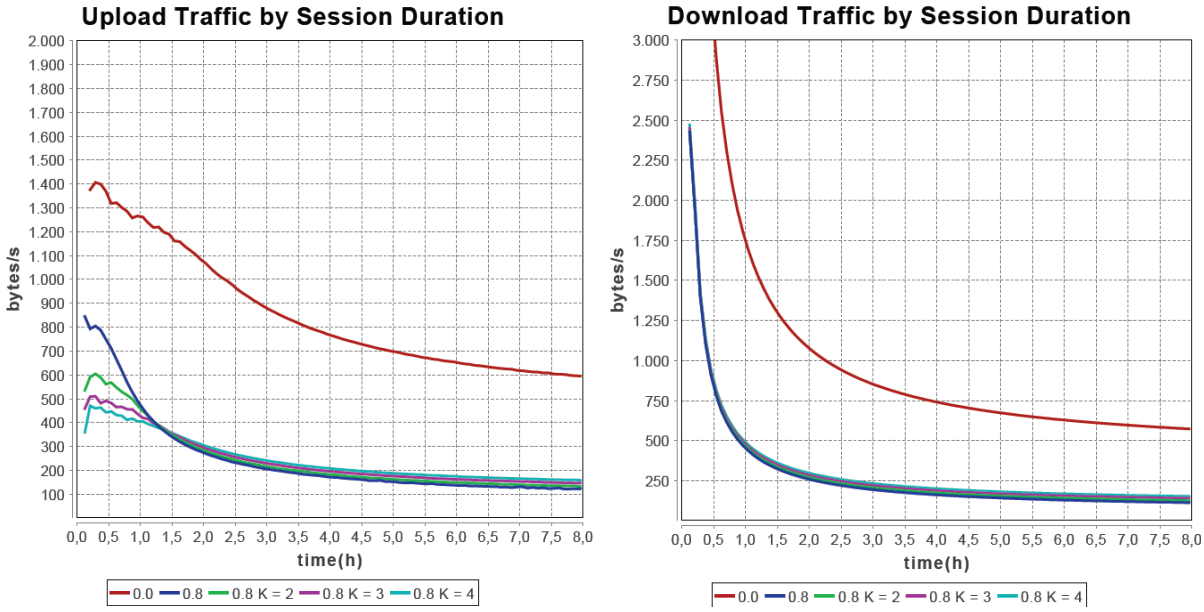


Figura 5.9: Gastos de *upload* e *download* por tempo de sessão, comparando solução base sem redundância com simulações com 0,8 de redundância, com e sem nós K, variando este parâmetro

Ao analisar a figura 5.9, destaca-se a proximidade dos gastos entre as experiências

Simulação	Upload médio	Download médio	Taxa Entrada	Tempo Sessão
Normal	909,9 bytes/s	1048,6 bytes/s	1,4 nós/s	4 horas
K = 1	393,5 bytes/s	458,1 bytes/s	0,6 nós/s	1,96 horas
K = 2	308,5 bytes/s	375 bytes/s	0,57 nós/s	2,62 horas
K = 3	283,3 bytes/s	344,3 bytes/s	0,59 nós/s	2,92 horas
K = 4	274,9 bytes/s	330,4 bytes/s	0,63 nós/s	3,1 horas

Tabela 5.7: Largura de banda utilizada e dados relevantes, variando *K*

com *K* e a solução base com redundância, estando estas simulações bastante abaixo da solução base sem redundância, em termos de gastos em largura de banda. O primeiro facto não é óbvio, pois o número de *super nós* no sistema cresceria, e portanto, a taxa de entrada destes também, levando a um maior custo.

De maneira a avaliar e entender os resultados obtidos nos gráficos, apresentamos os dados dos gastos médios e taxas de entrada de *super nós* correspondentes, na tabela 5.7.

Ao analisar a tabela 5.7, entende-se que o custo médio vai diminuindo, enquanto o *K* aumenta. Parte deste fenómeno pode ser explicado pelas taxas de entradas de *super nós* verificadas, muito mais baixas que o que seria óbvio acontecer, para o número de *super nós* obtido nas simulações *K*.

Contudo, este facto tem uma explicação simples. Como a recolha de métricas só é efectuada após o *warmup* de funcionamento, de maneira a estabilizar o sistema, este guarda os resultados a partir de uma fase em que os *super nós* por filtro já estão estabelecidos, bem como a maior parte dos nós *K*. Assim, a taxa de *super nós* variará por dois motivos: entrada de nós *K* em falta para um filtro e promoções de nós. De notar que os nós *K* são filhos de um *super nó*, não tendo filhos e, portanto, comportando-se como um filho especial, não havendo promoção caso este saia antes de ser representante de um filtro.

Por outro lado, como os nós *K* já são *super nós*, ao substituírem o pai não precisam de entrar no sistema, não sendo contabilizado este caso como uma entrada de *super nó*. Tal fará com que a taxa de entrada de *super nós* desça, sendo compensada por uma possível entrada de um novo nó *K*. Assim, percebe-se que a taxa de entrada de *super nós* não variará muito, para simulações com a mesma redundância e K diferente, explicando os resultados de largura de banda gasta.

Explicada a equivalência das taxas, é importante demonstrar o porquê da diminuição progressiva do custo médio, com o aumento de *K*. Este facto também é facilmente explicado. O tempo médio de sessão para os *super nós* aumenta, visto entrarem logo inicialmente como tal, diminuindo o custo médio por segundo suportado, sendo o custo de largura de banda diluído por um maior tempo de sessão.

Através destas conclusões pode-se inferir a viabilidade do funcionamento das arquitecturas com *K* nós. Porém, existe um custo inicial elevado para a estruturação e estabelecimento da rede, provocado pela entrada de *super nós* representantes e *K*, até ao

Experiência	Tempo de Entrada	% nó K é utilizado	% melhoria
Sem K	59,00	0	0
K=2	51,37	31	13
K=3	45,10	53	24
K=4	41,32	69	30

Tabela 5.8: Tempos de entrada e melhorias variando *K*

limite estabelecido. Nesta fase, a taxa de entrada de *super nós* deverá ser bastante elevada. Segundo estas distribuições do gasto de largura de banda, as simulações com *K* seriam viáveis neste tipo de sistema e até baixariam o custo médio para os *super nós*. Esta conclusão é bastante interessante pois, à partida, este resultado não seria óbvio.

5.2.2.3 Tempos

Como havíamos discutido na análise à fase anterior, uma das melhorias que este mecanismo traria seria a diminuição do tempo de espera para a entrada de um novo nó. Tal deve-se ao facto de, já estando na rede de *super nós*, mas permanecendo inactivo, o nó, mal detecta a falha do pai, entra como representante sem ter que descarregar os *endpoints* e filtros dentro do sistema.

De modo a verificar se esta melhoria está a ser aproveitada por este mecanismo, foi desenvolvido um estudo para o tempo que demora um *super nó* promovido a estar operacional. Esses resultados foram comparados com os tempos já descritos da fase anterior. A tabela 5.8 mostrará as seguintes métricas:o tempo médio para um nó promovido estar activo, em segundos e percentagem de vezes que um nó *K* é promovido.

Estudando a tabela 5.8 conclui-se que ao aumentar o *K*, reduzem-se as vezes que numa promoção o nó tem que entrar ainda na rede de *super nós*. Tal faz com que o tempo médio de correcção da saída do pai desça, como verificado nos resultados. Assim se conclui que a utilização deste mecanismo melhora a tolerância a falhas do sistema, melhorando os tempos de recuperação.

De notar que estes dados são médios do sistema, sendo que a entrada de um nó *K* será bastante mais rápida do que a de um *sub nó*. Mesmo que a taxa de utilização de *K* fosse total, o tempo não baixaria muito devido ao tempo médio de detecção da saída. De maneira a diminuir este tempo, tem que se alterar a periodicidade da verificação do pai, como já foi referido anteriormente.

Assim, termina a análise a este mecanismo de tolerância a falhas que demonstra alguns resultados interessantes, que não seriam óbvio à primeira vista.

5.2.3 Dimensionador do Sistema

Nesta secção do trabalho serão apresentados resultados sobre o funcionamento deste mecanismo do sistema, que permite definir uma quantidade extra de largura de banda que o sistema poderá utilizar, caso não o esteja a fazer.

A avaliação dos resultados desta funcionalidade basear-se-à nos custos de largura de banda, comparativamente com a solução com redundância e sem *budget*. Serão recolhidas métricas de forma a inferir a taxa de promoção desenvolvida por este mecanismo.

Desta maneira foi realizada uma simulação com base nos parâmetros da tabela 5.2, utilizando uma taxa de entrada de 1,4 nós/s. Será utilizada uma redundância de 0,8 e um *budget* para promoções de 120 bytes/s, cerca de 32% do custo médio esperado para a solução base (380 bytes/s). A figura 5.10 demonstra os gastos da simulação base e da utilização deste mecanismo.

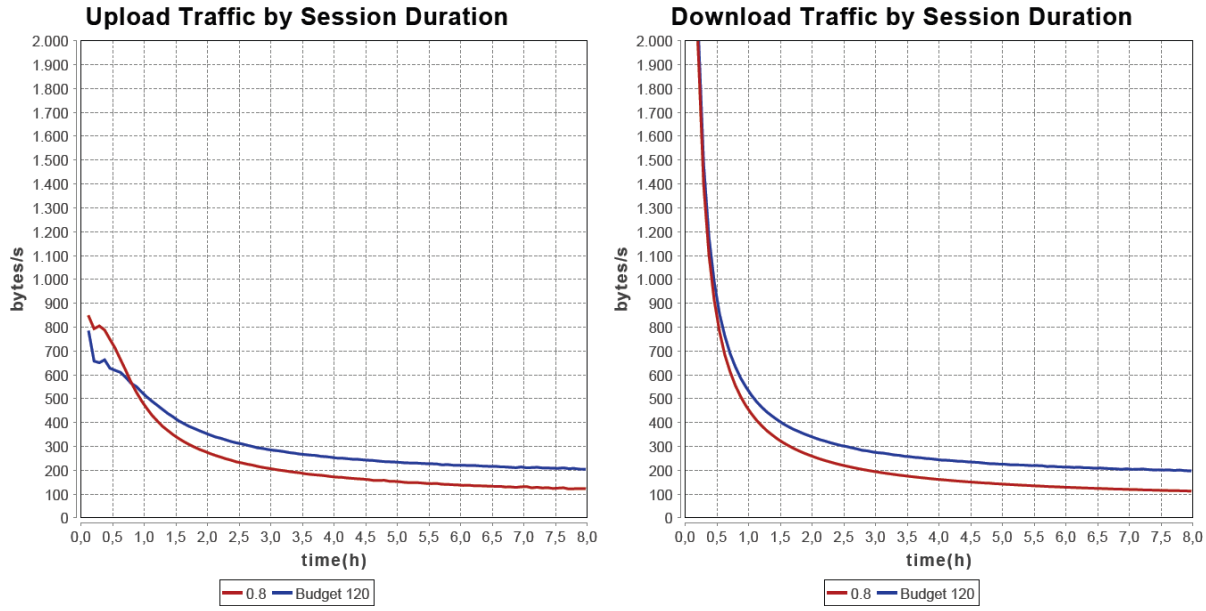


Figura 5.10: Gastos de *upload* e *download* por tempo de sessão para simulações com redundância de filtros de 0,8, uma taxa de entrada de 1,4 nós/s, tendo a experiência azul um *budjet* de 120 bytes/s definido

Ao analisarmos a figura 5.10, verifica-se que, para os variados tempos de sessão, a simulação com *budget* tem um custo maior que a base, tal como esperado. Porém, esse custo não deverá suplantar os 120 bytes/s, definidos para promoções. Analisando o gráfico verifica-se que a curva da simulação com *budjet* está maioritariamente acima da outra. A distância entre as curvas sugere, também, que o excedente de 120 bytes/s não é ultrapassado.

De forma a verificar se o objectivo do mecanismo é alcançado e de maneira a quantificar esse facto, foi desenvolvida uma comparação entre os gastos médios das simulações por tempo de sessão. Nesta análise foi feita uma média entre a diferença de gasto para cada tempo de sessão. Esses resultados estão apresentados na tabela 5.9.

Analisando os resultados presentes na tabela 5.9, percebe-se que, para cada tempo de sessão, em média, o mecanismo com *budget* gasta mais 80 bytes/s do que a solução de referência. Este valor está dentro do excedente de largura de banda definido, sendo assim cumprido o propósito de funcionamento do mecanismo.

Simulação	Upload	Download
Budget - Base	80,2 bytes/s	83,1 bytes/s
% Budjet utilizado	67	69

Tabela 5.9: Largura de banda excedente utilizada pelo dimensionador

Taxa	Sessão
0,29 nós/s	2,76 horas

Tabela 5.10: Taxa de entrada de nós *budget* e tempo de sessão médio para *super nós*

Porém, este valor é mais baixo do que o excedente estabelecido, mostrando que o mecanismo não aproveita a gama de largura de banda oferecida na totalidade. Avaliando quantitativamente a sua utilização, chega-se à conclusão que o mecanismo utiliza cerca de 67% do mecanismo em questão de *upload* e cerca de 69% em termos de *download*.

A não utilização de todo o *budjet* já havia sido prevista na fase de desenho do mecanismo. Como se referiu, a abordagem para este dimensionador é conservadora, de modo a nunca ultrapassar o excedente configurado. Para isso são descartados *slots* em várias situações (nova taxa ou duplicação do tamanho do sistema), explicando os resultados.

Para se aproximar o funcionamento deste mecanismo dos 100%, podem ser feitos estudos que, de acordo com um rácio, introduzem *slots* que não são calculados, de acordo com as percentagens normalmente obtidas e a sua diferença dos 100%. Outro método seria estudar cautelosamente o mecanismo, de forma a verificar quais *slots* podem não ser eliminados, nunca ultrapassando o limite de *upload* imposto.

De maneira a analisar de que modo este mecanismo afecta a taxa de entrada de nós no sistema, foi desenvolvida a tabela 5.10, que expõe a taxa de entrada de nós *budget* e o tempo médio de estadia no sistema para este mecanismo.

Estudando a tabela 5.10, percebe-se que o aumento do custo está intimamente ligado com a taxa de entrada de nós *budget*, que incrementa a taxa de entrada de *super nós*. Verifica-se, também, que o tempo de sessão médio para um *super nó* com este mecanismo, é significativamente maior do que sem ele. Este resultado explica-se facilmente pois, sendo escolhidos *super nós* ao acaso, estes poderão ter um tempo de vida maior do que se fossem promovidos algum tempo depois. Este valor também é explicado pela diminuição de nós filhos, diminuindo também o tempo de espera para se promover.

O tempo de sessão alto poderá explicar a anomalia verificada para sessões baixas. Ou seja, como o tempo médio aumenta, haverá menos nós com sessões baixas, sendo que a média feita não será a mais correcta para ser avaliada.

Em suma, este mecanismo utiliza cerca de 80 bytes/s dos 120 bytes/s para redimensionar a rede. Este aumento é traduzido por um aumento da taxa de entrada de *super nós*, sendo que a taxa de entrada de nós *budget* é de 0,29 nós/s. A percentagem do excedente utilizado é em média de 67%, conseguindo alcançar assim os propósitos deste

Parâmetros	
Tempo médio de sessão	4 horas
Tempo máximo de sessão	8 horas
Endpoint	6 bytes
Filtro	506 bytes
Tempo de Simulação	23 horas
Tamanho do evento	16 bytes
Taxa de entrada	0,7 nós/s
Taxa de eventos	0,5 min
Taxa de redundância de filtros	0,8
Dimensão do histórico	10 eventos

Tabela 5.11: Parâmetros de simulação - Disseminação Filtrada

mecanismo. Porém, seria desejado que esta percentagem estivesse mais perto dos 100% de excedente utilizado.

5.2.4 Disseminação Filtrada

5.2.4.1 Reparação

Expostos os resultados relativos aos mecanismos de filiação do novo sistema, serão agora descritas as experiências realizadas ao nível da disseminação filtrada de conteúdo, dentro da rede construída. O trabalho desenvolvido nesta fase foi ao nível da recuperação de falhas do sistema implementado para a disseminação do conteúdo. Foi também desenvolvido um método de entrega de eventos aos *sub nós*, com resultados demonstrados.

Com esta análise pretende-se medir a eficácia do método de reparação desenvolvido. Para efectuar este estudo, foi simulada uma experiência base, sem reparação na parte da disseminação filtrada. Como base desta experiência, estará todo o método de filiação desenvolvido anteriormente. A tabela 5.11 mostra os parâmetros utilizados nesta experiência.

Nesta experiência aparecem três parâmetros novos ao sistema. O tamanho do evento é a estipulação de quantos bytes gastará cada evento no sistema. A taxa de eventos representa o número de eventos de disseminação filtrada que são injectados no sistema, por minuto. Utilizou-se uma taxa baixa por motivos de simplificação e facilidade de teste. A dimensão do histórico partilhado é um parâmetro que expõe o número de eventos que poderão ser recuperados para trás. Foi utilizado um tamanho de 10 eventos, significando, em média 5 minutos de chegadas de eventos. Tal é muito cauteloso, mas como o volume de eventos é muito baixo, optou-se pela simplicidade.

Realizada esta experiência base, obtiveram-se os seguintes resultados, em termos de recepção de eventos e tempo médio para tal. Este estudo será evidenciado na tabela 5.12.

Ao analisar os resultados da simulação sem reparação, verificamos que a taxa de entrega é alta, mas não corresponde à não existência de falsos negativos. Tal deve-se à falha de algum nó ou *sub nó* no interior da árvore de envio do evento, privando os restantes

Reparação	% Eventos sucesso	% *Super nós*	% *Sub nós*	Tempo
Não	87	86,97	86,96	1,7 segundos
Sim	99,96	99,98	99,98	3,6 segundos

Tabela 5.12: Resultados da reparação

de o conhecerem. De notar que as taxas de entrega para *super nós* e *sub nós* são idênticas, mostrando que o mecanismo de envio dos eventos para os *sub nós* funciona. O tempo médio até um nó conhecer o evento é baixo, observando-se o bom funcionamento deste, mesmo para uma rede com grandes dimensões.

Quando a reparação é efectuada, a taxa de eventos entregues com sucesso é máxima, demonstrando que o mecanismo de reparação está a funcionar e consegue colmatar as falhas descritas. Assim, este mecanismo introduz um alto nível de tolerância a falhas, evitando ao máximo os falsos negativos, prejudiciais para o funcionamento deste aplicação.

Porém, estes não podem ser evitados pois, caso o nó que publica o evento falhe logo antes de o enviar, este nunca chegará a ninguém nem poderá ser recuperado. Estes casos são ignorados pelas estatísticas desenvolvidas em ambas as experiências devido à impossibilidade de reparação.

Foi, também, interessante de verificar os tempos médios até um nó conhecer um evento, tendo em conta a reparação efectuada. O tempo médio até um nó conhecer um evento foi de 3,6 segundos, aumentando (112%) em relação à simulação base. Tal é justificável pelo tempo que os nós que não receberam o evento, demoram a recuperar essa falha.

Este mecanismo só não atingiu os 100% por causa de um lapso de teste. O problema foi que nos testes realizados, os eventos não foram parados, estando ainda nós a recebê-los e repará-los. Assim, os mecanismo ainda estavam em curso, impossibilitando o reconhecimento total da taxa. A injecção de eventos deveria ter sido parada algum tempo antes do fim da simulação, de modo a obter os resultados totais, e não parciais. Por falta de tempo não foi possível repetir a experiência.

5.2.4.2 Custos e Histórico

A largura de banda gasta para efectuar o envio de eventos para os filhos é muito baixa por evento. Assim, o custo desta parte da solução dependerá quase exclusivamente do número de eventos recebidos por nó. De qualquer forma, por evento, são gastos poucos bytes quer para *super nós*, quer para *sub nós*, sendo praticamente desprezável esta medida para taxa de entrada de eventos baixas.

Outro custo que seria importante analisar era o custo da reparação efectuada, em termos da percentagem de reparação obtida. Ao analisar este mecanismo, entende-se que esta reparação estará dependente do tamanho do histórico trocado, bem como do número de eventos disponibilizados.

Assim, o tamanho do histórico traz uma discussão interessante. Caso se utilizasse um histórico muito grande, a reparação seria sempre total, comprometendo, porém, o custo deste mesmo mecanismo. Assim, deveria ser efectuado um estudo cuidado de forma a medir custos e percentagens de reparação, de forma a optimizar o funcionamento do algoritmo, não tornando a reparação excessivamente cara.

O tamanho do histórico deverá estar intimamente ligado ao tempo necessário até que todos os eventos perdidos sejam recuperados e a taxa de injecção de eventos. Tal deve-se ao facto de que, demorando x segundos a conhecer um evento, não é necessário guardá-lo no histórico partilhado por muito mais tempo. Um estudo mais aprofundado destas duas métricas poderia obter resultados em termos da definição da truncagem do histórico enviado.

5.3 Conclusões

Terminada a avaliação experimental para a nova arquitectura e mecanismos adicionais desenvolvidos, serão referidas as principais conclusões a tirar destes resultados.

Para o sistema base com a nova arquitectura, observa-se que a utilização da redundância dos filtros como método para diminuir o *churn* e, por conseguinte, a largura de banda, resulta. Assim, este novo sistema obtém gastos de largura de banda mais baixos que o anterior, sendo esta melhoria maior com o aumento da redundância de filtros. Este facto comprovou-se pela taxa de entrada de *super nós* verificada nas simulações que foram estudadas.

Uma conclusão interessante que se pode tirar foi que a relação entre os gastos médios do sistema e a taxa de entrada de *super nós* é idêntica, sugerindo que estes gastos estão efectivamente dependentes desta taxa. Ao efectuar a relação entre a taxa de entrada de nós e taxa de entrada de *super nós*, foi obtida uma relação idêntica para os vários *churn* testados. Tal sugere que o comportamento deste sistema é igual para cada taxa de redundância, não dependendo de outros factores de forma pronunciada.

Ao analisar a composição do sistema, percebeu-se que o número de filhos de um *super nó* apenas varia de acordo com a redundância do sistema, não sendo afectado por nenhuma taxa de entrada. Entendeu-se também, que a probabilidade de um nó ser *super nó* durante o seu tempo de sessão é mais alta do que a taxa de *super nós* esperada, devido ao mecanismo de promoções e tempo de sessão médio de um *super nó*. Verificou-se que este tempo de sessão médio de um *super nó* descia bastante para as simulações com a nova arquitectura, baixando também com o aumento da redundância de filtros.

Finalizando o sistema base, concluiu-se que os *sub nós* não efectuam quase trabalho nenhum no sistema, sendo os seus gastos desprezáveis e constituídos na sua maioria pelo mecanismo de promoção, cujo custo apenas depende do número de filhos por *super nó*. Assim, entende-se que deveria ser desenvolvido um mecanismo de delegação de trabalho, equilibrando a carga no sistema, tirando algum trabalho em excesso aos *super nós*.

Quanto à solução de *K* nós, verificou-se que o custo se mantém, para qualquer valor de *K*. Tal também acontece com a taxa de entrada de nós, sendo justificado pela não análise dos resultados enquanto o sistema estabiliza. Após estabilizar, o funcionamento será em tudo idêntico ao da solução de referência com a mesma redundância de filtros. Neste mecanismo observou-se uma melhoria nos tempos de entrada, após a detecção de uma falha.

Foi comprovado o funcionamento do dimensionador do sistema. Este aumenta o sistema de maneira equilibrada, mantendo os gastos adicionais, dentro do excedente que foi designado. Porém, apenas 67% deste excedente é utilizado, resultando numa taxa de entrada de nós *budget* proporcional ao aumento do custo.

Para terminar, quanto à disseminação filtrada, foram medidas quantitativamente as melhoras proporcionadas pelo mecanismo de tolerância a falhas desenvolvido. A taxa de recuperação de falhas obtida foi total, recuperando os cerca de 13% de eventos perdidos(eliminando os falsos negativos), excepto nos casos extremos de não publicação do evento. A taxa só não chegou efectivamente a esse valor devido ao lapso referido na experiência. Ficou por fazer um estudo relacionando a taxa de reparação, o tamanho do histórico trocado e os custos desta parte da solução.

6

Conclusão e Trabalho Futuro

6.1 Conclusões

Os sistemas Editor/Assinante baseados no conteúdo permitem que cada utilizador defina que conteúdos deseja receber, sendo estes encaminhados para o mesmo, aquando da sua disponibilidade. Este método simplifica a vida do utilizador, não havendo recepção de conteúdo indesejado, obtendo sempre a informação que procurava, ao contrário de modelos sem filtragem que apresentam informação indesejada ao utilizador.

Outra característica dos sistemas baseados neste paradigma é a notificação dos utilizadores, aquando da actualização de informação que estes definiram como interessante. Assim, ao ser publicada uma actualização de um conteúdo desejado pelo utilizador, os dados são-lhe encaminhados, recebendo a informação logo que possível. O método de disseminação de informação destes sistemas é simples e eficaz, poupando trabalho ao utilizador, característica fulcral dos sistemas de hoje em dia.

O sistema base de partilha de conteúdo que foi desenvolvido neste trabalho, funciona segundo o paradigma atrás definido. Este efectua a disseminação filtrada de conteúdo, sobre uma rede com visibilidade completa dos nós. Ou seja, os nós conhecem-se todos entre si e aos seus filtros, sendo o encaminhamento de informação imediato e facilitado.

Contudo, este sistema tem uma limitação ao nível de escalabilidade. Assim, com o aumento da taxa de entrada de nós no sistema, o custo deste aumenta, de forma a tornar sistemas muito dinâmicos caros demais para implementar a característica de visibilidade completa. Pretendia-se o desenvolvimento de uma arquitectura que mitigasse este problema, gastando, por conseguinte, menos largura de banda, suportando todo o tipo de dinamismo que se poderia observar na rede.

Desta maneira, foi criado um algoritmo de filiação novo para este sistema, baseado

em *super nós*. Estes *super nós* exploram uma característica bastante frequente em redes *peer-to-peer*, a redundância de filtros no sistema. Ou seja, é esperado que num sistema de grandes dimensões, existam vários participantes que pretendam o mesmo conteúdo, tendo estes filtros iguais. Assim, a nova arquitectura baseava-se na existência de apenas um *super nó* por filtro diferenciado no sistema, sendo a visibilidade completa da rede exclusivamente formada por *super nós*.

Desta forma, os *super nós* conhecem-se todos entre si, sendo, também, conhecidos todos os filtros presentes no sistema. Com isto, todos os outros participantes que entrem no sistema com um filtro que já estivesse presente neste, ligam-se ao *super nó* em questão com um filtro igual. Esse representante conhece assim, todos os seus filhos e estes, têm informação sobre o representante e os seus irmãos. Esta estrutura é mantida mesmo após a saída do *super nó*, de modo a conservar o sistema funcional. O mecanismo de promoções permite superar a saída do pai, passando um dos seus filhos para a sua posição.

A filiação na nova arquitectura é tratada segundo este método, proporcionando uma base estrutural para a disseminação de conteúdo. A disseminação filtrada necessitava de alterações, pois não contemplava o envio de notificações aos filhos. Desenvolveu-se essa abordagem, permitindo torna-la eficaz, fazendo chegar cada evento a todos os participantes que o desejam.

Este foi o sistema base desenhado e implementado. Complementarmente foram também desenvolvidos mecanismos adicionais, incidindo sobretudo na introdução de tolerância a falhas por parte do novo sistema. Deste modo, para a filiação, foi pensado e implementado um mecanismo que introduz no sistema no máximo *K super nós* do mesmo filtro.

Através destes nós adicionais podia ser obtida redundância para os representantes de cada filtro, relaxando a condição da existência de apenas um. Porém, estes *K super nós* têm mais uma tarefa dentro do sistema. São *super nós*, mas estão ligados a um representante do filtro. Assim, estes nós *K* são *super nós* inactivos, não tendo qualquer filho ligado a eles e delegando o trabalho de representante no *super nó* a quem estão ligados. No fundo, estes *super nós* criam um terceiro tipo de participantes no sistema, os *super nós* que também são filhos. Estes trazem vantagens em termos da promoção imediata após a detecção da saída do representante, e outros métodos de tolerância a falhas.

Foi também introduzido um mecanismo de tolerância e recuperação de falhas na disseminação filtrada de conteúdo. Esta, sem alterações, proporcionava uma entrega muito satisfatória de conteúdo aos nós que os desejam receber. Porém, não garantia a inexistência de falsos negativos, perdendo alguns eventos que deveriam ser entregues, devido à saída de nós que faziam parte da árvore de disseminação do evento.

Para colmatar estas falhas foi desenvolvido um mecanismo epidémico de recuperação de eventos, em que os nós com filtros semelhantes trocam os seus históricos de eventos recebidos. Tal é feito de forma a avaliar as possíveis notificações em falta, recebendo-as, caso seja necessário. Esta recuperação de eventos é também efectuada nos *sub nós*, podendo ocorrer o mesmo tipo de falhas na disseminação do evento para o *sub nó*.

O último mecanismo desenvolvido foi um dimensionador do sistema. Este faz crescer evolutivamente a rede, caso esta tenha *upload* excedente suficiente para suportar esse crescimento. Assim, é definido um excedente de largura de banda, que o dimensionador poderia utilizar para fazer crescer a rede, sempre dentro dos gastos definidos. Este mecanismo relaxa um pouco a definição da existência apenas de um *super nó* por filtro, podendo ser utilizado por outro tipo de arquitecturas baseadas em *super nós*.

Os resultados experimentais mostraram que a utilização da nova arquitectura é bastante proveitosa, gastando menos largura de banda comparativamente à arquitectura original. Observa-se que o custo baixa com o aumento da redundância de filtros presente na rede. Tal deve-se à diminuição da taxa de entrada de *super nós* nestes sistemas em relação a esta taxa de entrada de nós para o sistema base.

Após a análise dos resultados experimentais, observou-se, também, que existe uma forte ligação entre a taxa de entrada de *super nós* e o custo médio de largura de banda gasta, sendo as inclinações destas curvas iguais. Um facto interessante é o de que, independente da taxa de entrada de nós na rede, para cada redundância de filtros, a relação entre taxa de entrada de *super nós* e taxa de entrada de participantes mantém-se igual. Tal sugere que o funcionamento do novo sistema não depende das taxas de entrada, tendo sempre o mesmo comportamento para cada redundância. Com isto, concluiu-se que a solução atingiu os seus objectivos, ultrapassando limitação de escalabilidade provocada pelo *churn*. Desta maneira, o novo sistema tende a suportar redes com um dinamismo mais elevado e grande número de nós, sem ter problemas de escalabilidade ao nível da largura de banda gasta.

Os resultados observados para o mecanismo de K nós, sugerem que este melhora o tempo de entrada após uma falha. Foi demonstrado também que, ao contrário do esperado, variando o K o custo do sistema mantém-se na mesma ordem. Concluiu-se que tal era explicado pela manutenção da taxa de entrada e nós, visto os nós K já serem *super nós*. Desta maneira, sugere-se que este mecanismo será bastante viável tendo um custo baixo, faltando um simulador de falhas de modo a entender a real contribuição do mesmo. Porém, o período de estabelecimento inicial da arquitectura será mais caro com o aumentar de K, compensado pela sua *performance* após este período.

Analisando os resultados experimentais sobre o funcionamento do dimensionador do sistema, foi observado que este é eficaz. Verificou-se que o sistema é aumentado equilibradamente, dentro dos limites de largura de banda definidos. A utilização do excedente de *upload* definido por este mecanismo foi cerca de 67%, cumprindo o propósito para que foi desenhado.

Quanto à disseminação filtrada, os resultados analisados permitiram concluir que a recuperação de eventos revelou-se bem sucedida, conseguindo com elevada probabilidade a notificação de todos os nós que não conseguiram receber o evento a dada altura. Foi também concluído que o mecanismo de envio de eventos aos filhos é funcional, sendo o seu custo muito baixo para cada evento, e portanto, quase desprezável para taxas baixas de recepção de eventos.

Desta forma se conclui o trabalho realizado, sendo evidenciadas seguidamente as suas principais contribuições.

6.2 Contribuições Principais

As principais contribuições deste trabalho, descritas sumariamente, são as seguintes:

- Desenho e implementação de uma nova arquitectura para disseminação filtrada de conteúdo baseada em *super nós*, explorando os filtros redundantes no sistema, mitigando a limitação inerente ao *churn*;
- Mecanismos de tolerância a falhas para a filiação, com a utilização de K nós do mesmo filtro, e para a disseminação filtrada, recuperando eventos não recebidos;
- Um dimensionador do sistema que permite modelar o tamanho da rede de acordo com um excedente de *upload* definido;
- Uma avaliação experimental de resultados dos sistemas anteriormente descritos, tirando conclusões sobre o seu funcionamento.

6.3 Trabalho Futuro

Indicam-se, seguidamente, algumas das direcções que poderão futuramente ser seguidas com vista à continuação e melhoria da solução de difusão filtrada proposta nesta dissertação.

- Efectuar reparações e estabelecer uma ordem de chegada dos eventos para todos os nós do mesmo filtro. Este melhoramento seria desenvolvido com a introdução de sequenciadores na arquitectura corrente para a disseminação filtrada;
- Obter uma melhor distribuição da carga do sistema e equilíbrio da mesma com a delegação de trabalho nos filhos, de maneira a equiparar os gastos entre *super nós* e *sub nós*;
- Redimensionamento do sistema estabelecendo um limite máximo de *upload* gasto por nó. Ultrapassado esse valor, o número de *super nós* deveria ser diminuído pelo sistema, utilizando métodos de fusão de filtros, caso necessário;
- Estudar e investigar o papel do histórico na recuperação de eventos na disseminação filtrada.

Bibliografia

[Aek06] Ioannis Aekaterinidis. PastryStrings: Contentbased publish/subscribe over dht networks. *Computer Engineering*, pág. 10–12, 2006.

[BCMR04] J.W. Byers, J. Considine, M. Mitzenmacher, e S. Rost. Informed Content Delivery Across Adaptive Overlay Networks. *IEEE/ACM Transactions on Networking*, 12(5):767–780, Outubro 2004.

[BMVV05] R. Baldoni, C. Marchetti, A. Virgillito, e R. Vitenberg. Content-based publish-subscribe over structured overlay networks. In *Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on*, pág. 437–446. IEEE, 2005.

[CCR05] Miguel Castro, M. Costa, e A. Rowstron. Debunking some myths about structured and unstructured overlays. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pág. 85–98. USENIX Association, 2005.

[CDK05] G.F. Coulouris, J. Dollimore, e T. Kindberg. *Distributed systems: concepts and design*. Addison-Wesley Longman, 2005.

[CLB+06] Dave Clark, Bill Lehr, Steve Bauer, Peyman Faratin, Rahul Sami, e John Wroclawski. Overlay Networks and the Future of the Internet. *Communications and Strategies*, 63(63):109, 2006.

[CRW04] a. Carzaniga, M.J. Rutherford, e A.L. Wolf. A routing scheme for content-based networking. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 2, pág. 918–928. IEEE, 2004.

[CSWH01] Ian Clarke, Oskar Sandberg, Brandon Wiley, e T. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Designing Privacy Enhancing Technologies*, pág. 46–66. Springer, 2001.

[EFGK03] Patrick Th. Eugster, Pascal a. Felber, Rachid Guerraoui, e Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, Junho 2003.

[GBL+03] Indranil Gupta, Ken Birman, Prakash Linga, Al Demers, e R. Renesse. Kelips: Building an efficient and stable P2P DHT through increased memory and background overhead. *Peer-to-Peer Systems II*, (1):160–169, 2003.

[GH06] Patrick Goering e G. Heijenk. Service discovery using Bloom filters. In *Proc. Twelfth Annual Conference of the Advanced School for Computing and Imaging, Belgium*. Citeseer, 2006.

[GLR03] A. Gupta, Barbara Liskov, e Rodrigo Rodrigues. One hop lookups for peer-to-peer overlays. In *9th Workshop on Hot Topics in Operating Systems (HotOS)*, volume 18, pág. 21. Citeseer, 2003.

[GLR04] Anjali Gupta, Barbara Liskov, e Rodrigo Rodrigues. Efficient routing for peer-to-peer overlays. In *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation-Volume 1*, pág. 9. USENIX Association, 2004.

[KK03] M. Kaashoek e D. Karger. Koorde: A simple degree-optimal distributed hash table. *Peer-to-Peer Systems II*, (1):98–107, 2003.

[LCC+02] Qin Lv, Pei Cao, Edith Cohen, Kai Li, e Scott Shenker. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th international conference on Supercomputing*, pág. 84–95, New York, New York, USA, 2002. ACM.

[MD10] J. Martins e Sergio Duarte. Routing algorithms for content-based publish/-subscribe systems. *IEEE Communications Surveys & Tutorials*, 12(1):39–58, 2010.

[MFP06] G. Muhl, L. Fiege, e P. Pietzuch. *Distributed Event-Based Systems*. Springer-Verlag, 2006.

[MMDM09] Simão Mata, J.L. Martins, Sérgio Duarte, e M. Mamede. Análise do custo e da viabilidade de um sistema P2P com visibilidade completa. *INForum Simpósio de Informática*, pág. 333–344, 2009.

[MNR02] Dahlia Malkhi, M. Naor, e David Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pág. 183–192. ACM, 2002.

[PB02] P.R. Pietzuch e J.M. Bacon. Hermes: a distributed event-based middleware architecture. *Proceedings 22nd International Conference on Distributed Computing Systems Workshops*, pág. 611–618, 2002.

[RB04] Rodrigo Rodrigues e Charles Blake. When multi-hop peer-to-peer routing matters. In *Proceedings of the 3rd International Workshop on Peerto-Peer Systems (IPTPS'04)*. Citeseer, 2004.

[RD01] Antony Rowstron e Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. *Design*, (November 2001), 2001.

[RKCD01] Antony Rowstron, A.M. Kermarrec, Miguel Castro, e Peter Druschel. SCRIBE: The design of a large-scale event notification infrastructure. *Networked Group Communication*, pág. 30–43, 2001.

[SMLN+03] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, e H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, Fevereiro 2003.

[SR06] Daniel Stutzbach e Reza Rejaie. Understanding churn in peer-to-peer networks. *Proceedings of the 6th ACM SIGCOMM on Internet measurement - IMC '06*, pág. 189, 2006.

[Tar10] Sasu Tarkoma. *Overlay Networks - Toward Information Networking*. Auerbach Publications - Taylor & Francis Group, 2010.

[Wik09] Wikipedia. Internet Traffic, 2009.

[Wik10] Wikibooks. Building a P2P System, 2010.