Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Dissertação de Mestrado

Mestrado em Engenharia Informática

Desenho e Avaliação do Algoritmo de Filiação do Projecto LiveFeeds

Simão Mendes da Mata (aluno nº 26337)

2º Semestre de 2008/09

29 de Julho de 2009

Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Dissertação de Mestrado

Desenho e Avaliação do Algoritmo de Filiação do Projecto LiveFeeds

Simão Mendes da Mata (aluno nº 26337)

Orientador: Prof. Doutor José Legatheaux Martins

Trabalho apresentado no âmbito do Mestrado em Engenharia Informática, como requisito parcial para obtenção do grau de Mestre em Engenharia Informática.

2º Semestre de 2008/09

29 de Julho de 2009

Agradecimentos

Gostaria de agradecer a todos os que me ajudaram durante o meu último ano na FCT-UNL.

À Fundação da Faculdade de Ciências e Tecnologia pela atribuição de uma bolsa de investigação que contribuiu para a realização deste trabalho.

Ao Professor José Legatheaux Martins, meu orientador, pelo inestimável apoio e disponibilidade e por sempre me ter incentivado.

Ao Professor Sérgio Duarte por todo o aconselhamento dado e pelo apoio sempre incansável.

A todos os meus colegas e amigos dentro e fora da FCT-UNL por me motivarem e ajudarem sempre que precisei.

Ao Ilídio e à Cila por me ajudarem sempre em tudo o que puderam.

À Raquel, por estar ao meu lado todos os dias.

À minha Mãe por sempre me ter motivado e acreditado em mim e por me ter dado a possibilidade de prosseguir os meus estudos até ao fim.

Ao meu Pai.

A todos, muito obrigado.

Resumo

O tema desta dissertação centra-se no projecto LiveFeeds, mais concretamente no seu algoritmo de filiação, que gere as entradas e saídas dos utilizadores do sistema.

O LiveFeeds é um sistema editor/subscritor que pretende resolver o problema do crescente excesso de carga colocado nos servidores da *World Wide Web* pelos *feeds RSS*, problema este inerente ao modelo actual da WWW em que os utilizadores são forçados a inquirir um dado servidor para descobrir se existem alterações aos conteúdos em que estão interessados.

O LiveFeeds está estruturado seguindo uma abordagem *peer-to-peer*, tendo sido escolhida como forma de encaminhamento um algoritmo de encaminhamento com um número constante de passos. Uma aproximação deste tipo leva a que seja necessário reflectir aprofundadamente no algoritmo de filiação utilizado. Para resolver o problema deste tipo de encaminhamento é necessário que haja uma replicação total ou parcial da informação de filiação dos utilizadores em todos os utilizadores do sistema. Esta problemática da replicação da informação está assim inseparavelmente ligada ao algoritmo de filiação do sistema.

Dentro deste âmbito, esta dissertação proporciona diferentes contribuições. É desenvolvido um modelo que reflecte correctamente a dinâmica de entradas e saídas dos possíveis utilizadores do LiveFeeds que é utilizado para testar o algoritmo em condições próximas da sua execução real, em ambiente de simulação. Através de métodos analíticos e de simulação, é investigado se os custos de manutenção da replicação da filiação são suportáveis para que tal aproximação tenha viabilidade. É ainda sugerida uma alternativa ao algoritmo de filiação com visibilidade completa recorrendo a uma hierarquização do sistema baseada na noção de super-nós. Tal alternativa é estudada em profundidade sendo analisados os pontos centrais do desempenho do algoritmo, incluindo a análise de vários critérios que afectam o comportamento do mesmo. Do mesmo modo, são sugeridos alguns melhoramentos que poderão ser implementados no futuro.

Palavras-chave: editor/subscritor, encaminhamento, par-a-par, *feed*, *RSS*, super-nós

Abstract

LiveFeeds is a publish/subscribe system that aims to solve the problem of the increasing load that web servers must support in order to be capable of providing RSS feeds. This problem is inherent to the actual model of the World Wide Web in which users must inquire the web server every time they want to know if there are any changes to the contents they are interested in.

LiveFeeds is structured as a peer-to-peer system, using a routing algorithm with a constant number of hops. Such approach leads to a reflection about the membership algorithm. In order to solve the problem of routing in a constant number of hops, membership information needs to be replicated in all nodes of a system or in a subset of all nodes. Therefore, the problem of membership replication is directly related to the membership algorithm.

In this context, this thesis provides several contributions. A model that accurately reflects the dynamic of joins and departures of the possible users of the LiveFeeds system is used to test the membership algorithm using simulation under conditions similar to a real implementation. Using simulation and analytic methods, we study the magnitude of the costs of maintaining the membership replication and determine in which cases the use of this type of approximation is feasible. Furthermore, an alternative to the membership algorithm with full visibility is suggested in which a hierarchy is introduced in the system using super-nodes. The various criteria and parameters that affect the behaviour of this alternative are also studied. Finally, some improvements that can be implemented in the future are suggested.

Keywords: publish/subscribe, churn, routing, peer-to-peer, feed, RSS, super-peer

Conteúdo

1 Introdução **1**

1.1 O Projecto LiveFeeds e o seu algoritmo de filiação 1

1.2 Descrição e contexto 2

1.3 Ambiente de simulação 4

1.4 Principais contribuições 7

1.5 Organização do documento 8

2 O Algoritmo de filiação **11**

2.1 Apresentação do algoritmo de visibilidade completa 11

2.2 Particularidades do algoritmo de visibilidade completa apresentado 14

3 Trabalho relacionado **17**

3.1 Encaminhamento em sistemas P2P editor/subscritor 17

3.2 Encaminhamento utilizando um número constante de passos 19

3.3 Modelos de Churn 21

Como obter um modelo de *churn* 22

3.4 Métodos de Avaliação 24

3.5 Conclusão 28

4 Análise do custo da difusão usando árvores aleatórias **29**

4.1 Parâmetros da análise 29

4.2 Probabilidade de um nó ser folha 30

4.3 Tráfego por nó 31

4.4 Estimativa do valor de *r* 33

4.5 Exemplo dos custos 33

4.5.1 Cálculo inverso 34

4.6 Simulações sem *churn* e sem falhas 35

4.7 Conclusões 36

5 Custo da difusão da filiação num sistema dinâmico **39**

5.1 O Modelo de churn utilizado 39

5.2 Parâmetros e métricas da simulação 41

Métricas utilizadas e sua representação gráfica 42

Parâmetros utilizados 43

5.3 Resultados 44

5.4 Conclusões 48

6 Sistema hierárquico - Super-nós **51**
6.1 Trabalho Relacionado 51
6.2 O algoritmo de Super-nós 55
6.3 Critérios utilizados nas simulações com super-nós 59
6.4 Critérios de selecção de nós para promoção 61
6.4.1 Promoção do filho com maior tempo de vida futura 61
6.4.1.1 Informação perfeita sobre nós semente com promoção do filho com maior tempo de vida futura 61
6.4.1.2 Escolha aleatória de nós semente com promoção do filho com maior tempo de vida futura 64
6.4.2 Promoção do filho com menor tempo de vida futura 64
6.4.2.1 Informação perfeita sobre nós semente com promoção do filho com menor tempo de vida futura 65
6.4.2.2 Escolha aleatória de nós semente com promoção do filho com menor tempo de vida futura 67
6.4.3 Promoção do filho mais antigo 68
6.4.4 Promoção aleatória de um filho 69
6.4.5 Conclusões sobre os critérios de promoção de filhos 69
6.5 Delegação de trabalho para os filhos 72
6.6 Simulações com delegação de trabalho para os nós filhos 73
6.7 Resultados obtidos com delegação de trabalho para os filhos 75
6.8 Outros melhoramentos 79
6.8.1 Outros critérios para promoção dos filhos 80
6.8.2 Adaptação ao ritmo de entrada 80
6.8.3 Utilização de múltiplos pais 81
6.8.4 Comunicação entre filhos, compatibilidade de filtros e tempo antes de promoção 81

7 Conclusões e trabalho futuro **83**

A Setting up Modelnet: A large-scale network emulator **87**
A.1 Introduction 87
A.2 ModelNet Architecture 88
A.3 Setting up ModelNet under Debian Linux 88
A.3.1 Core nodes 89
A.3.2 Edge Nodes 91
A.3.3 Installing gexec and authd 91
A.4 Setting up a testbed environment using virtualization with VirtualBox 92
A.4.1 Configuring VirtualBox 92

A.5 Deploying topologies 94
A.5.1 A simple topology 96
A.5.2 Using Inet to generate a topology 98
A.6 Conclusion 99

Lista de Figuras

1.1 Aspecto da janela da interface gráfica do simulador referente à estrutura do sistema ao utilizar o algoritmo de visibilidade completa 5
1.2 Aspecto da janela da interface gráfica do simulador referente à estrutura do sistema ao utilizar o algoritmo de super-nós 6
1.3 Gráficos obtidos com o simulador do LiveFeeds para uma das simulações realizadas 7

2.1 Diagrama das comunicações provocadas pela entrada de um novo nó (o nó *a*) num sistema com 19 nós em que existe um único *slice leader* (o nó *sl*) e parametrizado para dividir os identificadores em $G = 2$ fatias 13

3.1 *Trade-off* entre exactidão e complexidade do método de avaliação 26

4.1 Valor de $P(interior)$ em função do grau G utilizado 31
4.2 Gráfico da expressão (4.20) para vários valores de m 34
4.3 Gráfico para t_{sessao} máximo em função de b_u para vários valores de N. 35
4.4 Tráfego médio usado por um sistema de 2000 nós, no momento $t = 20$ segundos (a), no momento $t = 1800$ segundos (b) e no momento $t = 3600$ segundos (c), quando estão a difundir mensagens de dimensão $m_t = 500$B ao ritmo $r = 1$ mensagens por segundo 38

5.1 Gráfico da função densidade de probabilidade para o tempo entre chegadas consecutivas, modelado por um processo de *Poisson* homogéneo com $\lambda_{chegadas} = 25$ o que equivale a um valor esperado de 25 centésimos de segundo. 41
5.2 Gráfico da *Comulative Distribution Function* para o tempo de duração de uma sessão modelada por uma distribuição de *Weibull* com parâmetros $\lambda sessao = 5000$ e $k = 0,5$ 42
5.3 Resultados obtidos com a distribuição do tempo entre chegadas parametrizada com $\lambda_{chegadas} = 1/2$ 45
5.4 Resultados obtidos com distribuição do tempo entre chegadas parametrizada com $\lambda_{chegadas} = 1/4$ 46
5.5 Resultados obtidos com distribuição do tempo entre chegadas parametrizada com $\lambda_{chegadas} = 1/10$ 47
5.6 Capacidade de *upstream* necessária, b_u (Bytes/s), em função da taxa de eventos observada no pior caso admitindo $m = 500B$. 49
5.7 Capacidade de *upstream* (b_u em Bytes/s) que cada nó deve despender em função do tempo de sessão médio dos nós, num sistema com $N = 10000$ nós, assumindo um modelo de *churn* simples 50

6.1 Diagrama de uma arquitectura típica de super-nós 52
6.2 Diagrama de uma arquitectura super-nós em que existem filhos com mais do que um super-nó pai 53
6.3 Probabilidade de um super-nó promover um nó ao adicionar um novo filho em função do número de filhos, dada pela função 6.1 com $X = 6$. 57
6.4 Algoritmo de entrada de um novo nó filho 58
6.5 Algoritmo executado por um super-nós ao receber uma mensagem do tipo PARENT_REQUEST de um nó a pedir para ser filho. A função p(X) corresponde à função representada pela expressão (6.1). 59
6.6 Resultados obtidos perante condições óptimas de selecção de nós semente e selecção de nós a promover. Com $\lambda_{entrada} = 1/4 \times 100$ o que equivale a um valor esperado de 4 chegadas por segundo e os parâmetros da tabela 6.1 62
6.7 Resultados obtidos perante condições óptimas de selecção de nós semente e selecção de nós a promover. Com $\lambda_{entrada} = 1/2 \times 100$ o que equivale a um valor esperado de 2 chegadas por segundo e os parâmetros da tabela 6.1 63
6.8 Resultados obtidos utilizando escolha aleatória de nós semente com escolha do melhor filho na altura da promoção ao utilizar $\lambda_{entrada} = 1/4 \times 100$ o que equivale a um valor esperado de 4 chegadas por segundo e os parâmetros da tabela 6.1 65
6.9 Resultados obtidos perante condições óptimas de selecção de nós semente e com escolha do pior filho na altura da promoção. Utilizando $\lambda_{entrada} = 1/4 \times 100$ o que equivale a um valor esperado de 4 chegadas por segundo e os parâmetros da tabela 6.1 66
6.10 Resultados obtidos perante condições óptimas de selecção de nós semente e com escolha do pior filho na altura da promoção. Utilizando $\lambda_{entrada} = 1/2 \times 100$ o que equivale a um valor esperado de 2 chegadas por segundo e os parâmetros da tabela 6.1 67
6.11 Resultados obtidos utilizando escolha aleatória de nós semente e com escolha do pior filho na altura da promoção. Utilizando $\lambda_{entrada} = 1/4 \times 100$ o que equivale a um valor esperado de 4 chegadas por segundo e os parâmetros da tabela 6.1 68
6.12 Resultados obtidos utilizando escolha aleatória de nós semente e escolha do nó que é filho há mais tempo para ser promovido ao utilizar $\lambda_{entrada} = 1/4 \times 100$ o que equivale a um valor esperado de 4 chegadas por segundo e os parâmetros da tabela 6.1 70
6.13 Resultados obtidos utilizando escolha aleatória de nós semente e escolha aleatória do filho a promover, utilizando $\lambda_{entrada} = 1/4 \times 100$ o que equivale a um valor esperado de 4 chegadas por segundo e os parâmetros da tabela 6.1 71

6.14 Resultados obtidos com uma simulação em que não foi utilizada delegação de trabalho para os filhos 74
6.15 Resultados obtidos com delegação de trabalho para os filhos com $G = 2$ 76
6.16 Resultados obtidos com delegação de trabalho para os filhos com $G = 4$ 77
6.17 Resultados obtidos com delegação de trabalho para os filhos com $G = 8$ 78
6.18 Resultados obtidos com delegação de trabalho para os filhos com $G = 10$ 79
6.19 Comparação dos resultados obtidos face ao esperado. A curva referente à variável "Upload Average Real" corresponde aos valores nas simulações apresentadas. A curva da variável "Upload Average Teórico" toma os valores da expressão (6.3), $b_u = \frac{400}{G}$. 80

A.1 Modelnet setup used 89

Lista de Tabelas

4.1 Valores dos parâmetros utilizados na análise do algoritmo utilizando árvores aleatórias 33

5.1 Parâmetros usados nas simulações com o modelo de *churn* 44

6.1 Parâmetros usados na simulação com super-nós 58
6.2 Critérios utilizados durante as simulações com o algoritmo de super-nós. 60
6.3 Diminuição obtida dos valores de "Upload Average" com a utilização dos diversos critérios e verificados pela observação dos gráficos das figuras apresentadas, com $\lambda_{chegadas} = 1/4 \times 100$ o que equivale a um valor esperado de 4 entradas/segundo. 72

1. Introdução

1.1 O Projecto LiveFeeds e o seu algoritmo de filiação

A natureza da *World Wide Web*, ou simplesmente WWW, leva a que os utilizadores precisem de ser eles próprios a procurar actualizações às páginas que costumam visitar normalmente. Para resolver este problema, existe o conceito de *Web Syndication* em que os utilizadores subscrevem canais de notificação, também designados por *feeds RSS - Really Simple Syndication*, daqui para a frente designados por *feeds*, para receberem actualizações sobre conteúdos em que estejam interessados.

Um *feed* é um ficheiro XML e contém notícias sobre conteúdos em que os utilizadores podem estar interessados. O *feed* é alterado quando o responsável pelo *feed* deseja que os utilizadores que o tenham subscrito sejam notificados de que existem novos conteúdos disponíveis, por exemplo quando surgem novas notícias sobre certos conteúdos. Um *feed* pode conter novos artigos de um *blog*, novas notícias de um jornal, etc.

Um utilizador subscreve *feeds* e actualiza-os periodicamente monitorizando as alterações aos ficheiros XML, e uma vez que os *feeds* são normalmente fornecidos por um servidor HTTP, o utilizador necessita de efectuar um pedido HTTP ao servidor sempre que queira actualizar o seu *feed*. Desta forma, é colocada uma grande carga nos servidores que oferecem o *feed*, pois podem existir grandes quantidades de utilizadores a actualizar o mesmo *feed* com elevada frequência.

O projecto LiveFeeds pretende dar uma solução para este problema, através da criação de um sistema *P2P*[1] em que os utilizadores colaboraram entre si para disseminar as alterações que detectem sobre um *feed*. Assim, a quantidade de pedidos sobre o servidor HTTP em relação ao *feed* diminui drasticamente, pois é apenas necessário que um pequeno número de utilizadores obtenha as alterações sobre o *feed* e disseminem essas alterações sobre toda uma rede *P2P* de utilizadores que desejem ser notificados sobre essas alterações e que portanto não precisam de estar constantemente a interrogar o servidor HTTP sobre actualizações que possam ter ocorrido.

O LiveFeeds tem assim como âmbito um sistema editor/subscritor [2]. Um sistema editor/subscritor permite a comunicação entre várias partes através de mensagens assíncronas de múltiplos editores para múltiplos subscritores. Uma das características principais deste tipo de sistemas é o facto de os editores estarem separados dos subscritores, tanto no espaço como no tempo, ou seja, uma comunicação entre partes não requer nenhuma agregação prévia entre as mesmas, os participantes apenas necessitam de se conectar a "espaços de informação"

[1] A abreviatura P2P é usada para o termo *Peer to Peer*, que designa sistemas distribuídos estruturados de forma alternativa à estruturação cliente-servidor. O termo pode ser traduzido em português por sistemas par-a-par ou parceiro-a-parceiro. No que se segue utilizaremos a notação P2P, bem estabelecida para designar este tipo de sistemas.

[2] Do inglês *publish/subscribe*

associados com os seus interesses. Os *feeds RSS* são um exemplo conhecido de um sistema editor/subscritor [15]. O projecto LiveFeeds pretende dar uma solução para o problema da disseminação de informação em sistemas editor/subscritor baseada em mensagens e usa a *Web Syndication*, na vertente que utiliza *feeds RSS*, como estudo de caso.

Este trabalho debruça-se sobre as especificidades da implementação e avaliação de um sistema *P2P* que solucione o problema do LiveFeeds com ênfase no algoritmo de filiação do sistema, isto é, a gestão de entradas e saídas dos utilizadores do LiveFeeds.

O algoritmo de filiação é um ponto crucial de um sistema P2P. Apesar de, do ponto de vista do utilizador final, este algoritmo ser apenas *overhead*, uma vez que o utilizador pretende apenas obter notificações sobre conteúdos e não necessita de saber como obtém essas notificações, é essencial que o algoritmo de filiação esteja bem desenhado e afinado para que o sistema funcione com os níveis de qualidade e desempenho requeridos pelos utilizadores.

1.2 Descrição e contexto

Num sistema baseado no modelo editor/subscritor, os subscritores registam o seu interesse num evento, ou num padrão de eventos e são posteriormente e assincronamente notificados sobre eventos gerados por editores que satisfaçam o padrão [6].

Como já foi referido em 1.1, os sistemas editor/subscritores permitem que várias partes comuniquem através de mensagens assíncronas, e uma das características principais desta aproximação é o facto de os editores e subscritores estarem separados no espaço e no tempo, sem que seja necessária nenhuma ligação explícita prévia entre as partes [15].

Uma vez que os subscritores subscrevem tópicos ou conteúdos, as notificações enviadas pelos editores para uma rede de um sistema editor/subscritor devem ser encaminhadas dentro da rede para os nós interessados em receber tais notificações. O sistema deve estar bem desenhado para que as mensagens entre as partes intervenientes possam chegar aos destinatários de maneira eficiente.

Para a resolução deste problema no contexto do projecto LiveFeeds, foi escolhida, como ponto de partida, uma aproximação em que as mensagens são encaminhadas num número constante de passos. A utilização de um método de encaminhamento deste tipo tem impactos directos no desenho do algoritmo de filiação do sistema, uma vez que os utilizadores ao entrarem e saírem do sistema devem provocar alterações na rede de maneira a que seja possível manter o encaminhamento eficiente de mensagens com um número constante de passos. Uma aproximação possível para que se possa implementar um algoritmo de encaminhamento num número constante de passos consiste na utilização uma replicação parcial ou total da filiação do sistema através de todos os nós, esta necessidade obriga a que o algoritmo de filiação seja desenhado de maneira a manter essa replicação.

O algoritmo de filiação utilizado no projecto LiveFeeds é baseado na utilização de árvores aleatórias de difusão, este algoritmo é descrito em maior detalhe no capítulo 2 e foi já apresentado em [16] onde é também feita uma análise aos custos e à viabilidade desta aproximação.

Para resolver o problema já referido do encaminhamento de mensagens num sistema P2P, foram sugeridas várias soluções. Em particular, em [22, 10] é sugerida uma aproximação para este problema, sendo analisada a possibilidade de os algoritmos já existentes para o encaminhamento em redes P2P, normalmente baseados na utilização de uma *DHT - Distributed Hash Table*[3], serem demasiado complexos para certos tipos de sistemas, sendo possível a utilização de algoritmos mais simples e com custos mais baixos para resolver o problema. A utilização de encaminhamento em apenas um passo é um desses algoritmos.

Segundo [22] a possibilidade de utilização deste tipo de algoritmos está confinada aos sistemas em que os utilizadores seguem um modelo de filiação estável, ou seja, a média de tempo de sessão é suficientemente alta para que existam poucas entradas e saídas por unidade de tempo. Além disso, é sugerido que a utilização de técnicas de encaminhamentos complexas só se torna vantajoso em sistemas deste tipo com mais de 10^6 utilizadores.

Em [10] é de novo questionado o facto de serem necessários algoritmos complexos de reencaminhamento nas *DHT* e é proposto um algoritmo para a implementação de uma *DHT* que suporta reencaminhamento em apenas um passo e são analisados os custos necessários para essa implementação, tendo no entanto em conta um modelo de estabilidade dos nós demasiado simplificado e provavelmente irrealista, uma vez que dificilmente um sistema real segue um tal modelo.

As soluções apresentadas em [22, 10] passam por uma replicação total ou parcial da filiação dos nós através de todos os nós do sistema. Este tipo de replicação pode ser facilmente implementada num contexto em que a informação de filiação dos nós consiste em pouco mais informação do que o identificador de um utilizador no sistema, mas tem fortes consequências quando é vantajoso que a informação de filiação consista em algo mais, tal como uma lista de interesses de subscrição, como é o caso do LiveFeeds. A replicação de informação de filiação de todos os nós por todos os outros nós pode atingir grandes proporções em relação à capacidade de comunicação requerida quando para cada nó é necessário, por exemplo, cerca de 1 *kilobyte* para representar a sua informação de filiação.

Além do tamanho da informação de filiação de cada utilizador, existe um outro factor que determina a aplicabilidade de uma solução de encaminhamento com um número constante de passos, o *churn* – o processo contínuo de chegada e partida de nós [21]. Ao verificar-se uma grande instabilidade na filiação do sistema, será mais difícil manter uma replicação fiável da filiação por todos os nós e é necessário que um utilizador tenha que contribuir com maior largura de banda para manter essa replicação, sendo possível que a aplicabilidade do sistema em causa seja assim comprometida.

[3] Tabela de dispersão distribuída

Para se poder estudar a possibilidade da utilização de algoritmos de encaminhamento com um número fixo de passos num sistema editor/subscritor, é necessário estudar a problemática da escala do sistema, a qual está intimamente ligada à dimensão do mesmo, à quantidade de informação de filiação a replicar e ao modelo *churn* que esteja adaptado ao contexto de utilização do sistema em estudo.

1.3 Ambiente de simulação

Uma parte significativa do trabalho apresentado é o teste do algoritmo de filiação e das alternativas propostas num ambiente próximo da sua execução real. O estudo dos vários métodos que se poderiam utilizar para se proceder à experimentação e avaliação da solução, como sistemas de emulação e simulação, constitui assim uma parte do trabalho realizado (cf. capítulo 3.4), tendo sido escolhida a utilização de um simulador para se proceder à análise do algoritmo de filiação do LiveFeeds. O ambiente de simulação utilizado é aqui apresentado.

O simulador utilizado foi criado no quadro do projecto LiveFeeds e foi desenvolvido utilizando a linguagem *Java* [17], podendo ser estendido com novas classes para que se possam implementar novas funcionalidades, comportamentos e algoritmos de maneira a que seja possível fazer uma correcta avaliação dos mesmos.Através de simulação é possível ter controlo total sobre todas as variáveis simuladas. O simulador é executado utilizando apenas um computador, sendo assim fácil de obter dados relativos ao comportamento dos algoritmos em estudo. Através dos dados obtidos, é possível traçar gráficos que representam a evolução do comportamento dos algoritmos estudados. Existem partes importantes de um sistema distribuído que não são no entanto simuladas, de forma a que seja possível fazer simulações com um grande número de nós. Toda a lógica de comunicações ao nível de rede entre os nós é abstraída, não sendo considerados efeitos como o preenchimento de filas de espera ou o tempo de estabelecimento de conexões *TCP*. Para que seja fácil observar o comportamento dos algoritmos utilizados, existe uma interface gráfica que pode ser configurada para que seja possível observar facilmente a evolução do estado do sistema. As figuras 1.1 e 1.2 mostram o aspecto da janela da interface gráfica referente à estrutura do sistema ao testar o comportamento de diferentes algoritmos.

Ao mesmo tempo que se pode observar o aspecto do sistema, podem-se recolher dados e apresentar gráficos com estatísticas sobre esses dados que vão sendo actualizados com novos dados ao longo do tempo. As figuras 1.3 (a) e 1.3 (b) mostram um exemplo dos gráficos obtidos que podem ser guardados para posterior análise. Os dados e estatísticas mostradas em todas as janelas podem ser alteradas para que se possam observar todos os aspectos da simulação.

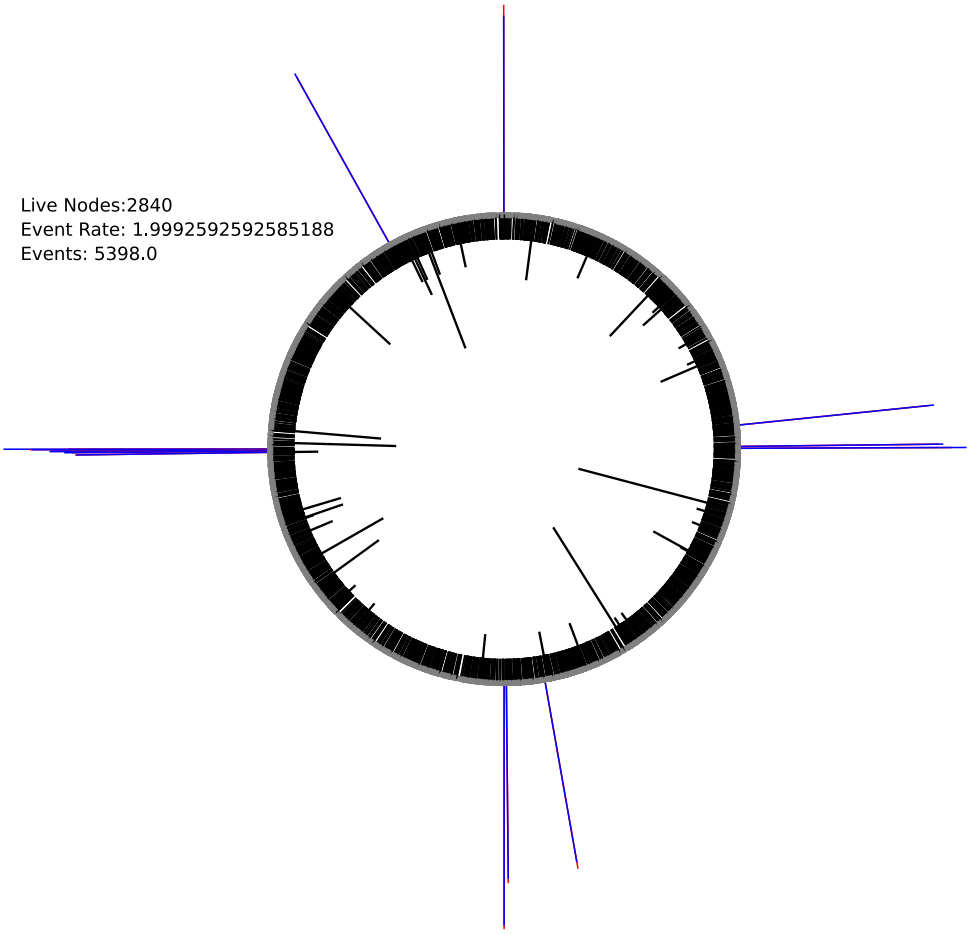


Figura 1.1: Aspecto da janela da interface gráfica do simulador referente à estrutura do sistema ao utilizar o algoritmo de visibilidade completa

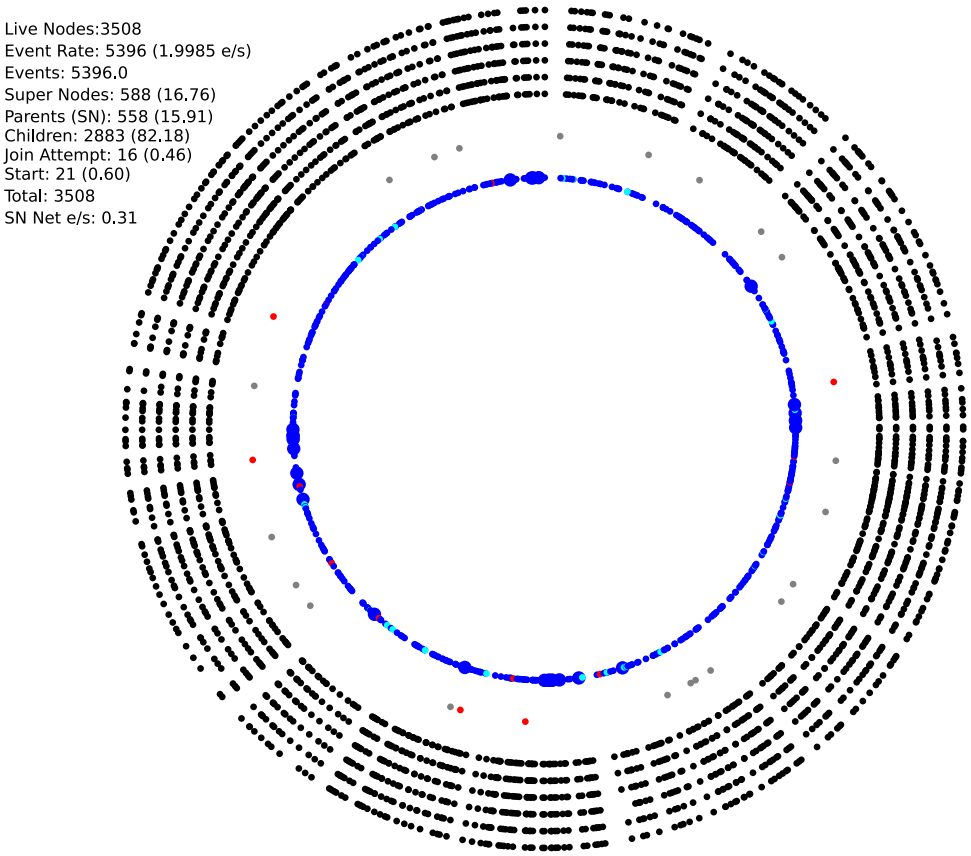
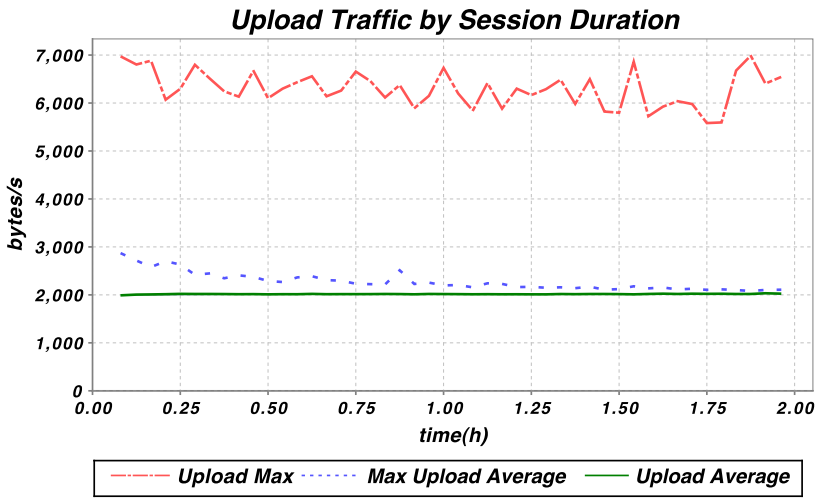
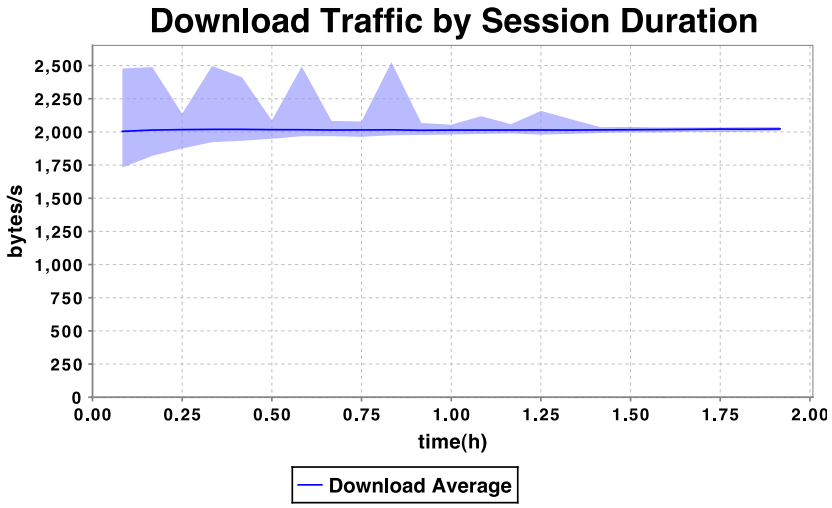


Figura 1.2: Aspecto da janela da interface gráfica do simulador referente à estrutura do sistema ao utilizar o algoritmo de super-nós



(a) Tráfego de *upstream*



(b) Tráfego de *downstream*

Figura 1.3: Gráficos obtidos com o simulador do LiveFeeds para uma das simulações realizadas

1.4 Principais contribuições

Dentro do contexto descrito em 1.2, consistindo na utilização de uma aproximação de encaminhamento com um número constante de passos, existem vários problemas a ter em consideração.

Um dos problemas foi a avaliação da aplicabilidade deste tipo de aproximação para o sistema em estudo, o LiveFeeds, bem como os limites deste tipo de mecanismo. Para que fosse possível fazer esta análise, foi necessário encontrar um modelo de *churn* realista que modelasse correctamente o comportamento esperado dos utilizadores do LiveFeeds. Após ter sido construído este modelo de *churn*, foi possível avaliar a exequibilidade do mecanismo de encaminhamento com visibilidade completa.

Apesar deste trabalho ter sido desenvolvido dentro de um contexto em que já existia uma

versão do algoritmo de filiação com visibilidade completa, esse algoritmo não tinha ainda sido testado e analisado de forma aprofundada.

As contribuições deste trabalho consistem assim no desenvolvimento de um modelo de *churn* realista e na utilização desse modelo na analise do algoritmo de filiação em condições próximas da sua execução real determinando assim quais os custos em termos de capacidade de *upstream* e *downstream* de cada nó para que o algoritmo de difusão com visibilidade completa utilizando árvores aleatórias tenha viabilidade.

Complementarmente, desenvolveu-se e analisou-se uma nova versão do algoritmo mais escalável que a anterior, para quando se consideram utilizadores com comportamento muito dinâmico e que consiste em utilizar super-nós para introduzir uma hierarquização do sistema. Assim, os nós mais estáveis e melhor ligados têm uma visibilidade completa da rede, enquanto que os nós recém chegados ou mais instáveis, têm uma visibilidade reduzida da rede, ficando dependentes de um ou mais *super-peers* para poderem operar. Em síntese, esta dissertação compreende as seguintes contribuições:

- Desenvolvimento de um modelo de *churn* realista;
- Teste do algoritmo com o modelo de *churn* realista desenvolvido;
- Teste do algoritmo em condições próximas da sua execução real;
- Análise do custo e da viabilidade da implementação do algoritmo de difusão com visibilidade completa utilizando árvores aleatórias;
- Estudo de uma alternativa para o algoritmo de difusão do LiveFeeds sem visibilidade completa, com base numa solução hierárquica.

1.5 Organização do documento

Os restantes capítulos desta dissertação são os seguintes. Em 2 é apresentado o algoritmo de filiação do LiveFeeds na vertente que utiliza visibilidade completa bem como todas as particularidades especificas de tal algoritmo.

O trabalho relacionado com a área de estudo desta dissertação é apresentado no capítulo 3, nomeadamente trabalhos sobre encaminhamento em sistemas P2P, modelos de *churn*, métodos de avaliação de sistemas distribuídos e desenho de sistemas com super-nós.

Em 4 é feita uma análise do custo de difusão utilizando árvores aleatórias e são feitas simulações preliminares que são analisadas no âmbito dos resultados obtidos de forma analítica.

O desenvolvimento de um modelo de *churn* realista é documentado no capítulo 5 onde são também feitas simulações que utilizam tal modelo de forma a analisar o comportamento do

algoritmo de filiação com um número constante de passos perante o comportamento dinâmico dos utilizadores.

No capítulo 6 é apresentada uma alternativa para o algoritmo de visibilidade completa que se baseia numa hierarquização do sistema e na noção de super-nó.

As conclusões são expostas no capítulo 7 onde é também referido o trabalho futuro a desenvolver.

Uma vez que a documentação existente sobre a instalação e configuração do emulador de rede ModelNet é muito escassa, foi também desenvolvido um relatório com os passos detalhados para que se possa utilizar este emulador. Este relatório é apresentado no anexo A. Este resultado fez parte dos estudos realizados para afinar o ambiente de teste que iria ser utilizado.

2 . O Algoritmo de filiação

A difusão filtrada baseada em conteúdo (*content-based*) em sistemas editor/subscritor é um problema complexo e para o qual têm sido propostas várias soluções [15]. O sistema LiveFeeds pretende propor uma solução para este problema. Para tal, parte de uma abordagem em que todos os nós conhecem todos os outros nós. O algoritmo aqui apresentado como ponto de partida desta dissertação é um algoritmo de visibilidade completa, todos os nós conhecem todos os outros nós do sistema e as respectivas subscrições. Este tipo de aproximação enquadra-se na filosofia *One Hop Routing* [10]. Uma vantagem proveniente da adopção de uma visibilidade completa das subscrições, é que o esforço despendido no encaminhamento de uma mensagem, que inclui a avaliação dos filtros subscritos pelos nós, recai apenas nos nós a que ela se destina, como a seguir se mostrará. Com este tipo de aproximação pretende-se também garantir a inexistência de falsos positivos, mensagens que são entregues a nós que não deviam recebe-las (*spam*), e de falsos negativos, isto é, nós que deveriam receber mensagens e não as recebem.

Neste capítulo é apresentado o algoritmo de filiação com visibilidade completa utilizado pelo LiveFeeds, e cujos custos são analisados em detalhe nos capítulos seguintes.

2.1 Apresentação do algoritmo de visibilidade completa

O algoritmo utiliza a informação completa que tem sobre a filiação de todos os nós para construir árvores de difusão aleatórias para disseminar a informação de filiação de novos nós. Paralelamente, é executado um algoritmo epidémico de forma a eliminar as inconsistências com origem nas entradas concorrentes e nas falhas de nós.

Em concreto, cada nó é conhecido por um identificador aleatório numérico (por hipótese, sem colisões e com uma distribuição uniforme), pelo seu endereço e por uma lista de subscrições (filtro). O domínio dos identificadores dos nós está dividido num pequeno número de fatias. Em cada fatia, o nó activo com o identificador numericamente menor é, tendencialmente, conhecido como o líder da fatia (*slice leader*) e tem um papel destacado no algoritmo. Não existe qualquer tipo de coordenação entre os *slice leaders*. É ainda de notar que, momentaneamente, pode haver mais do que um *slice leader* em cada fatia. O sistema assume, também, que existe conectividade entre todos os nós.

Quando um nó pretende juntar-se ao sistema, fá-lo através de um nó já presente no sistema, para isso, um nó que queira entrar no sistema deve conhecer um qualquer subconjunto dos nós já presentes no sistema. Este nó auxiliar serve para encaminhar o pedido de entrada para o *slice leader* relevante, determinado com base no identificador do nó que pretende entrar. A função de cada *slice leader* consiste em agregar pedidos de entrada antes de iniciar a sua difusão pelos nós do sistema. Essa acumulação realiza-se até ser atingido um número razoável de eventos, ou

até se esgotar o tempo máximo permitido de acumulação e destina-se a permitir a compressão de dados e a tornar o sistema mais eficaz. Nessa altura, é iniciada a difusão dos eventos de alteração da filiação pelo sistema através de uma árvore de grau G, construída no decurso do processo de difusão. Para tal, o *slice leader* começa por dividir o universo dos identificadores em G sub-intervalos. Para cada um deles, o *slice leader* selecciona um nó escolhido aleatoriamente a quem entrega os eventos a difundir e o sub-intervalo dos identificadores que o nó escolhido, por sua vez, deverá tratar. O processo repete-se recursivamente, em cada nó da árvore aleatória assim construída de forma aleatória, até se chegar ao ponto em que os intervalos ficam tão estreitos que os nós que restam podem ser tratados simplesmente como folhas. Um nó sabe que entrou no sistema quando recebe a mensagem que difunde a sua entrada no mesmo. Se tal não acontecer num determinado intervalo de tempo, o processo de entrada é repetido totalmente.

A figura 2.1 ajuda a compreender o algoritmo de entrada de um novo nó. Neste caso existem 19 nós no sistema, sendo um deles *slice-leader*, o nó *sl*. Um nó *a* contacta outro nó *b* para tentar entrar no sistema. O nó *b* reencaminha o nó *a* para o nó *sl*, o *slice leader* responsável pela fatia do identificador de *a*. O nó *sl* aceita a entrada de *a* e quando tiver acumulado pedidos suficientes ou passado um intervalo de tempo pré-definido, difunde a nova entrada. Neste caso o sistema está configurado para ter duas fatias ($G = 2$), o *slice leader* divide o espaço dos identificadores e envia a mensagem uma vez para cada fatia. Os nós que recebem a mensagem vinda do *slice leader*, neste caso *b* e *c*, difundem a mensagem dentro da sua fatia, enviando-a a $G = 2$ nós. O processo repete-se até que o número de nós restantes na fatia seja igual ou inferior a G (quando a mensagem é recebida pelos nós m, n, o, d, h e i) altura em que a mensagem é apenas enviada aos nós folha da árvore de difusão (nós r, s, p, q, f, g, j e k) que não reencaminham a mensagem pois a difusão está completa dentro daquela fatia. Os nós $b, c, l, m, d, e, n, o, h$ e i são designados nós interiores. Mais tarde, em difusões de filiação posteriores, o nó *a* será incluído numa das árvores formadas e terá assim conhecimento da sua entrada no sistema. Note-se que apesar do diagrama da figura 2.1 representar os nós organizados numa árvore com determinada topologia, esta topologia não existe de forma permanente, na próxima difusão irá ser construída uma nova árvore que pode não ter o mesmo aspecto. Todas as árvores construídas são aleatórias.

Sendo o processo de difusão anterior, essencialmente *best-effort*, a visão da filiação do sistema que os nós assim adquirem é, inevitavelmente, imperfeita. Falhas ocasionais de nós e eventos de difusão concorrentes geram inconsistências que têm que ser corrigidas. Para tal, cada novo evento agregado de filiação, iniciado por um *slice leader*, é identificado por um número de sequência local único e uma etiqueta (vectorial) global. Esta última tem por função permitir registar o evento de filiação emitido mais recentemente por cada um dos *slice leaders* do sistema. Cada uma destas etiquetas globais representa uma "vista" da filiação do sistema, que cada nó pode comparar com a sua própria visão, computada a partir dos eventos de filiação que efectivamente recebeu, a fim de detectar eventos perdidos. A recuperação dessas eventuais

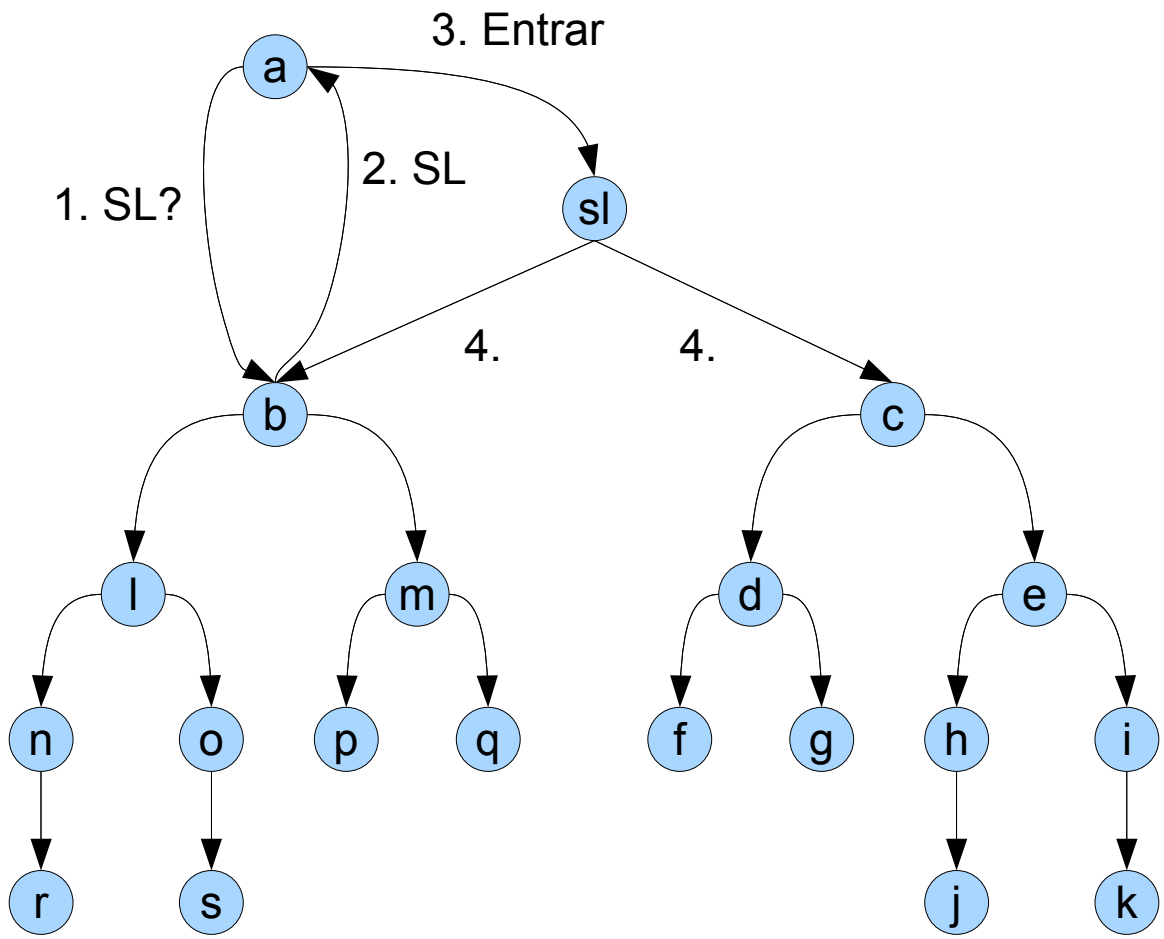


Figura 2.1: Diagrama das comunicações provocadas pela entrada de um novo nó (o nó *a*) num sistema com 19 nós em que existe um único *slice leader* (o nó *sl*) e parametrizado para dividir os identificadores em $G = 2$ fatias

omissões faz-se por epidemia. O processo é simples e consiste em cada nó periodicamente seleccionar um nó escolhido ao acaso e enviar-lhe a etiqueta vectorial que identifica os eventos já vistos, na esperança de receber na volta os eventos em falta.

Note-se, ainda, que assim que um nó é reconhecido como parte do sistema passa a poder ser seleccionado para nó interior da árvore de difusão. Tal facto obriga a que cada nó, previamente à sua entrada, obtenha uma cópia razoavelmente completa e actualizada da informação de filiação do sistema.

Sobre a visão completa e consistente, assim obtida, é implementado o processo de difusão filtrada. Essencialmente, trata-se de uma pesquisa distribuída dos nós cujos filtros aceitam a mensagem, ficando cada nó responsável por realizar uma parte do trabalho que resta, representado sob a forma de um intervalo de identificadores. Essa pesquisa faz-se através uma árvore aleatória que cobre apenas os nós cujo filtro aceita a mensagem em questão, ou seja apenas

estes realizam esta computação distribuída.

2.2 Particularidades do algoritmo de visibilidade completa apresentado

A utilização do algoritmo de visibilidade completa apresentado tem algumas particularidades com especial relevância e que são aqui apresentadas.

O algoritmo de filiação apresentado tem a particularidade de permitir a distribuição de carga segundo diferentes perspectivas:

- Distribuição dos custos de comunicação de *upstream* da difusão da filiação por todos os nós intervenientes
- Distribuição do trabalho de computação na comparação dos filhos apenas pelos nós interessados em receber as mensagens

Existe distribuição da capacidade de *upstream* efectuado pois como será demonstrado, em média, todos os nós contribuem com a mesma capacidade de *upstream*, sendo o valor deste trabalho igual à despendida em capacidade de *downstream*.

Do ponto de vista da distribuição do trabalho de computação, podemos observar que o trabalho feito para receber uma mensagem e calcular para que nós deve ser reencaminhada a mesma, através da comparação dos filtros de subscrições, é feito apenas pelos nós interessados em receber a mensagem em questão.

Atendendo a que apenas as entradas são difundidas, as vistas que os nós têm da filiação podem incluir informação de filiação de nós que já tenham saído do sistema e cuja saída ainda não foi detectada por não ter sido feita nenhuma tentativa de comunicação entre os nós em questão. A vista actual de cada nó deve ser examinada periodicamente para eliminar todos os nós que estão registados no sistema há mais tempo do que o tempo de sessão máximo. Evita-se assim o crescimento infinito das vistas de cada nó e eliminam-se inconsistências criadas pelas saídas de nós.

As falhas e saídas de nós induzem inconsistências nas vistas dos nós que podem demorar algum tempo a ser resolvidas, mas não apresentam problemas de maior ao algoritmo de filiação devido à sua natureza aleatória, as árvores de difusão formadas não têm sempre a mesma forma, não sendo sempre escolhidos os mesmos nós para as mesmas posições.

Mesmo que existam nós que não reencaminhem devidamente as mensagens recebidas por já não estarem no sistema, os erros esporádicos assim provocados podem ser recuperados através do mecanismo epidémico de recuperação de falhas e pela detecção da impossibilidade de comunicação com um nó que saiu do sistema.

Parte das inconsistências provêm da concorrência normalmente subjacente a este tipo de algoritmos distribuídos. Os eventos do sistema são processadas pelos nós presentes no sistema

de uma forma possivelmente não serializável, o que leva a que diferentes nós possam ter diferentes visões do estado da filiação, sendo este tipo de inconsistências reparado pelo algoritmo epidémico de reparação de falhas. Além das inconsistências introduzidas pela natureza concorrencial dos algoritmos distribuídos utilizados existem também inconsistências provocadas pelas falhas ocasionais de nós, podendo essas falhas ser de três tipos distintos:

1. O nó saiu mas esse facto ainda não foi reconhecido por alguns nós ou por todos os outros nós;

2. O nó está em funcionamento mas está temporariamente ou permanentemente incomunicável;

3. O nó bloqueou[1].

As falhas do tipo 1 e 3 são permanentes, isto é, ao bloquear ou sair, o nó não vai voltar ao seu estado operacional, conhecido por todos os outros nós. As falhas de tipo 2, em que é impossível comunicar com o nó em questão, podem ser temporárias ou permanentes pois pode acontecer que um nó esteja incomunicável apenas durante algum tempo. Todas as falhas citadas têm repercussões no algoritmo de filiação que pode tratar as mesmas de diversas formas.

Se um nó *a* selecciona um outro nó *b* para ser nó interior e *b* está incomunicável a partir de *a*, o algoritmo executado em *a* pode decidir seleccionar outro nó para continuar a difusão, mas caso a incomunicabilidade de *b* seja apenas temporária, *b* vai ficar com uma visão desactualizada do sistema, que terá de ser resolvida posteriormente. No caso em que *b* sofreu uma falha do tipo 1 ou 3 tendo o nó *a* escolhido outro nó para continuar a difusão, não existe qualquer problema.

Caso um nó tenha sido seleccionado para difundir a informação de filiação para um determinado número de nós e tenha bloqueado após ter recebido a mensagem com a informação de filiação a difundir, todo o intervalo atribuído a este nó fica com uma vista inconsistente do estado do sistema, que deverá ser reparada.

Sempre que um nó recebe uma mensagem com nova informação de filiação, pode verificar através das etiquetas temporais se tem uma versão desactualizada da informação de filiação do sistema. Caso essa inconsistência exista, o nó deve proceder à reparação das desactualizações através do algoritmo epidémico.

Existe a possibilidade de durante pequenos períodos de tempo exista mais do que um *slice leader* por fatia. Esta é uma consequência de se escolher sempre o nó com o identificador numericamente menor, pois poderão existir inconsistências na filiação durante algum tempo em que nem todos os nós estão de acordo sobre o nó eleito para *slice leader*. Uma vez que esta inconsistência não se estende por grandes períodos de tempo, a existência de mais do que

[1]Do inglês *crashed*

um *slice leader* por fatia não tem consequências no custo de manutenção da filiação, a taxa de alterações da difusão passa a ser dividindo pelos vários *slice leaders* da fatia, existindo apenas um decréscimo nos ganhos relativos à agregação de vários eventos por parte dos *slice leaders* em relação ao uso de apenas um nó com essa responsabilidade por fatia.

Estas inconsistências na selecção de *slice leaders* levam ao aumento do número de entradas e de etiquetas vectoriais. O mesmo se passa quando um *slice leader* antigo sai do sistema. A existência de um tempo máximo de permanência no sistema permite diminuir o tamanho das etiquetas vectoriais.

O custo do algoritmo epidémico de reparação de falhas pode ser amortizado ao longo de toda a permanência de um um nó dentro do sistema, mas não é no entanto objecto deste estudo.

3 . Trabalho relacionado

Neste capítulo são apresentados todos os trabalhos relacionados com a área de estudo desta dissertação que foram estudados e cujos resultados serviram de base para os estudos desenvolvidos nos capítulos seguintes.

O LiveFeeds é um sistema P2P editor/subscritor e assim são apresentados primeiramente os trabalhos relacionados nesta área que serviram como base teórica para o estudo deste tipo de sistemas.

Posteriormente e dado que o que se pretendeu analisar foi um algoritmo de encaminhamento com um número constante de passos, são apresentados alguns trabalhos sobre esta temática.

Para que se pudesse fazer uma avaliação realista do comportamento do algoritmo de filiação com visibilidade completa, foi necessário desenvolver um modelo de *churn* realista que representasse correctamente a dinâmica dos possíveis utilizadores do LiveFeeds e assim foram estudados alguns artigos científicos sobre assunto.

O algoritmo que se pretendeu estudar é um algoritmo distribuído. Existem diversas formas para avaliar o desempenho e o desenho deste tipo de algoritmos. As várias metodologias que podem ser utilizadas neste contexto são também apresentadas neste capítulo.

Depois de se ter avaliado o algoritmo de encaminhamento utilizando um número constante de passos no contexto do LiveFeeds, foi possível concluir que o mesmo é aplicável num quadro em que os utilizadores tenham um comportamento não muito dinâmico. Uma das soluções caso se pretenda suportar utilizadores com comportamento muito dinâmico seria introduzir uma hierarquização do sistema recorrendo à noção de super-nós. O trabalho relacionado referente a esta matéria é também aqui apresentado.

3.1 Encaminhamento em sistemas P2P editor/subscritor

Um sistema editor/subscritor permite aos editores e subscritores de eventos a comunicação por meio de mensagens assíncronas de muitos-para-muitos[1]. Apesar de este paradigma não ser uma novidade, ganhou uma nova dimensão com a emergência de novos sistemas globais de larga escala que utilizam a Internet [15]. O projecto LiveFeeds baseia-se neste paradigma para oferecer uma solução para o problema da disseminação de eventos através de vários utilizadores. O LiveFeeds, sendo um sistema editor/subscritor P2P utiliza um algoritmo de encaminhamento de mensagens para que os editores possam comunicar com os subscritores. O trabalho [15] oferece-nos uma observação sobre os trabalhos mais importantes sobre encaminhamento de notificações em sistemas distribuídos editor/subscritor baseados no conteúdo. De acordo com

[1] Do inglês many-to-many

[15], os sistemas editor/subscritor, podem ser *topic-based*[2] ou *content-based*[3]. Nos sistemas *topic-based*, cada espaço de informação está associado a um tópico, grupo ou canal. Os editores publicam as notificações para um canal e os subscritores subscrevem um canal ou canais em que estão interessados. Nos sistemas *content-based*, os subscritores subscrevem não um canal, mas um filtro semântico sobre os conteúdos das notificações [15]. O LiveFeeds é um sistema editor/subscritor *content-based*.

A utilização de sistemas editor/subscritor tem a vantagem de não ser necessário que as partes efectuem qualquer ligação explícita prévia entre si, estando os editores e subscritores desacoplados no espaço e no tempo [6]. Os participantes conectam-se apenas a espaços de informação associados com os seus interesses e não necessitam de qualquer comunicação prévia explícita entre estes [15].

A implementação de um sistema editor/subscritor utilizando uma arquitectura cliente-servidor, sofre do problema de escalabilidade deste tipo de sistemas e uma aproximação P2P surge naturalmente como uma solução alternativa. Em [11] é sugerida uma solução para o problema da disseminação de eventos dentro do paradigma editor/subscritor utilizando um sistema P2P baseado numa tabela de dispersão distribuída (DHT). Esta aproximação é atractiva, uma vez que esta arquitectura oferece flexibilidade, modularidade e escalabilidade. Com a utilização de um arquitectura P2P, qualquer utilizador pode actuar como editor ou subscritor e contribuir com poder de computação e comunicação para o sistema editor/subscritor.

O problema do método de encaminhamento a utilizar é inerente à utilização de uma aproximação P2P para a implementação de um sistema editor/subscritor. O problema consiste em saber como encaminhar eficientemente cada mensagem dos editores para todos os subscritores interessados no evento subjacente. O problema assume uma maior dimensão em sistemas editor/subscritor *content-based* uma vez que é necessário que se filtrem as mensagens que são encaminhadas de acordo com as preferências dos subscritores. Este problema é análogo ao problema do encaminhamento de mensagens *multicast* e normalmente são utilizados algoritmos semelhantes, com a diferença de que as árvores de difusão têm de ser dinamicamente podadas em função do conteúdo das mensagens e dos interesses dos subscritores [15]. Além dos algoritmos existentes relevantes para a resolução do problema de encaminhamento em redes P2P editor/subscritor, referidos em [15], podem ainda ser seguidas outras aproximações. No caso do projecto LiveFeeds, foi escolhida uma aproximação de encaminhamento utilizando um número constante de passos.

[2]Baseados em tópicos
[3]Baseados em conteúdo

3.2 Encaminhamento utilizando um número constante de passos

O problema do encaminhamento de mensagens é um dos problemas a resolver quando se pretende implementar um sistema P2P. Juntamente com a proposta da utilização de *DHT - Distributed Hash Tables*, surgiram várias soluções que passam todas pela utilização de encaminhamento multi-passo[4].

Numa implementação de uma *DHT* são normalmente utilizados métodos de encaminhamento com múltiplos passos de encaminhamento, tipicamente na ordem de $O(log(N))$, sendo N o número de nós no sistema. A utilização deste tipo de encaminhamento está relacionada com o facto de se pensar que o custo de manter a informação de filiação de todo o sistema em cada um dos nós é demasiado grande para sistemas de larga escala. Para que seja possível diminuir a quantidade de informação requerida em cada nó, uma solução possível é aumentar o número de passos do encaminhamento. Em [22] é analisado até que ponto o tamanho do sistema pode afectar a possibilidade de os nós manterem uma réplica total dessa filiação e é referido que a possibilidade de tal manutenção é possível e benéfica, quando são reunidas determinadas condições, como um tempo de sessão dos utilizadores elevado e disponibilidade de largura de banda adequada por parte dos mesmos. Num sistema que siga um modelo de *churn* demasiado dinâmico, será uma tarefa complexa manter a replicação da filiação necessária para que possa ser implementada um algoritmo de encaminhamento com um número constante de passos. É portanto concluído em [22] que a utilização de encaminhamento com um número constante de passos é vantajoso sempre que a filiação do sistema seja estável o suficiente para que a replicação total da filiação através de todos os nós do sistema seja uma operação de custo aceitável. A utilização de uma aproximação de encaminhamento num número variável de passos só deve ser equacionada apenas para um sistema "estável" comportando vários milhões de utilizadores.

A aplicabilidade de um mecanismo de encaminhamento com um número constante de passos é demonstrada em [10], onde são sugeridos dois algoritmos deste tipo. Um utiliza apenas um passo para encaminhamento, o que requer uma replicação total da filiação e outro utiliza dois passos para encaminhamento, requerendo apenas uma replicação parcial da replicação através dos nós do sistema. A solução proposta em [10] baseia-se novamente na premissa de que a utilização de uma aproximação do tipo multi-passo é demasiado complexa para certo tipo de sistemas. Os algoritmos propostos são analisados tanto formalmente como por simulação e é concluído que estes são aplicáveis para certos tipos de sistemas, perante uma variação da filiação adequada.

O modelo de *churn* utilizado em [22, 10] poderá ser demasiado simplificado, o que pode comprometer a aplicabilidade dos algoritmos propostos. É utilizada como base em [10] uma média de tempo de sessão de várias horas e uma distribuição uniforme da mesma que para

[4]Do inglês *multi-hop*

um sistema do tipo do LiveFeeds não será realista, facto que poderemos verificar ao construir um modelo de *churn* adequado à modelação do comportamento dos utilizadores do LiveFeeds. Neste enquadramento, a situação agrava-se pois a informação de filiação de cada utilizador não consiste apenas em poucos *bytes* de informação, essa informação pode ter um tamanho que pode ir até vários *kilobytes*, uma vez que a informação de filiação consiste também na informação sobre os filtros que caracteriza a subscrição do utilizador.

Os mecanismos de encaminhamento em múltiplos passos sofrem do problema de por vezes uma consulta ter uma latência demasiado elevada, o que pode ser um factor crítico dependendo do objectivo do sistema. Uma utilização de uma aproximação de encaminhamento num número constante de passos diminui consideravelmente a latência em cada consulta e este é um factor especialmente importante em sistemas editor/subscritor, como é o caso do LiveFeeds, uma vez que a filtragem e envio de mensagens entre editores e subscritores pode ser bastante mais eficiente caso a latência de encaminhamento seja menor, aumentando portanto o desempenho na propagação de notificações de eventos neste tipo de sistemas.

Outra solução para a concretização de uma aproximação do tipo de encaminhamento com um número constante de passos é dada em [20] através do sistema *Beehive* . Com esta solução, é utilizado um mecanismo de replicação de dados para que as consultas de uma *DHT* sejam satisfeitas com um número constante de passos. Como já foi referido, [20] volta a indicar que as implementações de *DHTs* existentes suportam tipicamente encaminhamento em $O(log(N))$ passos, o que introduz uma latência incomportável para alguns tipos de sistemas, nomeadamente sistemas editor/subscritor. O *Beehive* utiliza uma replicação pro-activa dos dados, ou seja, as réplicas de informação são espalhadas por todos os nós do sistema de acordo com um modelo analítico que reduz o número de passos necessários para resolver uma consulta. Ao diminuir a latência de cada consulta, o *Beehive* capacita uma *DHT* para que esta possa servir sistemas sensíveis em relação aos requerimentos de latência. Existe naturalmente um *trade-off* entre replicação e consequente diminuição do número de passos necessários com os recursos necessários de largura de banda e espaço de armazenamento. É utilizado o exemplo de caso de estudo o sistema *DNS*[5], uma vez que a distribuição das consultas efectuadas segue uma distribuição *Power-Law*. Segundo [20], a aplicabilidade de um sistema de replicação utilizado no *Beehive* só foi equacionada para sistemas em que a distribuição de consultas seja deste tipo. O *Beehive*, apesar das desvantagens relativas a [10], uma vez que foi desenhado apenas tendo em consideração distribuições *Power Law*, tem a vantagem de ser implementado em cima da *DHT Pastry* [23] e poder ser adaptado a qualquer implementação de uma *DHT* uma vez que é independente da *DHT* subjacente.

As soluções apresentadas de encaminhamento com um número constante de passos dependem sempre de um modelo de *churn* para que possam ser equacionadas, sendo esta uma componente fundamental para que se possam tomar decisões sobre a estrutura de um algoritmo

[5]Domain Name System

deste tipo.

3.3 Modelos de Churn

As entradas e saídas são eventos independentes num sistema P2P, a repetição de milhares ou milhões de eventos deste tipo cria um efeito que se chama de *churn* [26]. O *churn* consiste então na dinâmica das alterações de filiação de um sistema, ou seja, as alterações no conjunto de participantes devido às entradas e saídas de utilizadores, sendo este processo de chegada e partida um processo continuo [8, 21].

Em [26] é abordado o problema do *churn* em sistemas P2P, onde se refere que o conhecimento actual sobre modelos de *churn* neste tipo de sistemas é ainda muito escasso, no entanto a informação sobre esta característica de um sistema é fundamental para a sua correcta avaliação e melhoramento, uma vez que há várias decisões de desenho e implementação que podem ser tomadas tendo como suporte um bom modelo de *churn*. [26] tenta portanto perceber qual é a dinâmica dos utilizadores num sistema P2P, dado que na ausência de um modelo de *churn* adequado, os responsáveis pelo desenho de sistemas P2P podem ser forçados a assumir alguns factores sobre a distribuição de tempos de sessão e tempo entre chegadas que podem não estar correctos, levando a que não se observe o desempenho esperado aquando de uma implementação real. Para se poder caracterizar correctamente um modelo de *churn*, é necessário que se disponha de dados úteis na construção desse modelo, o que pode ser por vezes difícil de obter, pois o *churn* depende de vários factores e nem todos os sistemas seguem dinâmicas de filiação semelhantes. Em relação às métricas de *churn*, em [26] é sugerido que seja utilizada uma distribuição de *Weibull* para modelar a distribuição do tempo de sessão dos utilizadores do sistema. A utilização desta distribuição para modelar tal propriedade é também sugerida por outros trabalhos relacionados, que se referenciam mais tarde nesta secção. Em [26] é ainda sugerido que os dados obtidos sobre tempos de sessão passados, são bons indicadores sobre os tempos de sessão futuros, esta sugestão é feita com base em dados obtidos em sistemas P2P já implementados na rede Internet.

A classificação da necessidade de um modelo de *churn* adequado como uma necessidade fundamental é também evidenciada em [8], onde é referido que o efeito de um *churn* elevado leva à perca de mensagens, inconsistência de dados, aumento da latência e ao aumento da utilização de largura de banda, é então necessário que os sistemas sejam avaliados tendo em conta um modelo de *churn* adequado. Em [8] é apresentado um método para diminuir o *churn* observado em sistemas distribuídos P2P. O método apresentado passa por seleccionar os nós a utilizar através de um critério que minimize o *churn* no sistema. No entanto, os resultados apresentados em [8] só são válidos para algumas arquitecturas de sistemas, uma vez que essa selecção não é possível para todos os desenhos existentes. É assumido que o sistema em

causa usa por exemplo uma estrutura baseada em super-nós, sendo os critérios apresentados e discutidos úteis na selecção dos nós que devem desempenhar a função de super-nó. Uma das estratégias sugeridas para efectuar a escolha de nós, consiste em seleccionar nós aleatoriamente para desempenhar as funções pretendidas. Esta solução, apesar de simples, mostra ser uma boa opção ao ser avaliada, além de ser uma solução de baixo custo quando comparada com outras estratégias, como por exemplo uma estratégia em que os nós são escolhidos com base nos seus tempos de sessão no passado.

Em [21] é abordado o problema do *churn* em sistemas estruturados por *DHT* e são sugeridas algumas formas de diminuir os efeitos do *churn* neste tipo de arquitecturas. Os métodos apresentados permitem às *DHT* suportar algum grau de *churn*. Embora não seja seguida uma aproximação com a utilização de tabelas de dispersão distribuídas no caso do LiveFeeds, os métodos utilizados por [21] podem no entanto ser equacionados e adaptados para que se possa construir um sistema que suporte um modelo de *churn* mais dinâmico. As formas apresentadas para suportar esse *churn* passam pelo modo como são encontrados novos vizinhos em caso de falhas, como são calculados os *timeouts* de mensagens de consulta, e como são escolhidos vizinhos que estejam mais perto do ponto de vista da latência entre nós. Dentre todas estas soluções, no contexto do LiveFeeds, a reflexão sobre a escolha de vizinhos e o cálculo de *timeouts* são as soluções mais importantes uma vez que nos resultados podem ser estudados tendo em conta a aproximação utilizada no projecto. É sugerido que se faça o cálculo dos *timeouts* com base no mesmo método usado no protocolo TCP e que se faça a escolha de vizinhos com base num sistema de coordenadas virtuais, caso em que pode ser utilizado o sistema de coordenadas virtuais *Vivaldi*, também utilizado pela *DHT Chord* [25]. Em relação à utilização da escolha de vizinhos com base na sua proximidade, são apresentadas soluções que melhoram bastante a capacidade de suportar o *churn* de um sistema, o que sugere que a utilização de uma aproximação deste tipo pode ser vantajoso no projecto LiveFeeds, onde embora não exista o conceito de vizinhos, pelo menos no sentido tradicional já que todos os nós são vizinhos de todos os nós, existem nós que são seleccionados para desempenhar diversas funções, os resultados em [21] sugerem que o uso de coordenadas virtuais como métrica para seleccionar esses nós pode ser vantajoso. A validação e teste das soluções apresentadas foi feita através da utilização de um sistema de emulação de rede, o Modelnet [27]. A utilização de coordenadas virtuais para melhoramento do tempo de encaminhamento de mensagens no LiveFeeds é objecto de outra dissertação de mestrado [4].

Como obter um modelo de *churn*

Foram estudados vários trabalhos que se debruçam sobre os modelos de entrada e tempo de permanência dos utilizadores neste tipo de sistemas.

Em primeiro lugar foi estudado [9], este trabalho examina a rede *skype*, e é tanto quanto sabemos, o único estudo que fornece alguma informação sobre o comportamento dos utilizadores nesta rede.

Este estudo, refere que o comportamento dos utilizadores da rede *skype* não é semelhante ao dos utilizadores das redes de partilha de ficheiros anteriormente estudadas (*Gnutella* etc.), uma vez que a entrada e saída de utilizadores depende fortemente da altura do dia e da altura da semana, havendo mais entradas durante a manhã, mais saídas ao fim do dia e mais utilização da rede em dias úteis. Em [10], foi utilizado como base um valor para a média de tempo de sessão de cerca de 2.9 horas, baseada em estudos sobre a rede *Gnutella*, uma média de tempo de sessão com este valor não se verifica em redes como a rede *skype*.

Apesar de em [9] se ter focado principalmente o comportamento dos super-nós da rede, pensamos que as considerações acerca do comportamento dos utilizadores se podem, provavelmente, adaptar-se aos utilizadores normais da rede.

A ideia de que a entrada e saída de utilizadores depende da altura do dia e da semana que estamos a considerar, é também suportada por [29]. Apesar de este estudo apenas considerar o tráfego de uma grande empresa numa rede de *instant messaging*, os resultados referentes aos modelos de entrada e saída de utilizadores e modelo de tempo de permanência da rede, podem ser adaptados de forma a serem utilizados na construção de um modelo de *churn* a utilizar no estudo da rede LiveFeeds.

Uma das formas de se obter um modelo de *churn* é modelar as chegadas de utilizadores com uma distribuição de *Poisson* não homogénea, uma vez que a quantidade de entradas depende das alturas do dia e da semana, e modelar o tempo de permanência desses utilizadores com uma distribuição *heavy-tailed* [9].

Outros trabalhos podem ser considerados para que se obtenham um bom modelo de *churn* para o projecto LiveFeeds. Em [14] é demonstrado que os eventos referentes a uma actividade como a comunicação por *e-mail* podem ser modelados recorrendo a uma distribuição de *Poisson* não homogénea. Este tipo de utilização, tal como o tipo de utilização por parte dos utilizadores da rede *skype* pode ser comparável aos comportamentos previstos dos possíveis utilizadores do LiveFeeds. Como é afirmado mesmo pelos autores, apesar do modelo ter sido desenhado considerando apenas uma actividade como a correspondência por *e-mail*, os resultados obtidos podem ser adaptados para outros sistemas com características semelhantes. De facto, em [14] é de novo evidenciado que o comportamento dos utilizadores depende da altura do dia e da semana. Os resultados de [14], em que é mostrado como podem ser parametrizadas as distribuições para que modelem correctamente o comportamento dos utilizadores, são úteis na construção de um modelo de *churn* adequado, e corroboram os resultados inferidos através de [29, 9].

3.4 Métodos de Avaliação

Durante fase de desenho e implementação de um sistema distribuído é normalmente necessário avaliar e afinar o sistema desenvolvido. Para que esses melhoramentos possam ser efectuados e para avaliar o desempenho do novo sistema é geralmente necessário usar testes empíricos para que se possam chegar a melhores resultados e fazer uma avaliação mais rigorosa do sistema.

Mesmo que por vezes os testes aos sistemas desenvolvidos possam ser efectuados na *Internet*, normalmente esta não é uma maneira prática de efectuar testes ou avaliações de um sistema, uma vez que as condições da rede não são controláveis pelos responsáveis pelo sistema e é normalmente difícil de obter dados sobre o sistema devido à sua natureza distribuída.

Para se poder contornar as dificuldades inerentes ao teste e avaliação de sistemas e algoritmos distribuídos directamente na rede *Internet*, pode-se proceder a testes experimentais através de simulação ou emulação.

Um simulador de rede é um *framework*[6] que permite a simulação de uma rede, algoritmo ou sistema distribuído. Normalmente a execução de um simulador é feita numa estação de trabalho onde são simuladas as características de toda uma rede e de um conjunto de nós da rede que executam determinados algoritmos ou funções que formam o sistema. No entanto, os simuladores sacrificam normalmente alguma exactidão em troca de escalabilidade, abstraindo algumas propriedades importantes da rede que se pretende simular. Apesar disso e uma vez que os simuladores são executados normalmente em apenas um computador, estão limitados pelo *hardware* utilizado e suportam a simulação apenas de um número de nós reduzido. Para que se possa simular uma rede com um número de nós na ordem dos milhares é necessário dispensar de alguma exactidão, novamente através da abstracção de algumas propriedades da rede real.

Um emulador de rede consiste numa ferramenta distribuída composta por vários computadores e equipamentos de rede de forma a emular uma rede real, submetendo o tráfego aos constrangimentos de largura de banda de ligações entre nós, latência e perca de pacotes de uma topologia especificada pelos investigadores [27]. Os sistemas ou algoritmos distribuídos podem assim ser testados num ambiente mais próximo da rede *Internet* uma vez que os emuladores de rede emulam propriedades da rede que são normalmente abstraídas em testes experimentais efectuados através de simulação. Tal como os simuladores, os emuladores de rede usam geradores de topologias para obterem topologias de rede que se possam utilizar para aproximar a rede emulada da rede real.

A emulação é normalmente feita com recurso a vários computadores, o que geralmente não acontece com a simulação. Este facto permite que durante as simulações sejam abstraídas diversas especificidades da utilização de toda a pilha de protocolos de rede, o que não acontece durante a emulação de sistemas distribuídos, pois ao utilizar um *cluster* de computadores para emular uma certa topologia, está-se necessariamente a lidar com comunicações em rede

[6]Pode ser traduzido para português como plataforma

incluindo todas as suas características, o que pode permitir uma maior exactidão do que a simulação de uma rede. O desafio primário em ambientes de emulação é a escala da emulação.

Dado que a emulação é suportada por *hardware* e o sistema emulado é maior do que o sistema físico subjacente, é necessário que haja uma forma cuidada e inteligente de alocação de recursos de *hardware* [12]. A escala de uma emulação não é definida apenas pelo número de nós que uma emulação suporta mas também pela exactidão e facilidade de uso com que são realizados os testes empíricos. A transparência para as aplicações em teste é também um factor importante, idealmente, uma aplicação não deve sofrer alterações para que seja instalada em ambiente de emulação. A emulação é um processo mais vasto e complexo do que apenas virtualizar sistemas operativos e ligações de rede, inclui a resolução de vários problemas como a alocação de endereços IP e de recursos de *hardware* [12]. Para preparar este estudo foi instalado e configurado o emulador de rede ModelNet (cf. A).

Os sistemas desenvolvidos podem também ser testados na rede *Internet* através de plataformas como o *PlanetLab*[7]. O *PlanetLab* é uma rede de investigação que suporta o desenvolvimento de serviços de rede. Utilizando este sistema, os investigadores podem testar as suas aplicações utilizando como rede subjacente a rede *Internet*. As aplicações são assim testadas utilizando uma maior exactidão do que uma rede emulada em laboratório uma vez que os vários nós do sistema estão de facto distribuídos por toda a *Internet*. Esta plataforma consiste num sistema de emulação que está distribuído em diversas instituições espalhadas por todo o mundo e disponibilizam recursos de *hardware* onde as aplicações em teste podem ser executadas. Este tipo de aproximação levanta no entanto várias dificuldades, não só em relação à sua escalabilidade e facilidade de utilização mas também no domínio da alocação de recursos e na complexidade que existe ao obter dados sobre as experiências executadas, uma vez que todo sistema se encontra geograficamente distribuído.

Através da observação das características das aproximações referidas, simulação, emulação e teste em rede real, podemos concluir que é necessário abdicar de alguma simplicidade em troca de uma maior aproximação à realidade, a emulação é de facto mais próxima da realidade do que uma simulação, mas é necessário efectuar trabalho de instalação e configuração para se poder utilizar um sistema fiável de emulação. As dificuldades são também maiores quando se passa de uma emulação em laboratório para um teste experimental na rede *Internet*. Este *trade-off* entre complexidade e exactidão, observável entre os vários métodos de avaliação, está representado na figura 3.1.

A simulação é então normalmente o primeiro passo para avaliar empiricamente um sistema distribuído, enquanto a emulação surge naturalmente como o passo seguinte, antes de se testar o sistema numa melhor aproximação à *Internet* como é o caso do *PlanetLab*.

Para avaliar o sistema do projecto LiveFeeds, existe já um simulador em desenvolvimento que procura simular os algoritmos implementados de maneira a avaliar e testar o sistema para

[7] http://www.planet-lab.org/

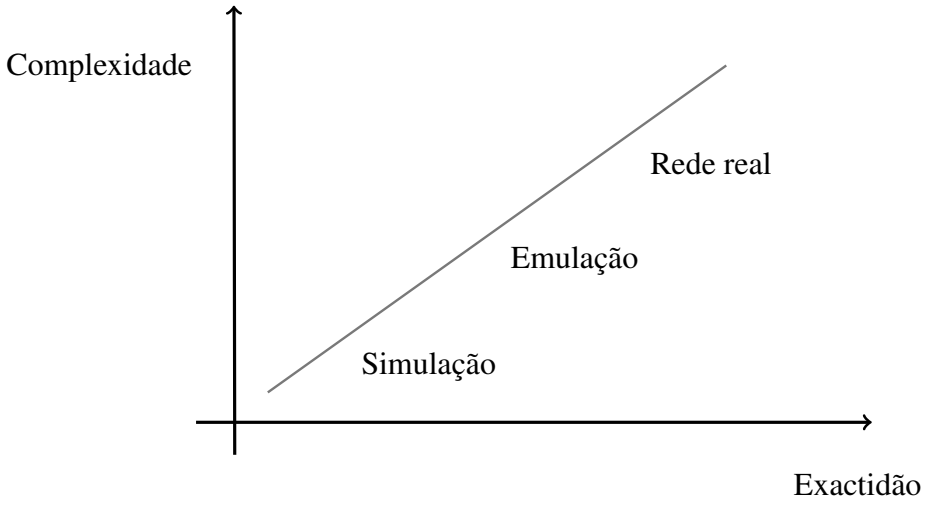


Figura 3.1: *Trade-off* entre exactidão e complexidade do método de avaliação

que se possam efectuar melhoramentos. O passo seguinte na avaliação e desenvolvimento do LiveFeeds é portanto a emulação em ambiente de laboratório. Para que isso seja possível, foi escolhido o emulador de rede ModelNet [27], este emulador permite que um sistema seja testado sem que seja necessário fazer alterações nos protótipos da aplicação já desenvolvida e emula uma rede com uma topologia que pode ser especificada pelos responsáveis pelos testes.

O emulador ModelNet trabalha utilizando um conjunto de computadores que desempenham tarefas de emulação. Um subconjunto de computadores executa o *software* em avaliação, que é executado sem alterações. Quando este *software* pretende enviar pacotes para outro nó que esteja a ser emulado, os pacotes são interceptados enviados para um outro subconjunto de computadores que emula uma topologia de rede, os *core nodes*[8], atrasando ou adiantando os pacotes recebidos e de seguida enviando-os para o nó virtual de destino, que pode até estar a ser executado no mesmo computador que o nó virtual que enviou o pacote.

Desta forma, uma emulação utilizando o ModelNet pode emular até milhares de nós de uma rede com elevada fiabilidade e com recursos de *hardware* de custo aceitável, devido à forma como foi desenhado este sistema. Uma vez que a topologia da rede utilizada pelos *core nodes* do ModelNet para emular o sistema é definida na fase de configuração do emulador, os testes efectuados podem ser altamente configuráveis de acordo com as pretensões dos investigadores. Uma vez que todo o tráfego é reencaminhado pelos *core nodes* os dados sobre o sistema em avaliação estão centralizados em apenas alguns computadores o que facilita a tarefa de análise do sistema em avaliação.

Em [12] é proposta outra forma de emular uma rede para execução de testes empíricos em sistemas distribuídos, o Emulab. Neste sistema é utilizada virtualização para que se consiga um grau de transparência para as aplicações que são executadas sem serem alteradas e são postas

[8]Pode ser traduzido para português como "nós do núcleo"

em prática várias formas de multiplexagem de recursos de *hardware* para que se consiga uma elevada escalabilidade do sistema de emulação. O *software* Emulab é um sistema de gestão para um conjunto de computadores que oferece uma forma de partilha de recursos para o estudo e teste experimental de sistemas distribuídos. Um dos objectivos deste sistema é oferecer a integração transparente de diversas experiências que são executadas ao mesmo tempo dentro do mesmo *cluster*[9] de computadores. Um investigador pode submeter uma topologia que será emulada pelo Emulab de forma independente, juntamente com outras topologia anteriormente submetidas por outros investigadores. Um topologia pode consistir num conjunto de nós e ligações e características como largura de banda, latência ou perca de pacotes. Podem também ser especificadas características de *hardware* para os diversos nós da topologia. Após a topologia ter sido submetida, os recursos necessários para a sua emulação são alocados pelo *software* de gestão Emulab e as experiências podem então ser iniciadas sobre essa topologia.

O ModelNet consegue chegar a uma alta escalabilidade através da abstracção de alguns detalhes no interior da topologia da rede. Ao instalar uma nova topologia no ModelNet, são calculados todos os caminhos possíveis entre nós e é feita uma estimativa da capacidade agregada das ligações para cada um desses caminhos. Desta forma, mesmo que uma ligação entre nós seja definida na topologia como um caminho de vários passos, para o ModelNet esse caminho é emulado como uma ligação com as características agregadas de todas as ligações que a definem. São assim abstraídas várias características importantes de um caminho de vários passos como o comportamento de protocolos de *routing*, perca de pacotes, etc, o que inibe o uso de algumas aplicações, como o *traceroute*, o que não acontece com o Emulab pois são emuladas correctamente todos os detalhes dos protocolos de routing utilizados. O ModelNet usa uma noção fraca de máquina virtual para cada nó virtual, não é oferecida uma virtualização do sistema de ficheiros e da banda disponível, sendo então diminuído o isolamento entre os nós virtuais que são executados no mesmo computador, ao contrário do que acontece com o Emulab em que cada nó dispõe da sua máquina virtual. Os dois sistemas de emulação são no entanto complementares uma vez a escalabilidade do ModelNet é uma importante vantagem e o Emulab é um sistema executado de forma remota e partilhada e não pode ser instalado em laboratório [12].

Embora a rede *Internet* não possa ser emulada ou simulada com uma exactidão perfeita, dado que que depende de inúmeros factores que não podem ser controlados, os emuladores e simuladores constituem ferramentas fundamentais na avaliação e afinação de algoritmos ou sistemas distribuídos permitindo uma aproximação o mais fiel possível em relação à rede que se pretende emular ou simular.

[9]Pode ser traduzido para português como agrupamento

3.5 Conclusão

Neste capítulo foram assim apresentados alguns trabalhos relacionados que foi necessário estudar para se fazer uma avaliação rigorosa e melhorar o algoritmo de filiação do projecto Live-Feeds.

O levantamento bibliográfico dos trabalhos relacionados com encaminhamento em sistemas editor/subscritor com uma arquitectura P2P e com encaminhamento com um número constante de passos ofereceram uma base teórica para a avaliação do algoritmo e para a ante-visão de possíveis melhoramentos.

Os trabalhos sobre o modelo de *churn* a utilizar permitiram que se construísse um modelo de *churn* realista com base no comportamento dos utilizadores em redes P2P como a rede *skype* ou redes de *instant-messaging*.

Em relação aos métodos de avaliação de sistema distribuídos, foi possível concluir que após ter sido feita uma análise teórica ao algoritmo de filiação, o próximo passo seria avaliar o comportamento do algoritmo em ambiente de simulação.

4. Análise do custo da difusão usando árvores aleatórias

Para podermos avaliar a exequibilidade do mecanismo de difusão descrito anteriormente, torna-se necessário que se quantifique os custos inerentes a este tipo de sistema.

O algoritmo aqui analisado consiste no seguinte: um nó a ao entrar no sistema contacta um outro nó b, seleccionado aleatoriamente, que fica encarregado de enviar a tabela de utilizadores ao nó a. Depois de receber a tabela, a contacta o *slice leader* correspondente à sua fatia e notifica-o da sua entrada. Um *slice leader* está encarregado de receber as várias notificações de entrada que ocorrem na sua fatia e agrupa essas notificações durante algum tempo, t_a. Findo esse tempo, selecciona um nó aleatoriamente, c, para ser a raiz da árvore de difusão das notificações que recebeu durante os últimos t_a segundos. O nó c dá então início à construção de uma árvore de difusão com grau G, processo que terá lugar simultaneamente com a própria difusão, como descrito no capítulo anterior.

Na análise que se segue pretende-se estabelecer o custo da realização da difusão das mensagens através de árvores aleatórias pressupondo que todos os nós têm já uma visão conjunta consistente da filiação.

A análise a seguir apresentada baseia-se em 3 hipóteses distintas:

- As árvores de difusão formadas pelo algoritmo de difusão são sempre completas e equilibradas, todos os nós interiores têm G filhos;
- O ritmo de novas entradas é constante havendo um certo número de entradas por segundo (r);
- O tempo de sessão de todos os nós é infinito, não existindo saídas nem falhas de nós.

4.1 Parâmetros da análise

Os seguintes parâmetros são utilizados para descrever o sistema em análise:

- N número de utilizadores do sistema
- G grau da árvore de difusão
- r taxa de eventos de entrada que acontecem no sistema, em eventos por segundo admitindo um ritmo constante de eventos
- t_a tempo durante o qual um *slice leader* agrupa notificações recebidas antes de nomear um nó para construir uma árvore de difusão
- k Número de *slices* do sistema

- m Tamanho da estrutura que representa um evento de entrada ou saída, depois de aplicada uma compressão de dados. O tamanho desta estrutura inclui o tamanho necessário para descrever uma lista de subscrições de um utilizador.

- l Tamanho da tabela de *routing do sistema*

Podem ainda ser definidos os seguintes valores a partir dos parâmetros acima descritos.

- m_t Tamanho total de uma mensagem de notificação que contem informação sobre e eventos de entrada/saída:

$$m_t = e.m \tag{4.1}$$

Seguidamente é analisado o custo, em termos de capacidade de rede, da difusão utilizando árvores aleatórias, com base nas hipóteses já estabelecidas e com os parâmetros apresentados.

4.2 Probabilidade de um nó ser folha

Como veremos na secção 4.3, o tráfego de um nó depende da sua posição na árvore de difusão e portanto torna-se necessário obter uma expressão para o valor da probabilidade de um nó ser nó interior ou folha da árvore de difusão.

Admitamos que a árvore de difusão construída é uma árvore equilibrada e completa, isto é, uma árvore em que cada nó tem 0 ou G filhos, o que corresponde a uma simplificação do sistema real. Com N nós, grau G e altura h, o número de nós dessa árvore é dado por:

$$N = \sum_{i=0}^{h} G^i = \frac{G^{h+1}-1}{G-1} \tag{4.2}$$

O número de folhas da árvore é dado por:

$$f = G^h \tag{4.3}$$

Então a probabilidade de um nó ser folha é dada por:

$$\frac{f}{n} = \frac{G^h(G-1)}{G^{h+1}-1} \tag{4.4}$$

$$= \frac{G^{h+1}-G^h}{G^{h+1}-1} \tag{4.5}$$

$$\approx \frac{G^{h+1}-G^h}{G^{h+1}} = 1-\frac{1}{G} \tag{4.6}$$

$$P(folha) = 1-\frac{1}{G} \tag{4.7}$$

A probabilidade de um nó ser escolhido para nó interior, $P(interior)$, é:

$$P(interior) = 1 - P(folha) \tag{4.8}$$

$$P(interior) = \frac{1}{G} \tag{4.9}$$

As expressões (4.10) e (4.11) serão úteis nesta análise:

$$\lim_{G \to \infty} P(interior) = 0 \tag{4.10}$$

$$\left(\frac{1}{G}\right)' = -\frac{1}{G^2}$$
$$\left(\frac{1}{G}\right)' < 0, \forall G \in \mathbb{N} \tag{4.11}$$

Isto é, a probabilidade de um nó ser interior na árvore decai com o aumento do grau utilizado, como se pode observar pela figura 4.1.

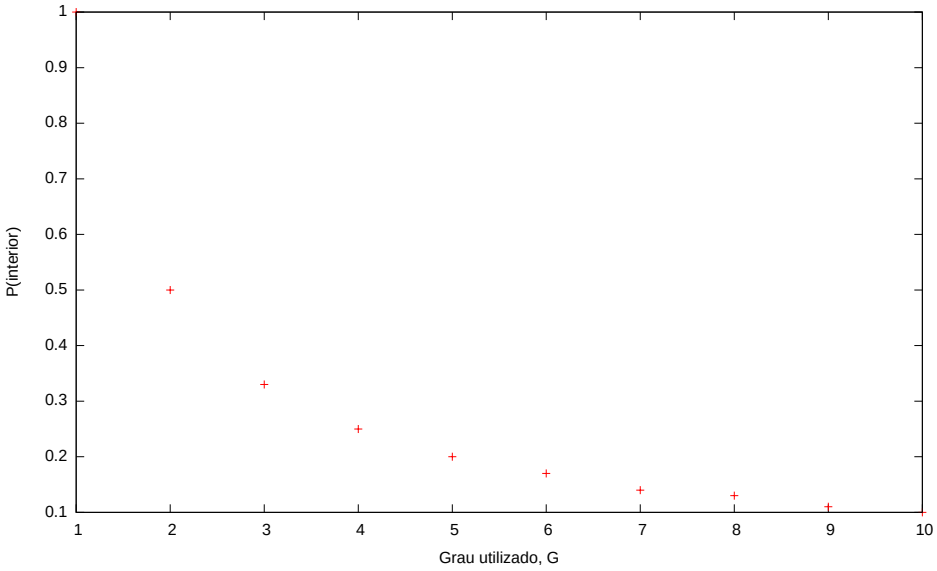


Figura 4.1: Valor de $P(interior)$ em função do grau G utilizado

4.3 Tráfego por nó

O tráfego de cada nó, aquando da recepção de uma mensagem de notificação contendo e eventos, depende da posição do nó na árvore de difusão.

As folhas da árvore contribuem com menos tráfego do que os nós interiores, um vez que só têm de receber a mensagem de notificação e não é necessário que essa mensagem seja reencaminhada para G nós, como acontece com os nós interiores. Um nó interior recebe m_t e envia $G \times m_t$, enquanto uma folha não envia mensagens e recebe m_t.

O tráfego médio de *upstream* por cada árvore de difusão a ser construída é dado por:

Por (4.8):

$$w_u = P(interior) \times G.m_t \tag{4.12}$$

$$w_u = \frac{1}{G} \times G.m_t \tag{4.13}$$

$$w_u = m_t \tag{4.14}$$

O tráfego médio de *downstream* em cada nó de cada uma dessas árvores é:

$$w_d = m_t \tag{4.15}$$

w_d não depende do posicionamento do nó na árvore de difusão uma vez que os nós interiores e os nós folha recebem o mesmo tráfego de *downstream*.

Admitindo que todos os *slice leaders* recebem pelo menos um evento a cada t_a segundos, um nova árvore de difusão é formada $\frac{k}{t_a}$ vezes por segundo. Então, a largura de banda média de *upstream*, b_u, é dada por:

$$\begin{aligned} b_u &= \frac{k}{t_a} \times \frac{r}{k}.t_a.m \\ b_u &= r.m \end{aligned} \tag{4.16}$$

Enquanto a largura de banda média de *downstream* é dada por:

$$\begin{aligned} b_d &= \frac{k}{t_a} \times m_t \\ b_d &= \frac{k}{t_a} \times \frac{r}{k}.t_a.m \\ b_d &= r.m \end{aligned} \tag{4.17}$$

Das expressões (4.16) e (4.17) podemos concluir que em média, todos os nós do sistema considerado contribuem com o mesmo tráfego de *upstream* e *downstream* uma vez que $b_u = b_d$.

Além disso, o nó seleccionado para enviar a tabela de *routing* ao novo nó, tem de enviar l bytes ao novo nó.

Os *slice leaders* estão sujeitos a uma maior carga, uma vez que têm de despender adicionalmente de uma largura de banda de *upstream* e *downstream* de:

$$\frac{r}{k} \times m \tag{4.18}$$

Que será tanto menor quanto maior o número de *slice leaders* (k).

4.4 Estimativa do valor de r

Para se tirar algumas conclusões a partir dos resultados obtidos de forma analítica é necessário estimar um valor de r plausível. De modo a obter uma simplificação do modelo considerado, admitimos que o sistema tem um número constante de nós e que os utilizadores têm uma duração de sessão idêntica constante ao longo do tempo. Portanto, ao sair um utilizador, existe uma correspondente entrada de outro utilizador que fica durante o mesmo tempo a utilizar o sistema. Desta forma, podemos considerar que o tempo de sessão, t_{sessao}, suportado pelo sistema é dado pela equação 4.19.

$$t_{sessao} = \frac{N}{r} \tag{4.19}$$

4.5 Exemplo dos custos

Para que seja possível termos uma ideia dos valores obtidos através destes cálculos, podemos escolher alguns valores para os parâmetros referidos na secção anterior e apresentar os valores obtidos.

Para os parâmetros já descritos, foram utilizados os valores representados na tabela 5.1. É utilizado $N = 59048$ para que a árvore formada possa ser completa e equilibrada e um $r = 15$ Eventos/Seg, que aplicando (4.19) equivale a um tempo de sessão médio de cerca de 3,9 horas.

Parâmetro	Valor
N	59048 Nós
G	3
r	15 Eventos/Seg.
t_a	10 Segundos
k	3 Slices
m	533 B
l	29,56 MB

Tabela 4.1: Valores dos parâmetros utilizados na análise do algoritmo utilizando árvores aleatórias

Com os valores da tabela 4.1 e as expressões obtidas na secção 4.3, obtém-se o valor de 62,46 Kbps para a largura de banda média de *upstream* e *downstream* (expressões (4.16) e (4.17)).

Por outro lado, um nó que seja seleccionado para enviar a tabela de *routing* ao novo nó, deve enviar pelo menos 29,96 MB ao mesmo. Um *slice leader* tem ainda de suportar um custo adicional de 20,82 Kbps de *upstream* e *downstream*.

Com os parâmetros definidos, obtemos $P(folha) \approx 0,67$ ou seja, cerca de 67% das vezes em que é formada uma árvore de difusão, um nó é folha e tem portanto de suportar menos tráfego de *upstream* uma vez que não precisa de enviar a mensagem de notificação recebida a G nós. A probabilidade de um nó ser nó interior é neste caso $P(interior) \approx 0,33$ ou seja, cerca de 33% das vezes em que é construída uma árvore de difusão, um nó é nó interior e deve então contribuir com mais tráfego de *upstream*.

4.5.1 Cálculo inverso

Nesta secção é analisado o problema de maneira inversa, obtemos aqui um valor para r, dado um valor de banda larga de *upstream*, b_u, através da mudança do parâmetro m do sistema. Assim obtemos o andamento do valor de r que um sistema deste tipo pode suportar.

Por (4.16), obtemos:

$$r = \frac{b_u}{m} \tag{4.20}$$

Através da expressão (4.20) podemos construir o gráfico da figura 4.2, útil na análise do valor de r suportado por um sistema.

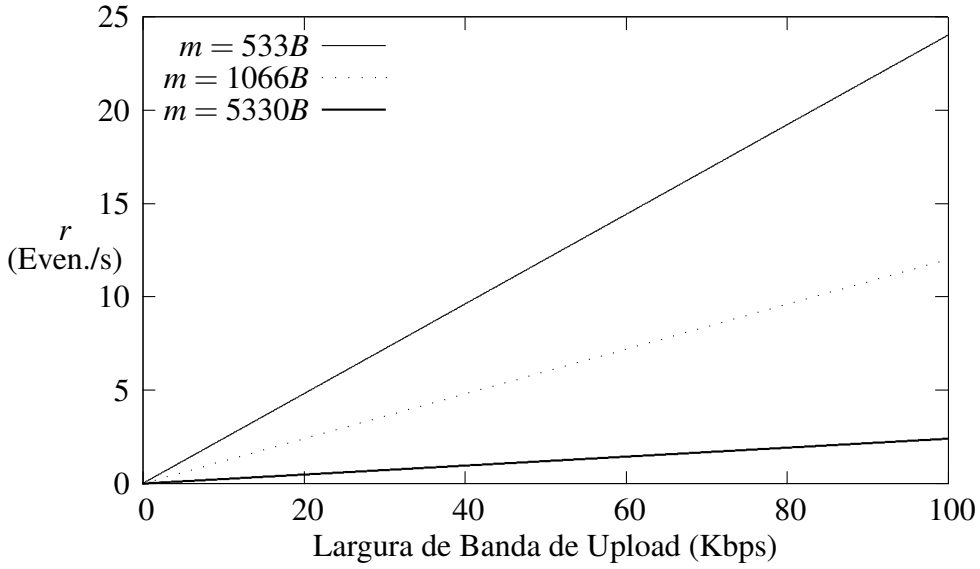


Figura 4.2: Gráfico da expressão (4.20) para vários valores de m

Para $b_u = 25,6$ kbps e utilizando os parâmetros da tabela 4.1 da secção 4.1, isto é, $m = 533B$, obtemos o valor de 6,15 eventos por segundo para r. O que, segundo a expressão (4.19), corresponde a um tempo de sessão de cerca de 5 horas num sistema com 59048 nós, ou seja, para que este tipo de aproximação possa ser viável, os utilizadores devem permanecer no sistema em média cerca de 5 horas.

Através da utilização das expressões (4.20) e (4.19), podemos desenhar o gráfico 4.3 que representa o tempo de sessão que os utilizadores devem permanecer no sistema em função da largura de banda de *upload* com que estão dispostos a contribuir para vários valores de N. No desenho do gráfico 4.3 foi considerado $m = 533B$.

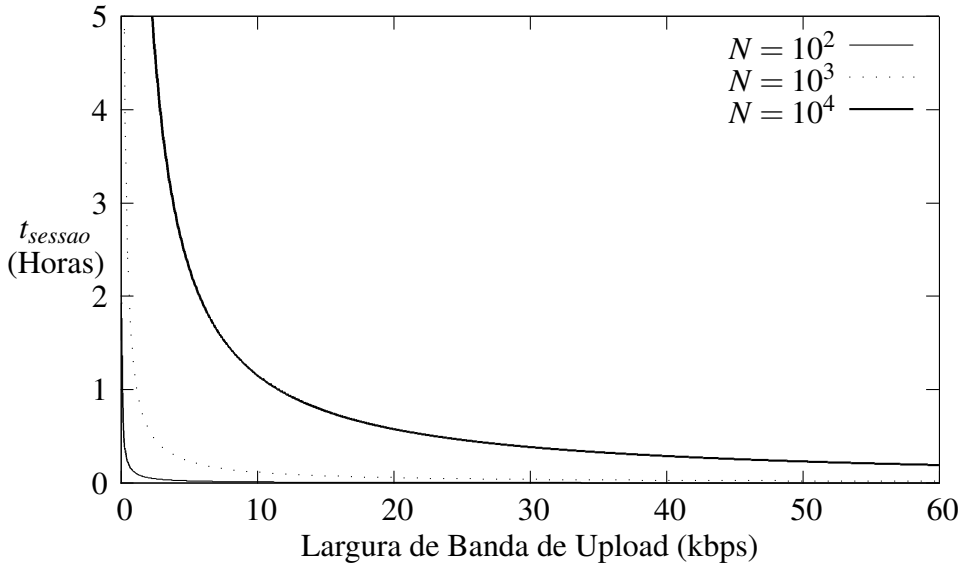


Figura 4.3: Gráfico para t_{sessao} máximo em função de b_u para vários valores de N.

4.6 Simulações sem *churn* e sem falhas

A análise anterior foi realizada com base numa restrição sobre os valores possíveis do número de nós do sistema (cf. a equação 4.2) e pressupondo que a árvore de difusão é completa e equilibrada. Foi necessário verificar, através de simulação, se as aproximações feitas seriam demasiado simplistas e se prejudicavam a análise feita ao algoritmo de filiação. Para o efeito, foram realizadas um conjunto de simulações que permitissem aferir se os resultados obtidos nas secções anteriores continuam a ser válidos quando o sistema não tem um número de nós como o examinado e quando a topologia das árvores construídas não é exactamente a mesma.

As simulações consistiram na introdução de N nós no sistema com um tempo de sessão infinito e sem que existam falhas dos nós. Note-se que o valor de N utilizado não satisfaz a equação (4.2). Após o sistema ter estabilizado, é escolhido um nó aleatoriamente, r vezes por

segundo, para iniciar o *broadcast* de uma mensagem. Os gráficos apresentados na figura 4.4 correspondem a uma das experiências realizadas.

Como se pode observar, o tráfego de *downstream* é representado por uma linha horizontal, o valor médio do tráfego de *downstream* é o mesmo para todos os nós e igual a 500 B/s. Recorde-se que fora concluído que $b_d = b_u = r \times m$ e $b_d = 1 \times 500 = 500B/s$, ou seja, o valor de b_d observado nesta simulação é o mesmo do que o esperado após a análise feita na secção 4.3.

Em relação ao tráfego de *upstream*, pode-se confirmar que os valores no início da simulação são diferentes entre os vários nós, mas com o aumento do número de árvores de difusão construídas, uma por cada *broadcast*, o valor do tráfego de *upstream* vai convergindo para um valor comum, para $b_u = 500B/s$ exactamente o valor correspondente ao tráfego de *downstream*, b_d, tal como tinha sido concluído na secção 4.3, ao fim de um grande número de *broadcasts*, o valor da média do tráfego de *upstream* de todos os nós converge para o mesmo valor da média do tráfego de *downstream*. Através destas simulações foi possível de verificar que apesar de não estarmos nas condições idealizadas em 4.6, os resultados obtidos constituem uma boa aproximação quando se simula o algoritmo de difusão do LiveFeeds.

Para $N = 2000$ Nós, foram necessários 3600 segundos para se observar a convergência representada na figura 4.4 (c), dado que é formada uma árvore de difusão a cada segundo, foram precisas apenas 3600 árvores de difusão para que se verificasse a igualdade $b_d = b_u$.

As simulações com vários valores de N confirmaram que o resultado anterior é válido mesmo para sistemas de vários tamanhos apesar de a árvore formada não ter a topologia anteriormente considerada, repare-se que, dado o algoritmo usado para construir as árvores, estas não são necessariamente equilibradas nem completas.

Os resultados destas simulações confirmam que mesmo quando o tamanho do sistema não é o mesmo do que o anteriormente considerado, tal como a topologia das árvores de difusão construídas e na ausência de falhas, os resultados obtidos no que se refere à capacidade de *upstream* e *downstream* que cada nó deve despender continuam a ser uma boa aproximação.

4.7 Conclusões

A partir dos valores e expressões obtidos podemos tirar algumas conclusões.

Considerando o facto de a probabilidade de um nó ser folha ser maior do que a probabilidade de um nó ser nó interior aquando da formação de uma nova árvore de difusão, em conjunto com a igualdade entre as larguras de banda de *upstream* e *downstream* médias, podemos concluir que as vezes em que um nó é escolhido para nó interior e tem de contribuir com mais tráfego são compensadas pelo número de vezes em que um nó é escolhido para ser folha e tem de cooperar com menos tráfego.

Para os parâmetros considerados, os valores dos custos médios de banda passante são bastante altos. Por exemplo, considerando que um utilizador está disposto a despender 10% dos seus 256 kbps de banda passante de *upstream*, $b_u = 25.6$ kbps, o sistema apenas suporta cerca de $r = 6.15$ eventos por segundo, o que é pouco para um universo de utilizadores de cerca de 10^5 utilizadores.

Considerando o facto de $P(interior)$ depender apenas de G, e tendo em conta o limite (4.10) e a expressão (4.11), podemos concluir que quanto maior for o grau da árvore de difusão, menor é a probabilidade de um nó ser escolhido para ser nó interior e portanto ter de contribuir com mais tráfego.

É ainda necessário ter em consideração que o modelo considerado não é completo, uma vez que não tem em conta alguns aspectos, tais como os *acknowledgements* de mensagens recebidas e o estabelecimento de conexões TCP.

Os custos aqui obtidos representam apenas o esforço necessário para que seja possível implementar o algoritmo de filiação pretendido e não são considerados custos adicionais para outros algoritmos de que o sistema dependa, o algoritmo de filiação é apenas *overhead* do ponto de vista do utilizador do sistema.

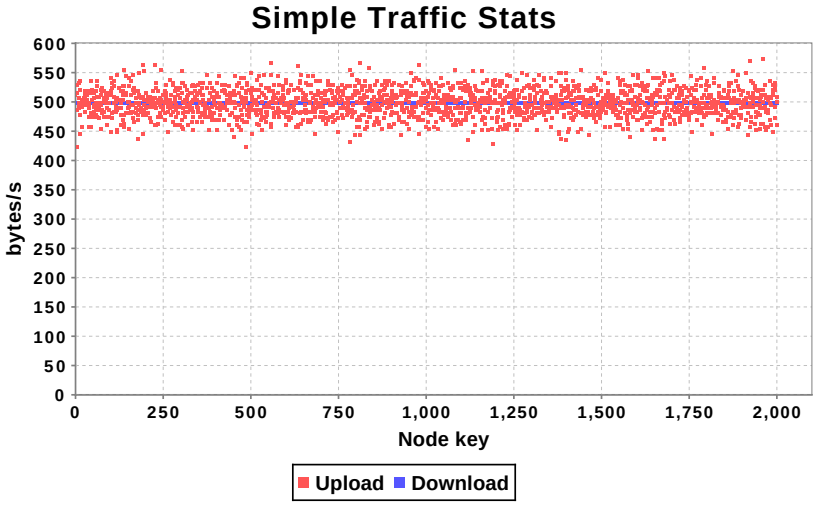
Não foi também tido em consideração que o valor de r num sistema real nunca será linear, isto é, varia de acordo com várias situações, como a hora do dia ou a altura da semana. Por consequência, os valores das expressões calculadas a partir destes parâmetros podem sofrer bastantes alterações numa implementação real do sistema, como será analisado no capítulo 5.

O modelo que acabámos de apresentar pressupõe que o algoritmo apenas forma árvores equilibradas e completas, na hipótese de que os identificadores se distribuem uniformemente, e que o número de nós no sistema é sempre da forma $N = \sum_{i=0}^{h} G^i$.

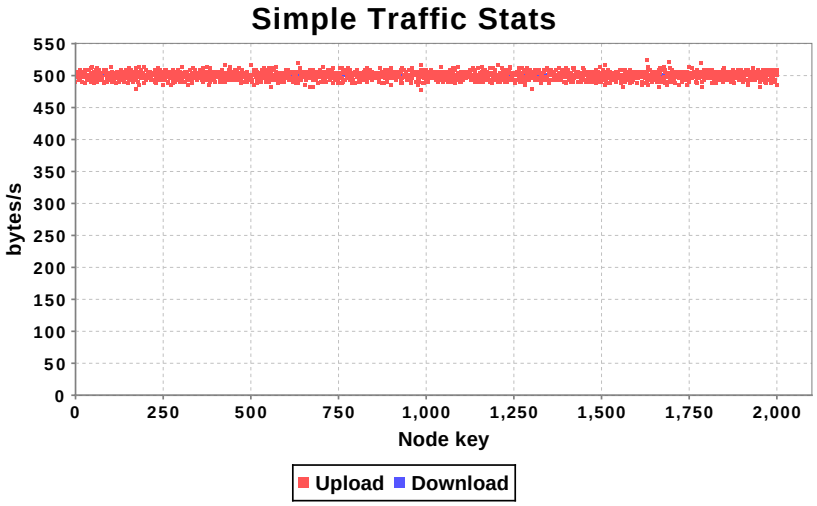
Finalmente, o modelo usado nesta análise não considera as entradas e saídas que estão a decorrer em paralelo com as difusões, nem a possibilidade de ocorrência de falhas.

Na prática nem o número de nós é necessariamente daquela forma nem a árvore é necessariamente equilibradas, pelo que a distribuição da probabilidade de cada nó ser folha ou nó interior é diferente da calculada e a distribuição de carga poderá não ser tão perfeita visto que outros nós tenderão a ter uma probabilidade superior de serem nós interiores e necessitarão de mais capacidade de *upstream*.

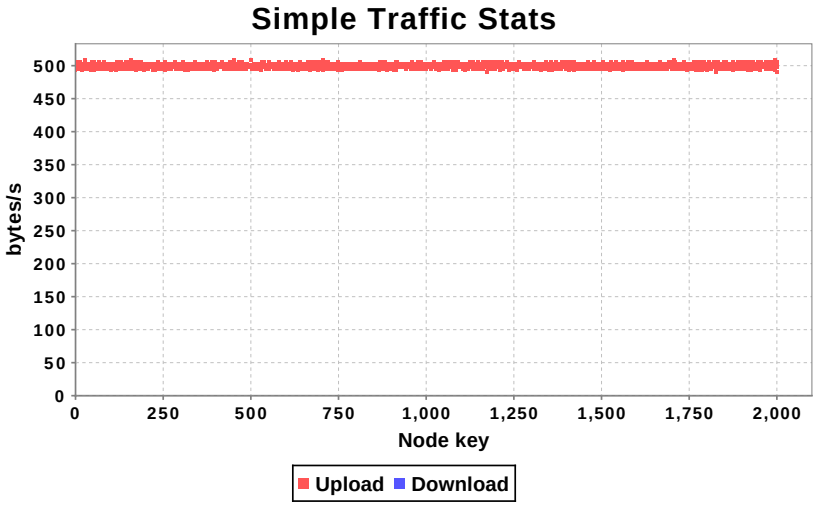
No entanto, como foi possível comprovar através de simulação, a análise feita oferece um bom ponto de partida para a análise do sistema. As simulações sem *churn* realizadas indicam que mesmo que não se verifiquem as hipóteses postas durante a análise teórica ao algoritmo, as expressões obtidas para o calculo do custo em termos de capacidade de rede são uma boa aproximação, nomeadamente as expressões $b_u = b_d$ e $b_u = r \times m$.



(a) $t = 20$ segundos



(b) $t = 1800$ segundos



(c) $t = 3600$ segundos

Figura 4.4: Tráfego médio usado por um sistema de 2000 nós, no momento $t = 20$ segundos (a), no momento $t = 1800$ segundos (b) e no momento $t = 3600$ segundos (c), quando estão a difundir mensagens de dimensão $m_t = 500$B ao ritmo $r = 1$ mensagens por segundo

5 . Custo da difusão da filiação num sistema dinâmico

Numa arquitectura *peer-to-peer*, os nós entram e saem do sistema consoante o comportamento dos utilizadores. Ao utilizar um algoritmo de visibilidade completa, todos estes eventos devem repercutir-se em modificações da tabela de filiação de todos os nós. Nesta análise do sistema as saídas não são considerados como eventos que afectam a taxa de eventos observados, como foi referido em 2. No capítulo anterior foram apresentadas análises e o resultado de algumas simulações que não levavam em linha de conta o dinamismo na filiação do sistema.

Neste capítulo é introduzido um modelo *churn* para que se possam realizar simulações mais próximas da realidade do que as simulações anteriores. Este modelo de *churn* foi desenvolvido tendo como base o trabalho relacionado já referido em 3.3 e é apresentado na secção 5.1.

5.1 O Modelo de churn utilizado

Durante o levantamento de trabalho relacionado sobre modelos de *churn* (secção 3.3), foram apresentadas algumas formas para a construção de um modelo de *churn*, em particular, um modelo de *churn* que se aproxime o mais possível da realidade do comportamento dos possíveis utilizadores do LiveFeeds. Seguindo os passos anteriormente estabelecidos no que se refere à construção de modelos de *churn*, podemos construir um modelo de *churn* com as características pretendidas. Neste contexto, considera-se que os utilizadores têm um comportamento semelhante aos utilizadores do *skype* e de clientes de *IM (Instant Messaging)*, uma vez que o objectivo deste tipo de aplicações é estarem ligadas enquanto o utilizador fica a utilizar o computador, de maneira a receber chamadas, mensagens, ou actualizações nas subscrições.

Para se modelar correctamente o *churn* na rede LiveFeeds podemos-nos basear em [9, 29], uma vez que nos fornecem informações para a construção deste modelo, estes trabalhos estudam dois sistemas em que os utilizadores têm um comportamento análogo ao comportamento que se espera dos utilizadores do LiveFeeds.

De acordo com [9], um modelo de *churn* para o tipo de sistema considerado é constituído por duas componentes:

1. Uma distribuição que modele o tempo entre duas entradas consecutivas;
2. Uma distribuição que modele o tempo de sessão de cada nó que entra no sistema.

Em relação à primeira componente, a distribuição que modela o tempo entre duas entradas consecutivas, sabemos que deve depende da altura do dia e da altura da semana. Para construir este modelo de entrada, deve ser utilizada uma distribuição de *Poisson* não homogénea [9, 14]. Este tipo de distribuições depende de uma função $\lambda(t)$, que reflicta o comportamento dos utilizadores consoante a altura do dia e da semana (t). $\lambda(t)$ pode ser obtido através do

estudo efectuado em [29], com as adaptações devidas, uma vez que esse estudo toma por foco o tráfego *IM* de uma grande empresa. Além de confirmar a ideia de que as chegadas dos utilizadores podem ser modeladas de acordo com um processo de *Poisson* não homogéneo, [14] sugere que $\lambda(t)$ seja uma função que dependa do tempo de uma maneira periódica e tome a forma $\lambda(t) = \lambda(t+W)$ onde W é o período do processo. É desenvolvido este raciocínio para demonstrar que $\lambda(t)$ deve tomar a forma da expressão (5.1), em que W é igual a uma semana e a constante de proporcionalidade N_w é o número médio de intervalos de actividade por semana, as distribuições $p_d(t)$ e $p_w(t)$ modelam o começo de actividades consoante a altura do dia ($p_d(t)$) ou da semana ($p_w(t)$).

$$\lambda(t) = N_w p_d(t) p_w(t) \tag{5.1}$$

Como demonstrado em [14], as distribuições a usar para $p_d(t)$ e $p_w(t)$ podem ser inferidas através de dados obtidos através de observações de sistemas semelhantes ao sistema para o qual se quer encontrar um modelo de *churn*. Estas distribuições representam apenas a probabilidade de começar uma dada actividade a uma dada altura do dia ou da semana, t.

A utilização de uma distribuição de *Poisson* não homogénea foi equacionada para que se modelasse correctamente o facto de os utilizadores terem diferentes comportamentos dependendo da altura do dia e da semana em consideração, no entanto como se pretende estudar o comportamento do algoritmo perante o pior caso possível, na altura pior do pior dia da semana. É então possível fazer uma simplificação do modelo de *churn*, através do uso de uma distribuição de *Poisson* homogénea que represente o pior caso do com que o sistema LiveFeeds tem de suportar. Com base em trabalhos relacionados, essa distribuição foi calibrada e foram executadas algumas simulações com diferentes valores de r, o número de eventos de entrada e saída por segundo. Uma vez que a distribuição homogénea de *Poisson* aceita um parâmetro λ, esse parâmetro foi modificado em função do r pretendido, considerando $\lambda_{chegada} = 1/r$, pois a distribuição modela o tempo entre chegadas consecutivas e assim um valor esperado de $1/r$ segundos entre chegadas corresponde a um valor esperado de r chegadas por segundo. Para que exista uma maior granularidade, o valor de $\lambda_{chegadas}$ é utilizado em centésimos de segundo, por exemplo, se o pretendido for um valor esperado de 4 chegadas por segundo, $\lambda_{chegadas}$ deve ser igual a $1/r \times 100 = 1/4 \times 100 = 25$. A figura 5.1 mostra o aspecto da função densidade de probabilidade de uma distribuição de *Poisson* homogénea parametrizada com $\lambda_{chegadas} = 25$.

Para finalizar a construção do modelo de *churn* do LiveFeeds, deve ser encontrado uma outra distribuição que modele correctamente tempo de permanência na rede de um nó que entra na rede.

O tempo de permanência dos nós na rede segue uma distribuição *heavy-tailed* [9, 29]. Em [29] é desenhado um gráfico da *Comulative Distribution Function* do tempo de permanência dos nós na rede. Seguindo a proposta sobre a utilização de uma distribuição *heavy-tailed*,

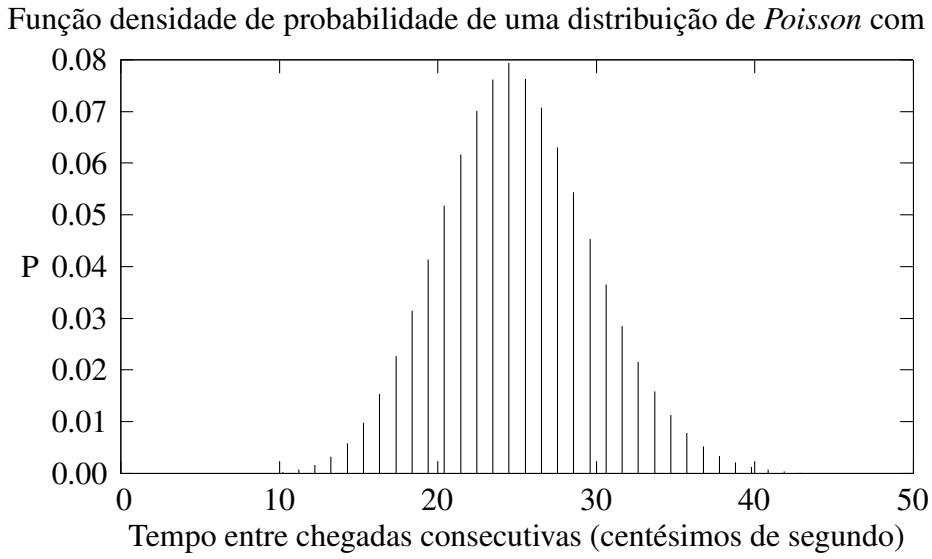


Figura 5.1: Gráfico da função densidade de probabilidade para o tempo entre chegadas consecutivas, modelado por um processo de *Poisson* homogéneo com $\lambda_{chegadas} = 25$ o que equivale a um valor esperado de 25 centésimos de segundo.

podemos utilizar uma distribuição de *Weibull* para modelar o tempo de permanência dos nós na rede [29] e de seguida construir um modelo de *churn*.

A distribuição de *Weibull* é definida por dois parâmetros, λ_{sessao}[1] e k, que podem ser obtidos a partir de [29]. Foram obtidos por observação dos gráficos desse trabalho, os valores de $\lambda_{sessao} = 5 \times 10^3$ e $k = 0.5$. Além desta observação dos gráficos, foi tido em conta o que estes parâmetros representam no contexto da distribuição de *Weibull*. O parâmetro k representa a taxa de "mortalidade infantil" dos nós, ou seja, a taxa de nós que está na rede durante muito pouco tempo. Foi definida uma taxa de 0.5 por ser o valor mais consistente com [29].

Obtivemos assim os componentes necessários para podermos construir um modelo de *churn* para o LiveFeeds. Esse modelo é composto pelo modelo de entradas de utilizadores, descrito por um processo de *Poisson* homogéneo, com parâmetro $\lambda_{chegadas}$ e pelo modelo de tempo de permanência dos utilizadores na rede, que é representado por uma distribuição de *Weibull* com parâmetros λ_{sessao} e k.

5.2 Parâmetros e métricas da simulação

As simulações apresentadas neste capítulo têm alguns parâmetros e métricas em comum.

[1]Para evitar ambiguidades, designa-se o λ utilizado pela distribuição de *Poisson* por $\lambda_{chegadas}$ e por λ_{sessao} o λ utilizado para a distribuição de *Weibull*

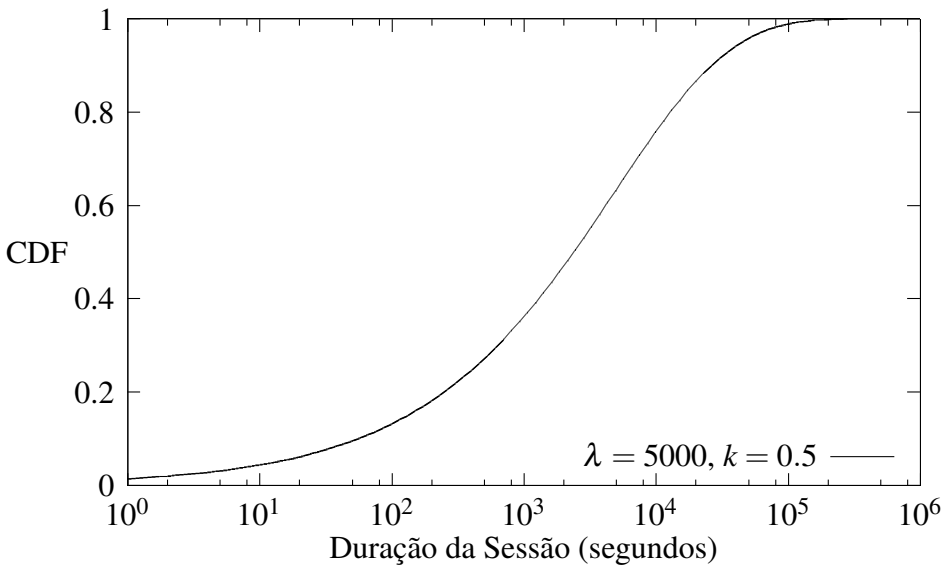


Figura 5.2: Gráfico da *Comulative Distribution Function* para o tempo de duração de uma sessão modelada por uma distribuição de *Weibull* com parâmetros $\lambda sessao = 5000$ e $k = 0,5$

Métricas utilizadas e sua representação gráfica

São aqui descritas todas métricas utilizadas nas simulações apresentadas nos capítulos seguintes.

Durante o seu tempo de vida e periodicamente, a cada t_s segundos, cada nó calcula o valor de $u = b/t_s$ onde b é o número de bytes enviados. No fim de vida de cada nó, é assim possível verificar qual foi o maior valor de u observado de entre todos os valores calculados a cada intervalo de amostragem de t_s segundos. É da mesma forma possível saber qual foi a média do valor de u durante todo o tempo de vida do nó.

É também possível obter o número de bytes enviados e recebidos durante todo o tempo de vida do nó, que é também conhecido, sendo assim possível calcular a média do tráfego de *upstream* e *downstream* com base no tempo de vida do nó.

Nos gráficos estão representadas as seguintes métricas:

Upload Max Esta métrica representa o maior valor de banda passante de *upstream* alguma vez observado durante o tempo de amostragem, no conjunto de nós com tempo de sessão igual a t. O valor de "Upload Max" é o máximo alguma vez observado no conjunto de todos os valores de u calculados por todos os nós com tempo de sessão igual a t.

Max Upload Average Esta métrica representa o valor máximo da média de banda passante despendida pelos nós com tempo de sessão igual a t, calculado com base numa amostra de t_s segundos. Para cada nó com tempo de sessão igual a t é calculada a média de u em relação a todos os valores de u observados ao longo do tempo de vida do nó. O valor de

"Max Upload Average" é o pelo máximo de todas essas médias.

Upload Average Para cada nó que termina a sua sessão é calculado o valor da divisão do número de bytes enviados sobre o tempo de sessão, obtendo-se assim a média de banda passante ao longo de toda a sessão do nó. O valor representado representa a média do valor médio de banda passante de *upstream* necessário pelos nós com tempo de sessão t.

Download Average Este valor é análogo ao valor "Upload Average", representando assim a média do valor médio de banda passante de *downstream* necessário pelos nós com tempo de sessão t. Nos gráficos apresentados relativos ao tráfego de *downstream* são ainda apresentados os valores minímos e máximos observados para "Download Average" através de uma área sombreada.

Uma vez que os dados registados por um nó são obtidos no fim de vida do mesmo, os dados obtidos são agrupados tendo em conta a duração da sessão dos nós. No eixo das abcissas são representados os tempos de sessão dos nós, enquanto no eixo das ordenadas é representado o valor em bytes / segundo correspondente à série em análise.

Parâmetros utilizados

Durante as experiências realizadas através de simulação admitiu-se que as saídas dos utilizadores não são tratadas de uma forma explicita pelos nós que permanecem no sistema, tal como já foi referido no capítulo 2. O valor de r considerado, que representa o número de eventos por segundo, passa assim a representar o número de entradas por segundo, pois as saídas não levam a que sejam formadas árvores de difusão. É assim possível determinar o número de eventos gerados por segundo através da parametrização do processo de *Poisson* homogéneo, que toma como parâmetro $\lambda chegadas$. Nesta hipótese, a saída de um nó não tem qualquer influência sobre o algoritmo de filiação considerado e a distribuição utilizada para modelar o tempo de sessão de um nó, bem como os parâmetros utilizados para calibrar essa distribuição, não têm influência sobre os resultados obtidos, pois como vimos, o tráfego de *upstream* e *downstream* dos nós do sistema toma valores segundo a expressão $b_u = b_d = r \times m$, onde r é a taxa de eventos observada pelos nós. O factor determinante na modelação do sistema é assim a distribuição de *Poisson* parametrizada pelo parâmetro $\lambda_{chegadas}$ que pode ser alterado em função do r pretendido para cada simulação, utilizando a expressão $\lambda_{chegadas} = 1/r$. A distribuição de *Poisson* configura o comportamento do algoritmo de filiação pois esta distribuição é o único factor que afecta o valor da taxa de eventos por segundo no sistema, r.

Outro factor que tem especial importância nesta fase é o tempo durante o qual o *slice leader* agrega mensagens antes de iniciar o *broadcast* correspondente a essas mensagens. O sistema foi configurado de maneira a que os *slice leaders* esperassem a recepção de um número aleatório de mensagens entre 15 e 20 ou esperassem entre 20 a 30 segundos antes de iniciar o *broadcast*

das mensagens. O objectivo deste mecanismo foi o de evitar a sincronização entre os *slice leaders* de forma a que se diminuísse a probabilidade de um nó ser escolhido por vários *slice leaders* ao mesmo tempo para iniciar a árvore de difusão, o que levaria a que momentaneamente fosse requerida uma grande capacidade de comunicação suplementar. O mesmo fenómeno de sincronização poderia acontecer nos outros nós interiores em árvores de difusão simultâneas.

As simulações apresentadas de seguida foram realizadas usando os parâmetros representados na tabela 5.1.

Parâmetro	Valor
Grau da árvore de difusão, G	4
Tamanho da tabela de filiação	0 B
Tamanho da informação de filiação de um nó	500 B
Número de *slice leaders* (SL)	4
Periodicidade de envio de mensagem de um SL	25 a 30s ou 15 a 20 mensagens
Tempo de sessão	300 a 7200 segundos
Distribuição de tempo de sessão	*Weibull* com $\lambda sessao = 5000$ e $k = 0,5$
Distribuição dos tempo entre chegadas	*Poisson* com $\lambda_{chegadas} = 1/2$, $1/4$ e $1/10$ (x 100)
Tempo de amostragem, t_s	25 s

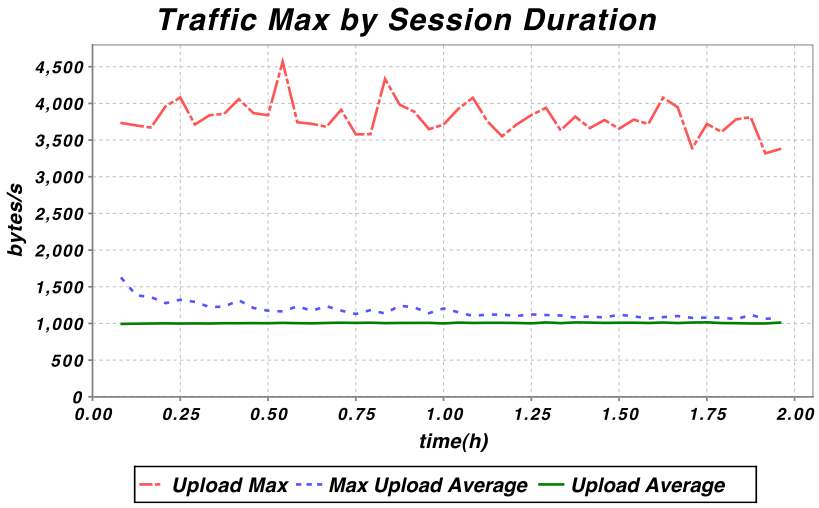
Tabela 5.1: Parâmetros usados nas simulações com o modelo de *churn*

5.3 Resultados

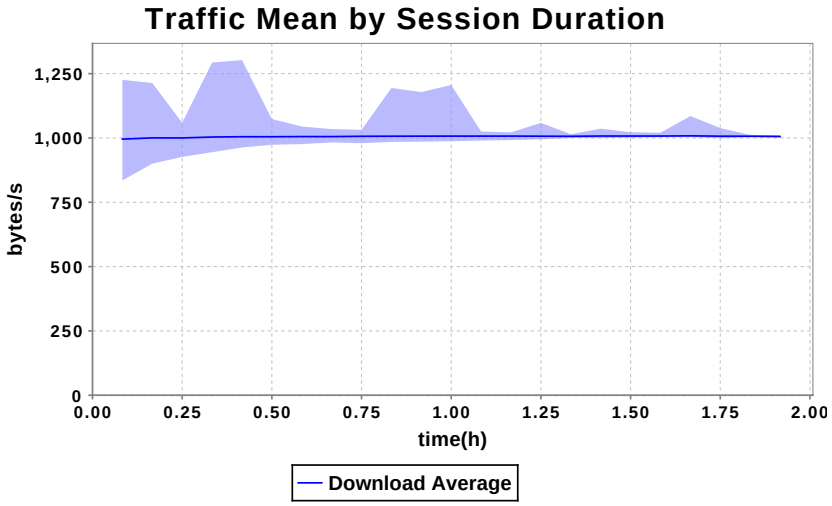
São aqui apresentadas algumas simulações para vários valores de r, a taxa de entradas por segundo.

Os gráficos da figura 5.4 apresentam os valores médios das grandezas acima descritas para nós agrupados por classes caracterizadas por tempos de vida no caso em que a distribuição do tempo entre chegadas dos nós é parametrizada por $\lambda = 1/4 \times 100 = 25$, o que equivale a um valor esperado de 25 centésimos de segundo entre chegadas e a um valor médio de $r = 4$ chegadas/segundo.

De acordo com os resultados obtidos no capítulo 4, num sistema não dinâmico a difundir as mensagens correspondentes às alterações de filiação, alterações estas que ocorrem r vezes por segundo, os nós deveriam suportar uma capacidade de *upstream* e *downstream* caracterizada por $b_u = b_d = r \times m_t$ B/s. Como se pode observar pelas figuras 5.4, 5.3 e 5.5, os dados obtidos para os valores médios estimados no fim de vida de cada nó, confirmam que a expressão para b_u e b_d previamente obtida continua a ser uma boa aproximação mesmo quando transposta para condições de simulação em que existe *churn*. Por exemplo, no caso em que o valor esperado



(a) Tráfego de *upstream*

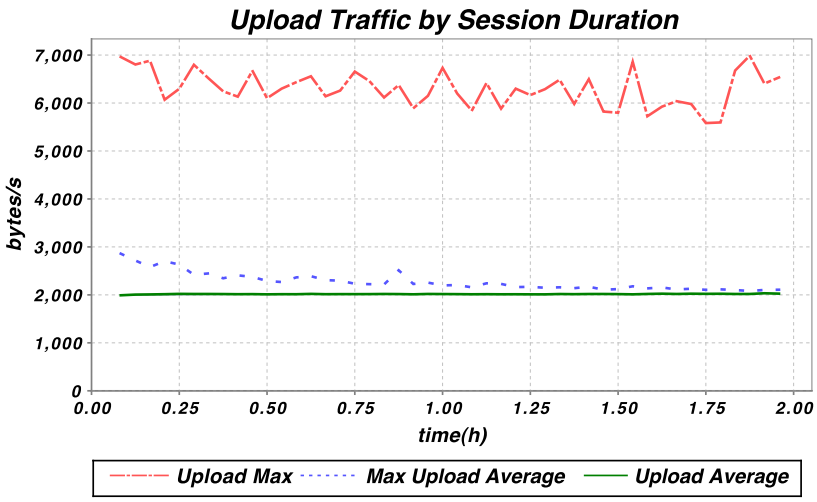


(b) Tráfego de *downstream*

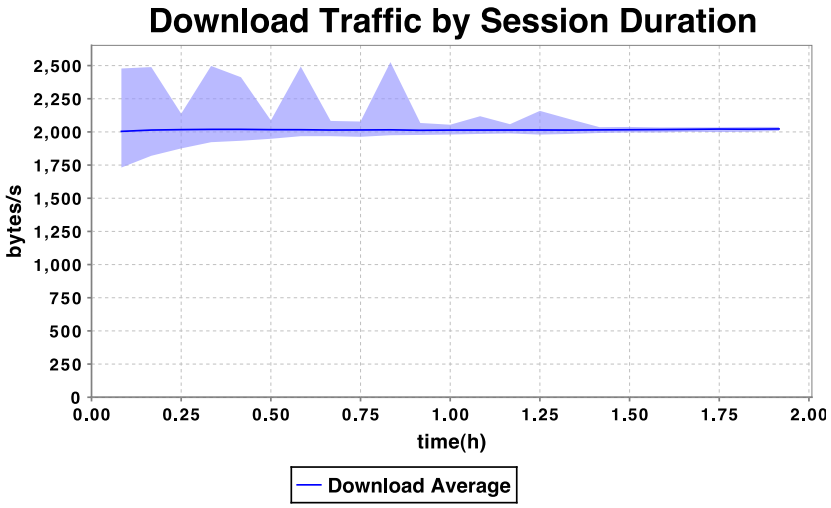
Figura 5.3: Resultados obtidos com a distribuição do tempo entre chegadas parametrizada com $\lambda_{chegadas} = 1/2$

para r é 4 Eventos/Segundo, podemos verificar na figura 5.4 que o valor médio do tráfego de *upstream* é de cerca de $2000B/s$, o que é consistente com a expressão $b_u = r \times m = 4 \times 500 = 2000B/s$. É também claramente visível que os valores médios encontrados para a capacidade despendida convergem para um valor comum. Quanto maior é a sessão de um nó, menos díspares são os valores observados, o que é consistente com a análise feita anteriormente.

Os valores obtidos para "Upload Max" mostram o pior caso possível que alguns nós tiveram de suportar, uma vez que representa a banda passante necessária por parte de um nó durante os piores t_s segundos de entre todos os nós com o mesmo tempo de sessão. Esta métrica fornece assim um ponto extremo para o cálculo dos limites do algoritmo simulado. No caso em que o valor esperado era $r = 4$ Eventos/Segundo, existiu pelo menos um nó que necessitou de suportar o envio de $7000B/s$ num intervalo de $t_s = 25s$, tendo estado esse nó no sistema cerca de 1,9



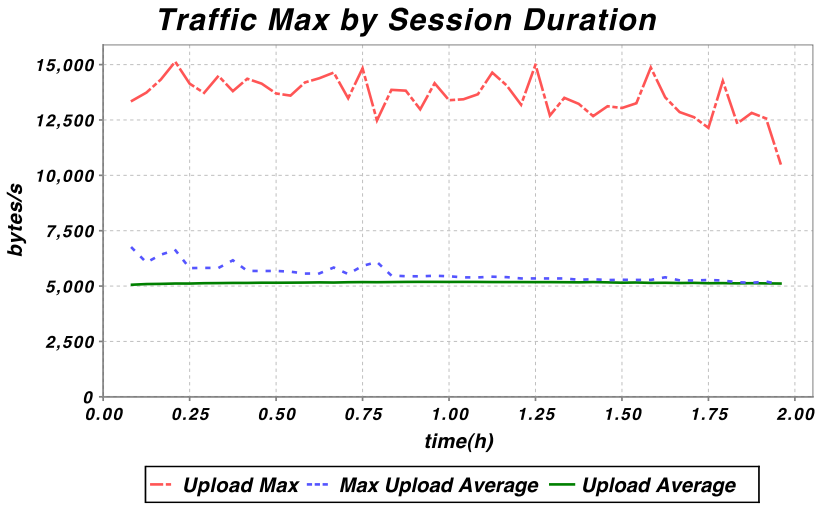
(a) Tráfego de *upstream*



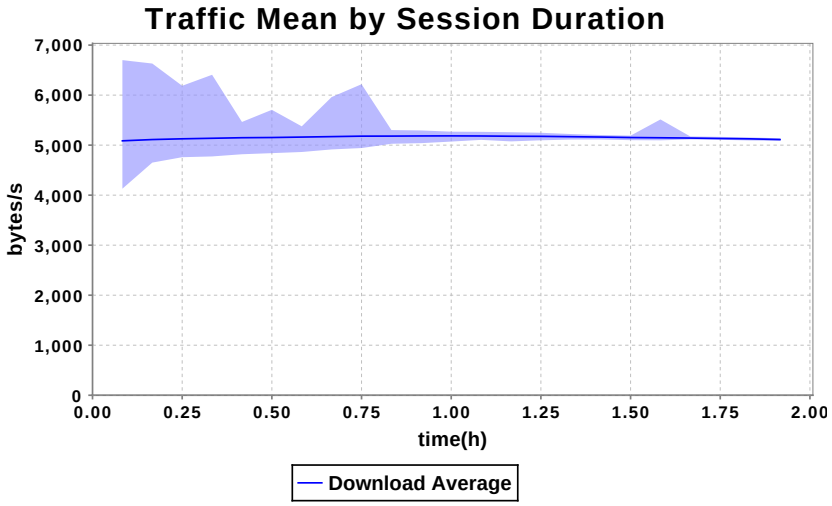
(b) Tráfego de *downstream*

Figura 5.4: Resultados obtidos com distribuição do tempo entre chegadas parametrizada com $\lambda_{chegadas} = 1/4$

horas e necessitado maior valor de banda passante durante o período de amostragem do que os outros nós. Em relação aos valores para a banda passante de *downstream*, para o mesmo valor de r, não se nota uma variação significativa nos valores observados. Embora existam algumas flutuações, o valor médio situa-se sempre nos $b_d = r \times 500 = 2000B$, o valor esperado no cenário ideal analisado no capítulo 4. Pela observação deste gráfico corrobora-se também o resultado já referido de que em média e ao fim de um número muito grande de *broadcasts* o valor da banda passante de *downstream* será igual ao valor da banda passante de *downstream*, uma vez que se verifica que $b_u = b_d \approx 2000B$. Tal facto é posto em evidência pelo facto de o intervalo de variação entre o mínimo e o máximo diminuir com o aumento do tempo de sessão, como se pode observar pela diminuição, ao longo do tempo de sessão, da área sombreada dos gráficos referentes ao tráfego de *downstream* que representa os valores minímos e máximos



(a) Tráfego de *upstream*



(b) Tráfego de *downstream*

Figura 5.5: Resultados obtidos com distribuição do tempo entre chegadas parametrizada com $\lambda_{chegadas} = 1/10$

registados para a métrica "Download Average". Os valores observados nas simulações com os restantes valores de r são equivalentes e os resultados análogos.

O facto de o valor de "Max Upload Average", medido de forma diferente de "Upload Average", uma vez que é calculado com base em intervalos de t_s segundos e não no tempo total de sessão, estar sempre muito próximo de "Upload Average", sugere que o tempo de amostragem utilizado de $t_s = 25s$ é adequado, validando os resultados obtidos na simulação. Note-se ainda que quando representados os valores de "Upload Average" utilizando como intervalo de amostra não todo o tempo de sessão do nó, mas apenas o intervalo t_s ("Upload Average Max"), as linhas do gráfico sobrepõem-se, confirmando novamente que a aproximação da utilização de $t_s = 25s$ é uma boa opção pois embora os valores para a métrica "Upload Average" calculados

com $t_s = t_{sessao}$ sejam semelhantes aos valores obtidos com $t_s = 25s$, a utilização de um intervalo de amostra mais pequeno permite obter uma maior resolução sobre os valores máximos da banda passante de *upstream* necessária pelo nó, "Upload Max".

5.4 Conclusões

Com base na análise feita, podemos tirar algumas conclusões sobre a aplicabilidade do algoritmo de filiação do LiveFeeds com visibilidade completa. A utilização de árvores de difusão aleatórias tem como benefício que a capacidade de *upstream* necessária é proporcional à taxa de eventos por segundo observada no sistema. A capacidade que cada nó deve suportar para que o mecanismo considerado possa ser utilizado não depende assim do tamanho do sistema, mas sim do comportamento dos seus utilizadores. Se esse comportamento for estável o suficiente, o mecanismo de visibilidade completa pode ser utilizado com um grande número de nós. Desta forma, é possível representar o custo de implementação do sistema de visibilidade completa analisado com base na expressão $b_u = r \times m$, dependendo do que os utilizadores estão dispostos a despender no pior caso em relação às capacidades de *upstream* e *downstream* e do tamanho da informação de filiação do sistema, m, o algoritmo suportará uma taxa de eventos difundidos por segundo, r, maior ou menor.

Dependendo do tipo de conectividade dos possíveis utilizadores do sistema, é possível avaliar a viabilidade com base na expressão $b_u = r \times m$, se os utilizadores tiverem um comportamento suficientemente estável, será possível utilizar este mecanismo despendendo de um valor de capacidade de *upstream* relativamente baixo, a figura 5.6 mostra a capacidade de *upstream* (igual à capacidade de *downstream*) necessária em função da taxa de eventos por segundo, r.

Neste sentido, podemos sem dúvida considerar realista o uso do mecanismo de visibilidade completa num sistema em que os nós se comportam como *brokers* e poderão estar alojados em instituições de ensino, *ISPs*, grandes empresas, etc. e que servem os *feeds* aos utilizadores. Este tipo de nós têm tipicamente boa conectividade e tempos de sessão muito longos, o que leva a que, num sistema constituído maioritariamente por este tipo de nós, a taxa de eventos por segundo seja baixa o suficiente para viabilizar que seja suportado um grande número de nós. A título de exemplo, se considerarmos uma rede com $N = 10000$ nós, em que os nós têm um tempo sessão médio de cerca de 3 horas, isto equivale a uma taxa de eventos de cerca de $10000/10800 \approx 0,9$ eventos/segundo. Os nós teriam assim de contribuir com uma capacidade de *downstream* e *upstream* de $r \times m = 0,9 \times 500 = 450$ bytes por segundo para manter a informação de filiação. Este custo é bastante baixo para o tipo de nós em questão o que torna a implementação possível neste cenário. O gráfico da figura assume um modelo de *churn* simples para mostrar qual a capacidade de *upstream* (igual à capacidade de *downstream*) que cada nó deve despender em função do tempo de sessão médio dos nós, para um sistema com $N = 10000$

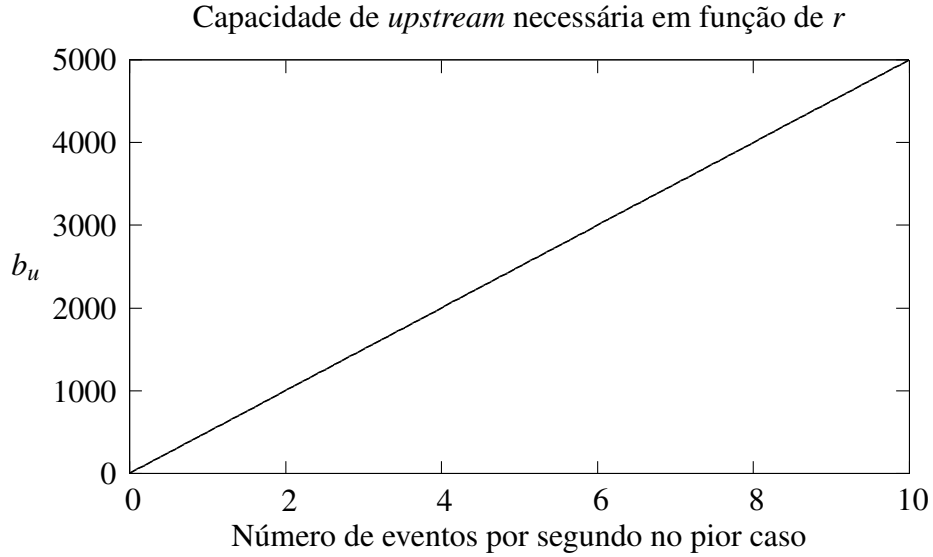


Figura 5.6: Capacidade de *upstream* necessária, b_u (Bytes/s), em função da taxa de eventos observada no pior caso admitindo $m = 500B$.

nós.

Em comparação com o algoritmo proposto em [10], o algoritmo de encaminhamento com visibilidade completa requer valores de capacidade de *upstream* e *downstream* da mesma ordem de grandeza, no entanto, oferece uma distribuição melhor da carga por todos os nós intervenientes, não existindo vários tipos de nós a desempenhar funções diferentes, à excepção dos *slice leaders* cuja capacidade requerida é o dobro da dos nós normais. É possível no entanto utilizar um sistema com um número muito pequeno de *slice leaders*, o que leva a que a distribuição de carga não seja afectada significativamente.

Para que o sistema suporte utilizadores com um comportamento mais dinâmico, é necessário encontrar outras soluções, como a introdução de super-nós, esta solução é objecto de estudo no capítulo 6.

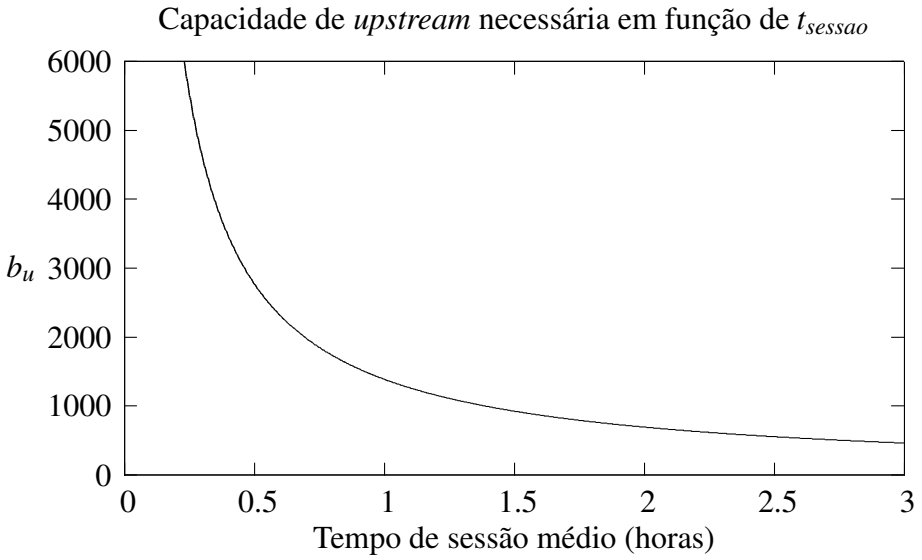


Figura 5.7: Capacidade de *upstream* (b_u em Bytes/s) que cada nó deve despender em função do tempo de sessão médio dos nós, num sistema com $N = 10000$ nós, assumindo um modelo de *churn* simples

6 . Sistema hierárquico - Super-nós

A análise anterior mostra que os custos de comunicação envolvidos no algoritmo de filiação estão dependentes da taxa de entrada de utilizadores no sistema. Dependendo da capacidade de rede que se pretende investir no algoritmo, o mesmo será ou não aceitável em determinados contextos. É assim necessário encontrar outras soluções para que o sistema possa ser utilizado num quadro em que o seu custo não é compatível com a capacidade disponível.

Existem várias vias para diminuir a dependência do algoritmo de filiação em relação à taxa de entradas observadas. No que se segue exploraremos uma solução que apesar de manter a linearidade, diminui a derivada dessa dependência. Essa solução baseia-se na noção de "super-nó".

Em primeiro lugar, são apresentados alguns trabalhos relacionados com a utilização de super-nós para resolver alguns problemas normalmente encontrados ao desenhar redes P2P. Em 6.2 é apresentado o algoritmo de super-nós utilizado. Seguidamente são expostos alguns resultados obtidos através da utilização do algoritmo de super-nós descrito. Na secção 6.3 é feita uma síntese dos critérios utilizados nas simulações e em 6.4 são apresentados os resultados de simulações em que são utilizados diferentes critérios de selecção de nós para promoção ao estatuto de super-nó. Na secção 6.5 é proposto e analisado um melhoramento ao algoritmo que consiste na utilização dos nós filhos no envio de mensagens com informação de filiação. Finalmente, na secção 6.8 são analisados outros melhoramentos que podem ser feitos ao algoritmo de super-nós.

6.1 Trabalho Relacionado

Depois de ter sido analisado o custo, em termos de capacidade de *upstream* e *downstream*, para implementar o algoritmo de visibilidade completa, foi necessário encontrar uma solução para que fosse possível suportar utilizadores mais dinâmicos, o que exigiria um custo de comunicação superior ao aceitável num dado contexto.

A utilização de super-nós é uma solução comum em sistemas P2P que queiram alcançar uma maior escalabilidade suportando um elevado *churn* dos utilizadores, e é a aproximação utilizada por sistemas como o *Skype* [9] e o *KaZaA*.

A utilização de super-nós em sistemas editor/subscritor em redes *peer-to-peer* foi já proposta em [5], onde é reconhecida a complexidade subjacente a este paradigma e é aceite o facto de que mesmo utilizando super-nós para actuar como servidores para os subscritores, é necessário que os nós pertencentes à rede de super-nós tenham uma visibilidade completa dos filtros de todos os super-nós para que a difusão filtrada possa ser eficiente. Tal trabalho estuda a hipótese de utilização de múltiplos grafos acíclicos para que possa ser feito um encaminhamento

eficiente das notificações entre editores e subscritores. Apesar de esta solução ser definida formalmente, em [5] não são propostos métodos e métricas que possam ser utilizadas para avaliar o custo da utilização deste sistema em termos da capacidade de rede necessária por parte de cada nó interveniente e não é também apresentado trabalho experimental de forma a avaliar o sistema definido formalmente, em particular, não é utilizado um modelo de *churn* adequado para que possa ser avaliada a aplicabilidade do sistema proposto.

Um super-nó é um nó de uma rede P2P que opera como servidor para um conjunto de clientes e ao mesmo tempo opera como um nó da mesma hierarquia numa rede de P2P de super-nós. A utilização de uma arquitectura de super-nós permite assim usufruir das vantagens de uma arquitectura híbrida entre as arquitecturas cliente-servidor e P2P [30]. Do ponto de vista de ISPs, existem ainda vários motivos para um ISPs disponibilizar serviços de super-nós numa rede P2P, pois todo o tráfego entre clientes do ISPs e os super-nós do mesmo ISP será sempre mais económico para o ISP do que tráfego que atravesse diferentes ISPs [24]. A figura 6.1 apresenta o diagrama de uma arquitectura típica de super-nós em que os nós de cor escura formam uma rede independente de super-nós e os nós mais claros são clientes de um super-nó que actua como servidor para os mesmos. A figura 6.2 representa uma arquitectura de super-nós em que os filhos podem ter mais do que um super-nó pai.

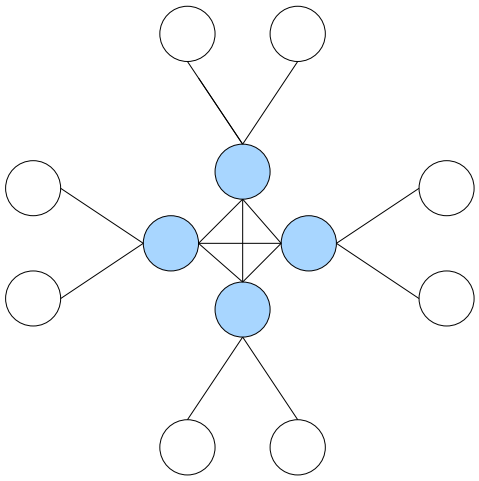


Figura 6.1: Diagrama de uma arquitectura típica de super-nós

Utilizando os super-nós como servidores e os nós normais como clientes, as operações sobre o sistema P2P podem ser feitas pelos clientes dos super-nós de uma maneira mais eficiente do que se estes tivessem de realizar todo o trabalho necessário para completar uma operação, tal como acontece num sistema P2P puro. A utilização de super-nós diminui o número de nós que têm de executar os algoritmos P2P, contribuindo assim para o aumento da escalabilidade do sistema. Por outro lado, os clientes dos super-nós ficam assim isentos de algumas dificuldades

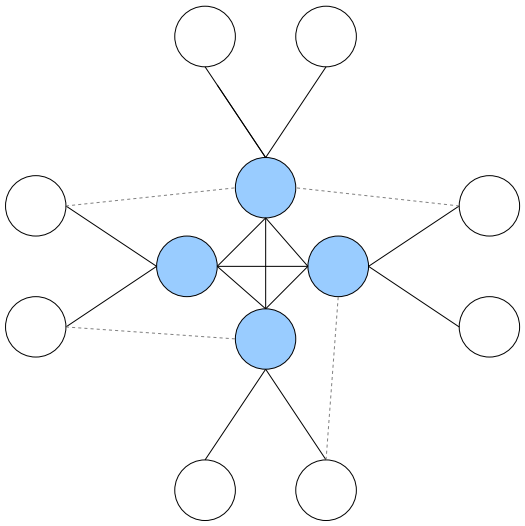


Figura 6.2: Diagrama de uma arquitectura super-nós em que existem filhos com mais do que um super-nó pai

associadas à realização de consultas numa arquitectura P2P, pois o super-nó actua como um servidor de operações dentro da rede de super-nós, mas não existe um único ponto de falha tal como acontece numa arquitectura cliente-servidor tradicional.

Uma arquitectura de super-nós permite atribuir mais responsabilidades e consequentemente mais trabalho aos nós da rede que possam suportar esta responsabilidade. Esta hierarquização não é observada nas arquitecturas P2P típicas, sem super-nós, mas permite que se utilize a maior capacidade de rede de nós que possam despender desta capacidade, o que permite minimizar o sub-aproveitamento existente normalmente em redes cujos nós intervenientes configuram um sistema heterogéneo com nós de diferentes características, uns com mais capacidade do que outros. Apesar destes melhoramentos, utilizando uma arquitectura de super-nós, mantém-se a robustez e escalabilidade subjacentes a uma arquitectura P2P. A utilização de uma arquitectura deste tipo está no entanto revestida de dificuldades, questões como a determinação da correcta proporção de nós para super-nós ou a implementação de formas de suporte a falhas dos super-nós [30], são ainda determinantes numa arquitectura de super-nós e são difíceis de resolver.

Ao desenhar uma rede de super-nós que possa resolver o problema do LiveFeeds, surgem várias opções de desenho que devem ser cuidadosamente ponderadas. Ao ser responsável por um conjunto de nós filhos, um super-nó fica com a responsabilidade de promover os seus filhos a super-nós de maneira a que a rede de super-nós possa crescer e suportar assim mais nós filhos. A escolha do momento mais apropriado para uma promoção por parte de um super-nó é também um problema que pode ter uma grande influencia no comportamento do algoritmo de super-nós implementado. Também o critério que é utilizado para escolher o super-nó que fica responsável por cada nó que entra na rede deve ser estudado de forma a que a entrada de novos

nós seja o mais eficiente possível. O número ideal de clientes que os super-nós devem aceitar é também um parâmetro de grande relevância pois vai ter influência tanto no comportamento global do sistema como no esforço requerido por um novo nó.

Em relação ao critério de selecção do nó que deve ser pai de um novo nó filho, podemos começar por fazer uma selecção aleatória de entre o conjunto de super-nós do sistema e atribuir esta responsabilidade a um super-nós assim escolhido. Por outro lado, existem trabalhos [18, 7], que investigam a escolha de critérios mais complexos para esta escolha do super-nó que serve de super-nó semente para a entrada de um novo nó na rede. Em [18] são sugeridos diferentes critérios para fazer esta selecção, sendo um desses critérios a utilização de um sistema que possa medir a distância entre os nós do sistema utilizando um sistema de coordenadas virtuais. Ao ter este tipo de informação disponível entre os vários nós, poderá ser vantajoso agrupar nós que estejam mais perto entre si e atribuir a responsabilidade de um novo filho a um super-nó que esteja mais perto desse filho. No mesmo trabalho, é sugerido um critério em que os filhos são atribuídos a super-nós com base na compatibilidade dos filtros de subscrições dos filhos com os super-nós. Também em [7] é estudado um sistema de gestão de relações entre nós do sistema que se baseia no agrupamento de utilizadores utilizando a semelhança entre os filtros de subscrição. Este critério pode revelar-se uma boa aproximação no futuro ao ser introduzido o estudo da distribuição dos filtros de subscrição dos possíveis utilizadores do LiveFeeds. Em [18] é ainda sugerido outro critério em que são utilizadas métricas para medir a capacidade de computação de um nó. Embora este critério não pareça ser determinante do ponto de vista da escolha do nó semente no caso do LiveFeeds, poderá revelar-se vantajoso como critério para que um super-nó faça a selecção dos filhos a promover.

Quanto ao critério de escolha do nó filho que um super-nó deve seleccionar para ser super-nó, quando é decidido que essa promoção deve ser feita, podem existir também vários critérios de escolha. Os critérios em que são escolhidos os nós filhos com base no tempo de sessão do nó são um natural primeiro passo. Outros critérios mais complexos poderão ser equacionados no futuro e o seu desempenho medido de forma cuidada. Em particular, em [3], é sugerido um protocolo para gerir a reputação dos nós do sistema, esta reputação pode ser medida em termos da capacidade de rede de um nó. Este trabalho pode no entanto inspirar a criação de um sistema de reputação em que esta é medida em termos de tempos de sessão passados, o que permite estimar o tempo de sessão futuro de todos os filhos e promover o filho com maior probabilidade de ter um tempo de sessão maior. Espera-se que este tipo de critério leve a que existam super-nós com tempos de sessão longos o que se traduz em menos entradas na rede uma vez que os filhos dos super-nós que saem devem sempre encontrar um novo super-nó. Os custos de implementação de um mecanismo podem, no entanto, revelar-se demasiado elevados, uma vez que o sistema de reputação sugerido em [3] depende da utilização de identificadores persistentes entre sessões e armazenamento persistente dos dados sobre reputação de utilizadores de forma a garantir a validade do sistema. O sistema proposto em [3] baseia-se ainda na partilha de

informação acerca da reputação dos utilizadores por parte de todos os super-nós, o que leva ao aumento dos custos em termos de capacidade de rede quando se pretende partilhar informação de reputação sobre muitos utilizadores. Depois de determinar o critério a utilizar para escolher o nó filho a promover, os super-nós devem seguir um outro critério para decidir quando devem promover um dos seus filhos. A promoção de filhos apenas quando um super-nó atinge o número máximo de de filhos é uma das soluções, no entanto, esta aproximação pode levar a um efeito de *feedback* na rede de super-nós. Para evitar este efeito, pode ser utilizado um mecanismo de promoção por antecipação em que os filhos são promovidos antes de o super-nó ser responsável pelo número máximo de filhos e com base numa função de probabilidade. Seria ainda possível a utilização de um critério em que os super-nós promovem os seus filhos com base no estado actual da rede de super-nós, de forma a adaptar o ritmo de entrada de novos super-nós à taxa de entradas por segundo observadas, sendo também necessário utilizar formas de evitar o efeito de *feedback* possivelmente introduzido através da utilização deste critério.

Outros trabalhos foram também considerados de forma a investigar possíveis novos melhoramentos que possam ser relevantes no futuro. Da arquitectura de super-nós proposta, podemos verificar que ao sair da rede, um super-nó provoca a entrada na rede dos filhos que se tornam órfãos. Este problema pode ser atenuado se forem seguidas as ideias estudas em [19], que sugere que se atribuam vários super-nós pais ao mesmo filho de forma a que um filho não se torne órfão devido à simples saída do super-nó responsável pelo nó. A ideia da utilização de múltiplos super-nós por parte dos filhos é suportada por diversos trabalhos estudados [7, 30, 19]. Em [19] é ainda sugerido que sejam utilizados mecanismos dentro da rede de super-nós de forma a que os vários super-nós partilhem informação sobre a carga a que estão sujeitos de forma a poderem fazer uma melhor distribuição de carga. Esta partilha de informação pode ser ainda estendida à informação acerca dos filhos dos super-nós vizinhos e pode levar assim à diminuição do tempo de recuperação do sistema no caso de falha de um super-nó. Este tempo de recuperação pode ser ainda mais breve se for utilizada uma forma de cada super-nó detectar falhas de nós vizinhos e actuar de forma a minimizar o impacto dessa falha, acolhendo os filhos do super-nó que falhou [19].

6.2 O algoritmo de Super-nós

O algoritmo aqui apresentado suporta-se na ideia de que existem nós que são responsáveis pelos novos nós do sistema. Ao entrar no sistema, um nó torna-se nó filho e fica dependente de um super-nó, o seu nó pai. Este nó filho não entra directamente na rede super-nós, isso só acontecerá se o seu pai decidir que o deve promover a super-nó. É apenas nesta altura que os outros super-nós vão observar a entrada de um novo nó no sistema, pois cada super-nó observa apenas a entrada de nós na rede de super-nós e a entrada dos seus novos filhos.

Introduz-se assim uma hierarquização do sistema, passam a existir nós de dois tipos distintos: os nós e os super-nós, sendo criada uma rede em que todos os super-nós conhecem todos os outros super-nós para além de conhecerem os seus próprios filhos. Ao introduzir este tipo de organização no sistema, diminui-se a taxa de entradas observadas pelos super-nós, uma vez que a visibilidade de cada super-nó está limitada à rede de super-nós. O problema dos utilizadores que estão muito pouco tempo dentro do sistema é atenuado pois este tipo de nós não chegam provavelmente a ser promovidos e assim a sua entrada não é observada por todos os super-nós mas apenas pelo nó pai. Naturalmente, o pai fica responsável por encaminhar toda a informação necessária ao correcto funcionamento do sistema de difusão de mensagens referentes à difusão de conteúdos, pois os filhos não são conhecidos pelos outros super-nós.

No algoritmo de super-nós, um nó a para entrar no sistema contacta um super-nó, s, que serve de semente para o nó a. Ao receber o pedido de a, s deve decidir se pode aceitar um novo filho. Seguidamente, o super-nó s comunica a sua decisão ao nó a. Caso s decida que não pode aceitar a como filho, envia juntamente com a resposta o endereço de um outro nó ao qual a se deve dirigir para tentar novamente. Este processo repete-se até que um super-nó aceite a como filho e destina-se a tentar que o número de filhos seja equilibrado nos diferentes super-nós.

Após algumas tentativas recusadas, o filho pode forçar a sua entrada. Ao receber um filho nestas condições, um super-nó deve promover um dos seus filhos para que este se junte à rede de super-nós de forma a que o pai possa acomodar o novo nó. O critério que o super-nó utiliza para seleccionar o filho que deve promover tem grande influência no comportamento do sistema e é objecto de estudo neste capítulo.

Depois de o nó a ter encontrado um super-nó para seu pai, deve permanecer no sistema aguardando mensagens vindas do seu pai até que receba uma mensagem de promoção, altura em que pode iniciar o processo de entrada na rede de super-nós. Este processo é semelhante à entrada de um nó no sistema no caso em que não se utilizam super-nós (cf. capítulo 4).

No algoritmo aqui considerado, não existe por parte dos super-nós uma adaptação à taxa de entradas observada uma vez que os filhos são promovidos quando é necessário que um novo nó entre para o conjunto de filhos de um super-nó. Esta forma de gerir as promoções de filhos pode originar um comportamento em que todos os nós promovem filhos ao mesmo tempo levando a períodos em que existe um grande número de promoções e outros períodos em que poucos filhos são promovidos. Para anular este efeito de *feedback*, os super-nós implementam uma politica de promoção por antecipação em que os filhos são promovidos antecipadamente, mesmo antes de os super-nós terem atingido o seu número máximo de filhos. Ao adicionar um novo filho ao seu conjunto de filhos, um super-nó promove ou não um dos seus filhos com probabilidade P, sendo que P é o resultado de uma função $p(x)$ que depende do tamanho actual do conjunto de filhos, quanto mais próximo do número de filhos máximo, maior será a probabilidade de um super-nó promover um dos seus filhos. Nas simulações deste capítulo foi utilizada a função assim definida:

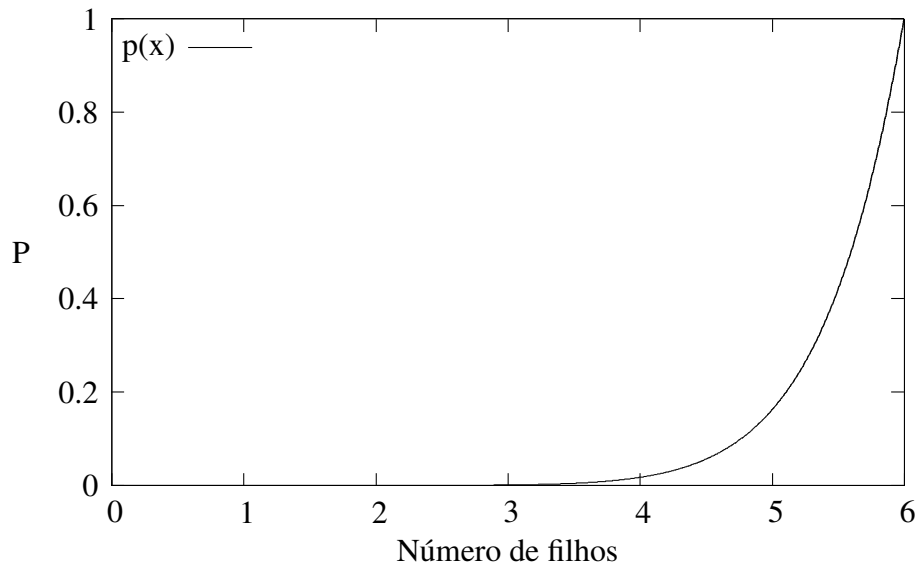


Figura 6.3: Probabilidade de um super-nó promover um nó ao adicionar um novo filho em função do número de filhos, dada pela função 6.1 com $X = 6$.

$$p(x) = \left(\frac{x}{X}\right)^{10} \tag{6.1}$$

Em que x é o número de filhos na altura da avaliação do valor de $p(x)$ e X é o número máximo de filhos que cada super-nó pode aceitar. A função $p(x)$ está representada na figura 6.3 para o caso em que $X = 6$. Como se pode observar, a probabilidade de um super-nó promover um filho ao acomodar um novo filho vai aumentando com o número de filhos actual.

É na altura em que o super-nó aceita um novo filho que decide ou não promover um dos seus filhos. Caso o super-nó decida, em função da probabilidade calculada da forma já referida, promover um nó, deve ser seleccionado um nó de entre todos os nós dos quais o super-nó é pai. Esta escolha pode ser feita tomando em conta vários critérios que têm assim uma grande influência no comportamento do algoritmo, como é mostrado na secção 6.4.

No algoritmo de super-nós sob consideração, os super-nós promovem filhos com base apenas nos pedidos de entrada de outros filhos, assim, o único evento que pode levar a uma promoção de um nó é a entrada de um novo filho seja esta uma entrada forçada por esgotamento de tentativas ou uma entrada que acontece em condições ditas normais.

O algoritmo simplificado executado pelos novos filhos é representado na figura 6.4, enquanto o algoritmo executado pelos super-nós ao receber um novo pedido por parte dos filhos é apresentado em 6.5.

As simulações deste capítulo têm em comum alguns parâmetros cujos valores são apresentados na tabela 6.1.

Parâmetro	Valor
Número de *slice leaders* (SL)	4
Periodicidade de envio de mensagem de um SL	25 a 30s ou 15 a 20 msgs
Tamanho de uma mensagem de filiação	500 Bytes
Tempo de sessão	300 a 7200 segundos (média de 2460 segundos)
Distribuição de tempo de sessão	*Weibull* com $\lambda_{tsessao} = 5000$ e $k = 0,5$
Distribuição dos tempo entre chegadas	*Poisson* com $\lambda_{chegada} = 1/2$, $1/4$ e $1/10$ (x 100)
Número máximo de nós filhos	6

Tabela 6.1: Parâmetros usados na simulação com super-nós

```
parent ← null
seed ← randomNode()
while NTries ≤ maxTries do
   reply ← msg_send(seed, PARENT_REQUEST)
   if reply = PARENT_REPLY_OK then
      parent ← seed
      break
   else
      seed ← reply.seed
   end if
   ntries++
end while
if parent = null then
   reply ← msg_send(seed, PARENT_REQUEST_FORCE_PARENT)
   parent ← seed
end if
```

Figura 6.4: Algoritmo de entrada de um novo nó filho

Como se pode observar pelo algoritmo apresentado na figura 6.4, um nó ao entrar no sistema deve encontrar um nó aleatório para iniciar a sua busca por um super-nó que aceite ser seu pai. Na realidade, um nó não conhece obviamente todos os super-nós do sistema para que possa escolher um aleatoriamente, mas assume-se nesta análise que os nós que entram no sistema conhecem um pequeno subconjunto de todos os nós e escolhem um nó aleatório desse subconjunto. Numa implementação real do algoritmo, a escolha do pai por parte do filho deve ter em conta a compatibilidade entre os filtros dos nós.

De seguida apresentam-se os critérios utilizados durante as simulações para avaliar o algoritmo de super-nós.

```
onReceive(src, msg, PARENT_REQUEST):
if childrenList is full then
   msg_send(src, PARENT_REPLY_NOT_OK)
else if childrenList is not full OR msg.is_forced then
   if p(childrenList.size) ≤ rand() then
      promoteOneChild()
   end if
   addNewChild(src)
   msg_send(src, PARENT_REPLY_OK)
end if
```

Figura 6.5: Algoritmo executado por um super-nós ao receber uma mensagem do tipo PARENT_REQUEST de um nó a pedir para ser filho. A função p(X) corresponde à função representada pela expressão (6.1).

6.3 Critérios utilizados nas simulações com super-nós

Para avaliar o algoritmo de super-nós desenvolvido foram utilizados vários critérios durante as simulações realizadas. Estes critérios são apresentados na tabela 6.2. Esta tabela está desenhada segundo duas dimensões fundamentais e relevantes para esta análise:

1. Os critérios utilizados para seleccionar um nó semente de entre todos os super-nós possíveis. Um novo nó ao entrar na rede deve encontrar um novo pai para ser seu super-nó, foram considerados dois critérios para escolher este super-nó semente:

   Escolha do nó semente com base na informação perfeita em que o novo nó sabe sempre se existe um super-nó que o possa aceitar como filho e qual é o endereço desse super-nó;

   Escolha aleatória de um super-nó semente em que o filho tenta vários super-nós até que um o aceite como filho.

2. Os critérios utilizados pelos super-nós para seleccionar um filho de entre todos os seus filhos, quando pretende promover um nó. Foram testados diferentes critérios para fazer esta selecção:

   Mais tempo Critério em que é escolhido o nó com melhor tempo de vida futura para ser promovido. Este critério não é realista pois baseia-se numa informação que não existe num contexto de implementação real e serve assim para estabelecer um limite superior dos ganhos que podem ser obtidos.

   Menos tempo Análogo ao critério anterior, utilizando este critério é escolhido o nó com pior tempo de vida futura. Este critério serve para estabelecer um limite inferior dos ganhos que podem ser obtidos.

Aleatório Utilizando este critério é escolhido um nó aleatoriamente, de entre todos os filhos do super-nó, para ser promovido.

Mais antigo Neste caso é escolhido o filho mais antigo do super-nó para ser inserido na rede de super-nós.

Nem todas as combinações de critérios segundo estas duas dimensões são relevantes para a análise do algoritmo de super-nós e assim existem combinações marcadas na tabela 6.2 como não testados, que não foram testados em ambiente de simulação.

Critério de selecção de filhos	Método de escolha de nó semente	
	Entrada com semente aleatória	Entrada com informação perfeita
Mais tempo	Testado	Testado, Melhor caso
Menos tempo	Testado, Pior caso	Testado
Mais antigo	Testado	Não testado
Aleatório	Testado	Não testado

Tabela 6.2: Critérios utilizados durante as simulações com o algoritmo de super-nós.

A combinação de critérios em que se utiliza entrada com informação perfeita na escolha do super-nó semente e selecção do filho com mais tempo de vida futura para ser promovido representa o melhor caso possível e representa o limite superior dos ganhos que podem ser obtidos através da afinação e escolha de critérios realistas. Não é possível num sistema real implementar tais critérios, no entanto, os valores obtidos através de critérios realistas nunca serão melhores do que os valores estabelecidos através desta combinação de critérios. Ao utilizar um critério em que os nós dispõem de informação perfeita ao entrar na rede, está-se a isolar o custo dos vários critérios de promoção de filhos utilizados.

Da mesma forma, a combinação de critérios em que o nó semente é escolhido aleatoriamente e é sempre seleccionado o nó filho com menos tempo de vida futura para ser promovido corresponde ao pior caso estudado, estabelecendo assim o limite inferior para os ganhos obtidos através da afinação dos critérios utilizados. Tal como no caso em que é seleccionado o nó filho com melhor tempo de vida, este critério não pode ser implementado num sistema real em que não se dispõe de informação perfeita, mas contribui para que se possa avaliar o comportamento dos critérios realistas estabelecendo os minímos de melhoramentos que se podem obter.

No que se segue são apresentados os resultados obtidos através da simulação do sistema utilizando o algoritmo de super-nós descrito configurado com as combinações de critérios representadas como testadas na tabela 6.2.

6.4 Critérios de selecção de nós para promoção

Nesta secção são analisadas algumas simulações de acordo com os critérios já referidos em 6.3. Em alguns casos, são utilizados critérios que utilizam mecanismos que fornecem informação privilegiada sobre o sistema para promover os seus filhos. Estes mecanismos consistem na utilização de uma forma de um nó que entre no sistema saiba sempre se existe algum super-nó com capacidade para ser seu pai, isto é, um super-nó que seja responsável por um número de nós inferior ao máximo permitido. Neste caso, a rede de super-nós estabiliza em relação ao número de super-nós até ao momento em que todos os super-nós são responsáveis pelo número máximo de filhos e portanto são forçados a promover filhos para que o sistema possa crescer.

Para além do mecanismo já referido em que um nó que entra sabe sempre quais os super-nós que o podem aceitar como filho, caso existam, os critérios são testados utilizando uma forma mais realista de entrada em que os super-nós semente são escolhidos aleatoriamente.

As simulações expostas apresentam as mesmas métricas utilizadas anteriormente para que seja possível fazer uma comparação entre os algoritmos já considerados.

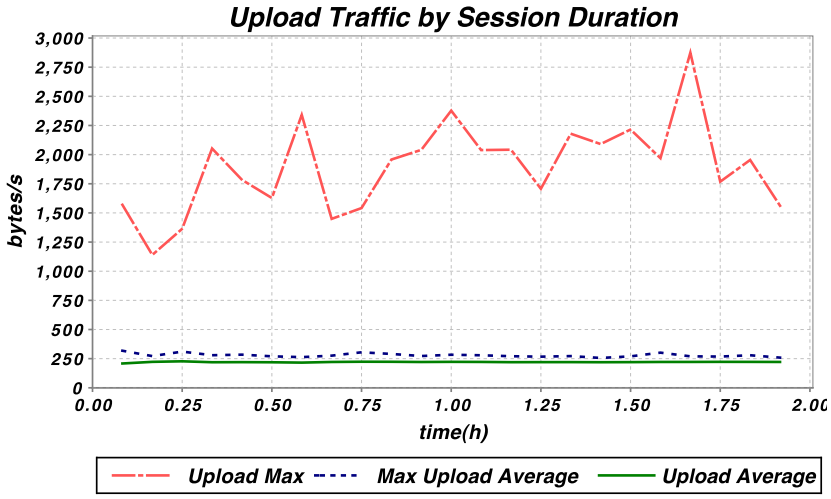
6.4.1 Promoção do filho com maior tempo de vida futura

Nesta secção são apresentados os resultados provenientes da utilização de um critério em que o super-nó sabe sempre qual dos seus filhos vai ficar no sistema durante mais tempo, promovendo esse nó. Quanto mais tempo um super-nó estiver dentro do sistema, menor será o *churn* observado na rede de super-nós, uma vez que quando um nó sai do sistema, todos os seus filhos vão procurar um novo pai, o que pode levar à promoção de um ou vários nós. No que se segue é analisado o comportamento do algoritmo de super-nós quando se utiliza este critério para a escolha do filho a promover perante dois casos distintos. O caso em que cada nó ao entrar no sistema tem conhecimento da existência ou não existência de um super-nó que possa ser seu pai e o caso em que um nó deve contactar super-nós sementes até encontrar um super-nó que aceite ser seu pai.

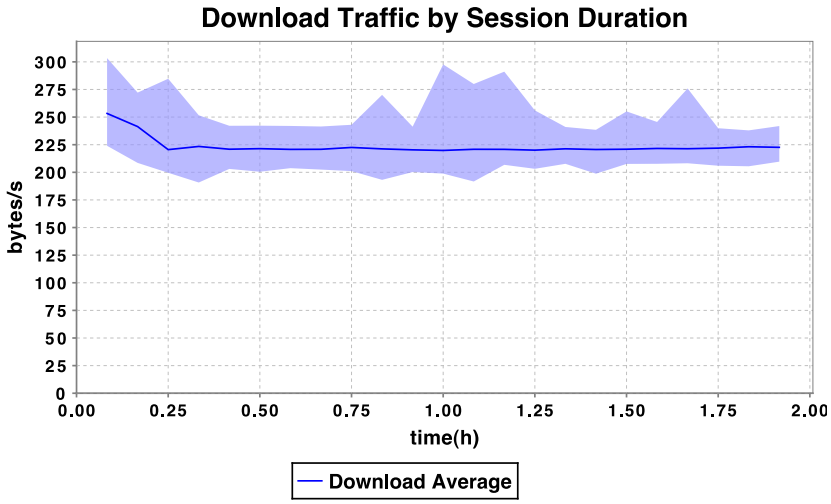
6.4.1.1 Informação perfeita sobre nós semente com promoção do filho com maior tempo de vida futura

O conhecimento por parte dos super-nós sobre a esperança de vida dos seus filhos, juntamente com a informação privilegiada sobre as vagas nos super-nós que os filhos têm ao entrar no sistema leva a que este caso corresponda a uma situação ideal, estabelecendo um limite superior para os ganhos que é possível obter através da afinação do algoritmo de super-nós.

Como se pode observar pela comparação das figuras 6.6 (a) e 6.7 (a) com as figuras 5.4 (a)



(a) Tráfego de *upstream*

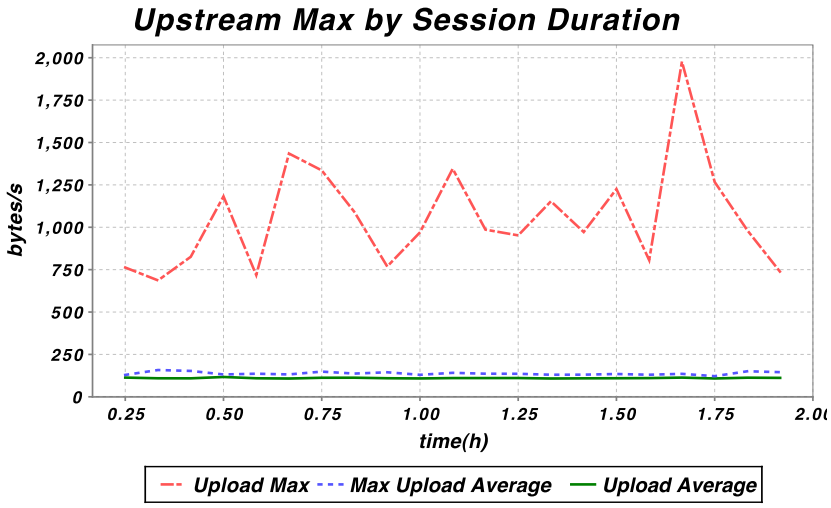


(b) Tráfego de *downstream*

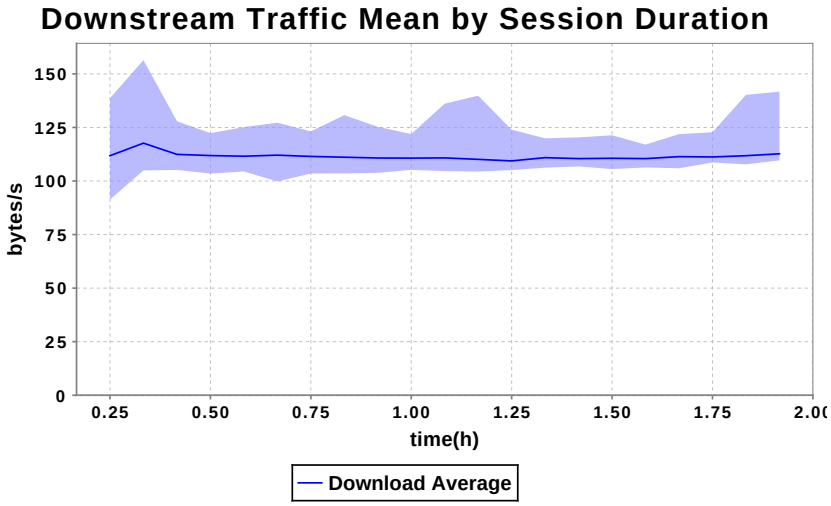
Figura 6.6: Resultados obtidos perante condições óptimas de selecção de nós semente e selecção de nós a promover. Com $\lambda_{entrada} = 1/4 \times 100$ o que equivale a um valor esperado de 4 chegadas por segundo e os parâmetros da tabela 6.1

e 5.3 (a) do capítulo 5, o valor do tráfego de *upstream* utilizado pelos nós desceu significativamente na ordem de grandeza de 8 vezes, pois passa a depender linearmente do número de entradas por segundo na rede de super-nós e não do número de entradas por segundo em todo o sistema.

Em relação aos valores observados nos gráficos referentes ao tráfego de *downstream* (figuras 6.6 (b) e 6.7 (b)) é possível verificar que se observa uma descida na mesma proporção, isto é, o valor para o tráfego médio continua a ser sensivelmente o mesmo para o tráfego de *downstream* e *upstream* e em média ao fim de algum tempo, a quantidade de informação recebida é igual à quantidade de informação enviada.



(a) Tráfego de *upstream*



(b) Tráfego de *downstream*

Figura 6.7: Resultados obtidos perante condições óptimas de selecção de nós semente e selecção de nós a promover. Com $\lambda_{entrada} = 1/2 \times 100$ o que equivale a um valor esperado de 2 chegadas por segundo e os parâmetros da tabela 6.1

Pelas simulações realizadas utilizando os critérios já referidos, é possível concluir que a introdução de uma hierarquia deste tipo no sistema contribui de forma positiva para a diminuição do *churn* observado pelos super-nós. Em relação às simulações anteriormente consideradas em que o tráfego observado podia ser representado por $b_u = r \times m$, podemos verificar que existe uma significativa descida do tráfego de *upstream* e *downstream* com que cada nó contribuiu. Antes de se introduzir o mecanismo de super-nós, no caso em que $r = 4$ Eventos/Segundo, era esperado que os nós contribuíssem com $b_u = 4 \times 500 = 2000$ B/s. Como se pode observar pela figura 6.6, introduzindo o mecanismo de super-nós, cada super-nó passa a contribuir com menos de 250 B/s. Ao utilizar $r = 2$ Eventos/Segundo, obtém-se um melhoramento na mesma proporção, como se pode ver na figura 6.7.

Com estas simulações, é possível estabelecer um limite para os melhoramentos que se podem obter através da utilização de uma estrutura de super-nós para o algoritmo de filiação do LiveFeeds. Os resultados obtidos com estas simulações mostram que não é possível obter um melhoramento melhor do que a diminuição em cerca oito vezes do tráfego de *upstream* e *downstream*, quando o número de filhos máximo é igual a 6, pois neste caso foi utilizada uma forma de os nós usufruírem de informação privilegiada. Este tipo de cenário não é realista numa implementação real do LiveFeeds sendo útil para aferir qual o limite superior dos melhoramentos que é possível obter através da utilização dos diferentes critérios que se enumeram neste capítulo.

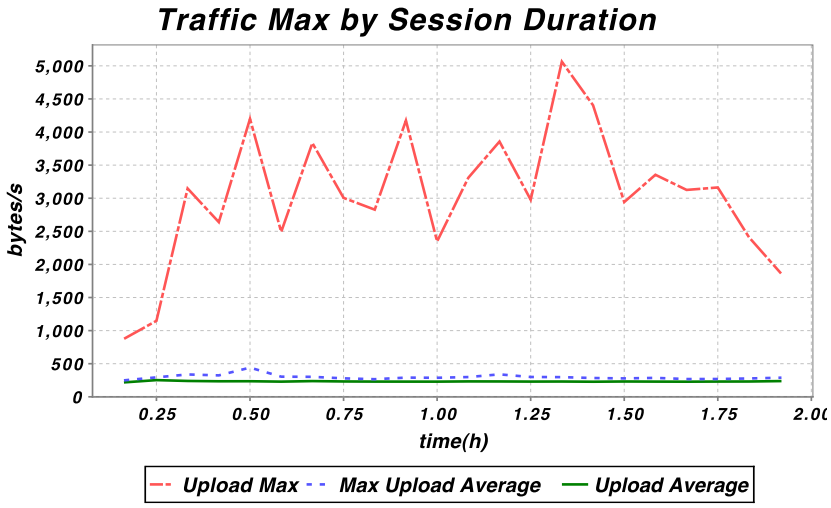
6.4.1.2 Escolha aleatória de nós semente com promoção do filho com maior tempo de vida futura

Nesta fase foi implementada uma forma de retirar a informação perfeita que um novo nó tinha ao entrar no sistema, isto é, ao entrar no sistema, um nó deve procurar um super-nó que possa ser seu pai. Ao fim de algumas tentativas o nó pode forçar a sua entrada como filho o que pode levar a uma promoção de um outro filho para super-nó caso seja necessário.

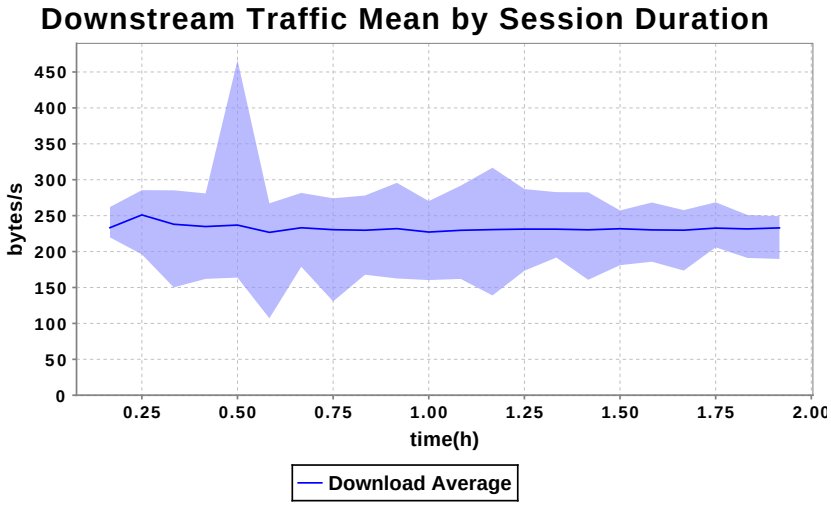
A figura 6.8 mostra os resultados obtidos ao utilizar esta forma de entrada e um critério de promoção em que é sempre escolhido o filho com maior tempo de vida futura. Comparando os gráficos das figuras 6.6 e 6.8, é possível verificar que o valor do custo em termos de capacidade de *upstream* e *downstream* é muito semelhante, donde se pode concluir que a introdução de um método realista de escolha do super-nó semente não tem um impacto significativo nos valores dos custos do algoritmo de super-nós, existindo apenas um aumento dos valores de "Upload Max" mas não dos valores de "Max Upload Average", os critérios de promoção de filhos são assim o aspecto mais importante na afinação do algoritmo e onde se pode obter maiores ganhos em termos dos custos de comunicação.

6.4.2 Promoção do filho com menor tempo de vida futura

Tal como na sub-secção anterior, dividimos aqui as simulações em dois casos distintos: O caso em que os nós semente são escolhidos utilizando informação privilegiada e o caso em que os nós semente são escolhidos aleatoriamente. As simulações desta secção têm no entanto em comum o critério utilizado para escolher o nó filho a promover, é sempre escolhido o filho com menor tempo de vida futura.



(a) Tráfego de *upstream*



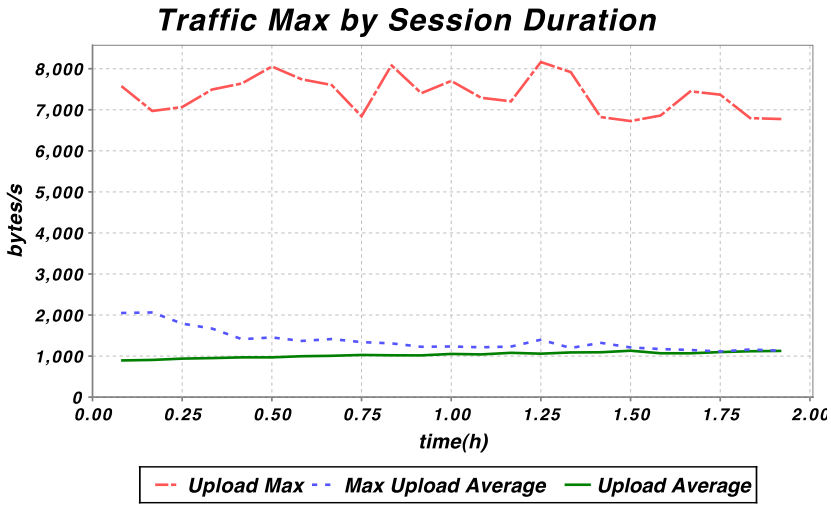
(b) Tráfego de *downstream*

Figura 6.8: Resultados obtidos utilizando escolha aleatória de nós semente com escolha do melhor filho na altura da promoção ao utilizar $\lambda_{entrada} = 1/4 \times 100$ o que equivale a um valor esperado de 4 chegadas por segundo e os parâmetros da tabela 6.1

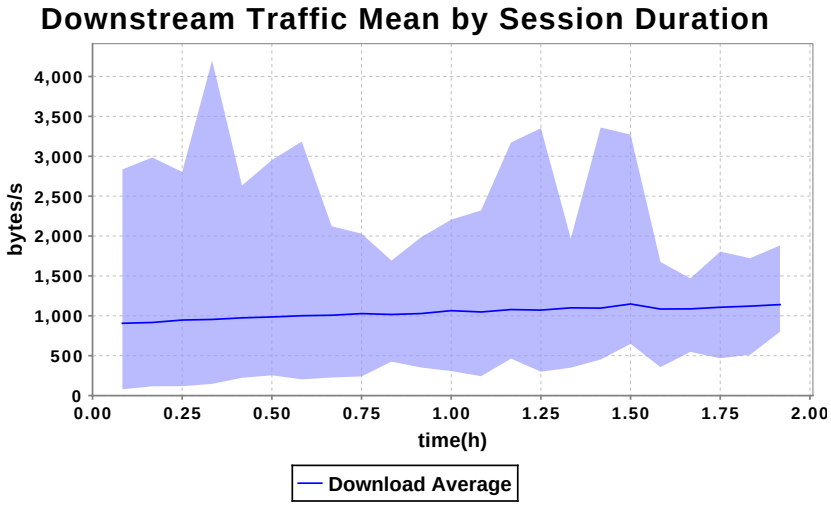
6.4.2.1 Informação perfeita sobre nós semente com promoção do filho com menor tempo de vida futura

Neste caso, mantém-se a informação perfeita por parte de um nó que entra no sistema e procura um pai, mas os super-nós seleccionam sempre o nó que irá ficar menos tempo no sistema. Este método de escolha dos filhos a promover introduz, como se verá, alguma instabilidade no sistema pois ao promover os nós que irão ficar menos tempo como super-nós aumenta-se a taxa de entradas por segundo observadas dentro da rede de super-nós visto que a procura de novos pais por parte dos seus filhos pode provocar a entrada de outros nós na rede de super-nós. Esta subida da taxa de entradas provoca, como já foi referido, um aumento do tráfego de *upstream* e *downstream*. Através da utilização deste critério, oposto ao considerado na secção

anterior, é possível aferir quais os melhoramentos minímos que podem ser obtidos através da utilização de outros critérios de selecção de filhos mais realistas do que uma escolha com base em informação perfeita sobre a vida futura dos nós filhos de um super-nó.



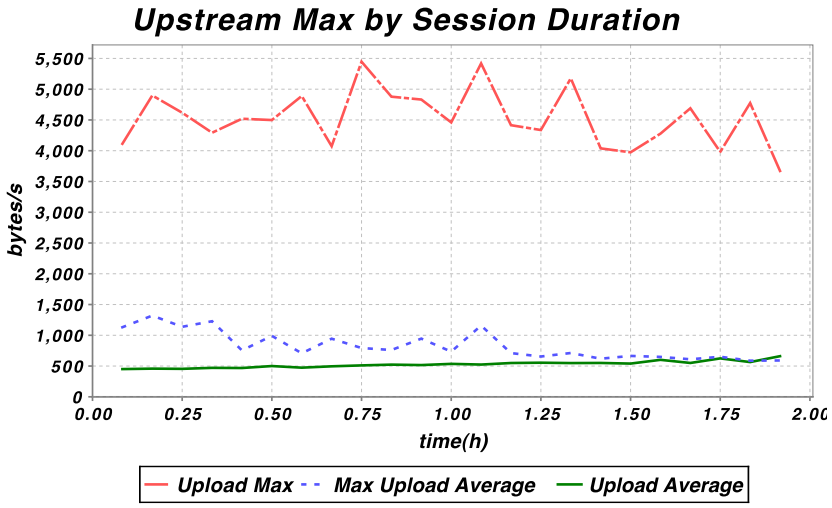
(a) Tráfego de *upstream*



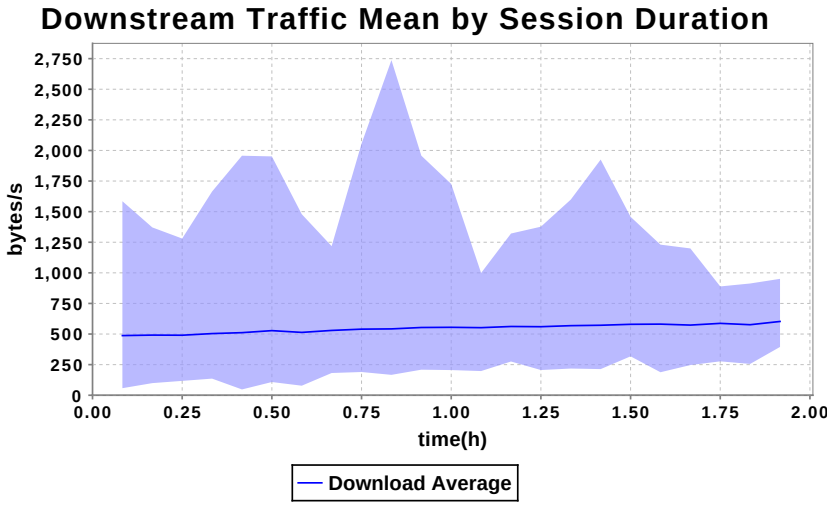
(b) Tráfego de *downstream*

Figura 6.9: Resultados obtidos perante condições óptimas de selecção de nós semente e com escolha do pior filho na altura da promoção. Utilizando $\lambda_{entrada} = 1/4 \times 100$ o que equivale a um valor esperado de 4 chegadas por segundo e os parâmetros da tabela 6.1

Como se pode observar nas figuras 6.9 e 6.10, comparando os resultados com os valores obtidos na secção anterior, existe um claro aumento do valor do tráfego de *upstream* e *downstream* despendido pelos nós. Utilizando este critério obtém-se uma diminuição para apenas cerca de metade do tráfego despendido sem recorrer à utilização de super-nós. Podemos assim concluir, através destas simulações, que o critério de escolha de um dos nós do conjunto de filhos de um super-nó é de extrema importância para o comportamento do algoritmo de filiação do LiveFeeds. Podemos também concluir que utilizando o pior critério possível, obtém-se mesmo



(a) Tráfego de *upstream*



(b) Tráfego de *downstream*

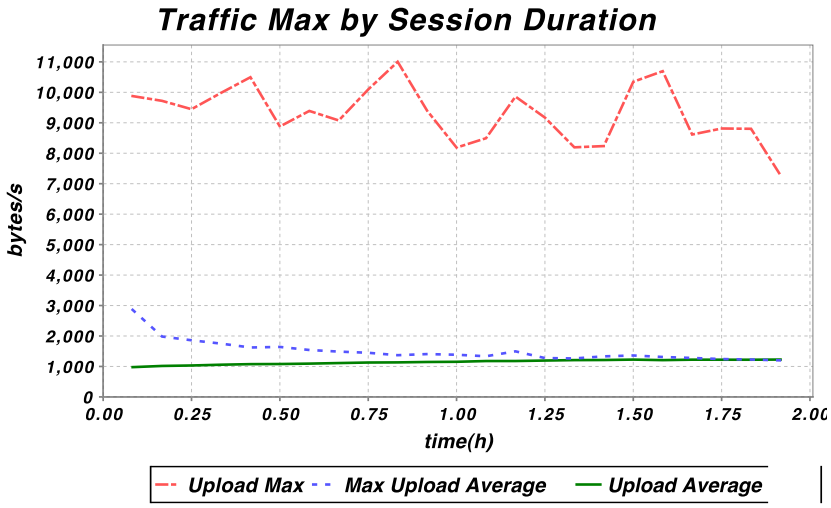
Figura 6.10: Resultados obtidos perante condições óptimas de selecção de nós semente e com escolha do pior filho na altura da promoção. Utilizando $\lambda_{entrada} = 1/2 \times 100$ o que equivale a um valor esperado de 2 chegadas por segundo e os parâmetros da tabela 6.1

assim uma diminuição do tráfego face à utilização do algoritmo de visibilidade completa estudado nos capítulos anteriores, implementando um bom critério de selecção do nó a promover espera-se assim uma melhoria nunca inferior a metade do tráfego de *upstream* e *downstream* anteriormente despendido.

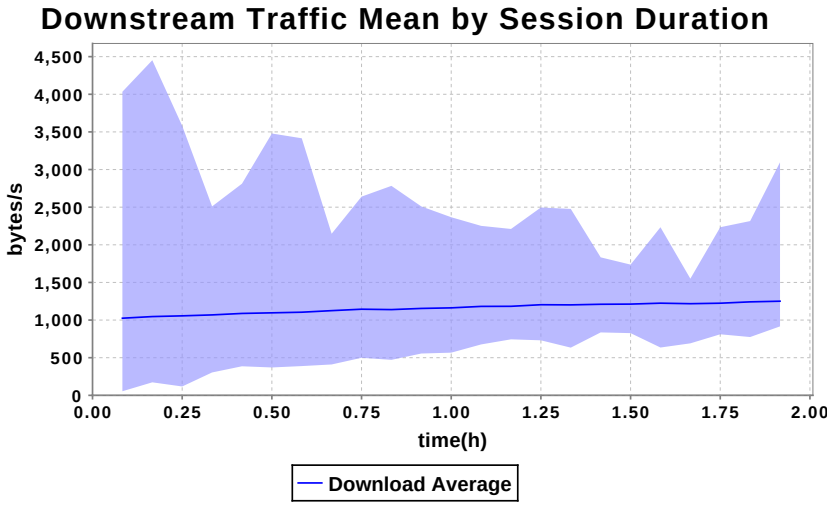
6.4.2.2 Escolha aleatória de nós semente com promoção do filho com menor tempo de vida futura

O caso em que os nós ao entrar no sistema devem tentar encontrar de forma aleatória um nó semente que possa ser seu pai é aqui apresentado para que se possam fazer algumas comparações entre diferentes critérios. A figura 6.11 mostra os resultados obtidos ao utilizar esta forma de

entrada e um critério de promoção em que é sempre escolhido o filho com pior tempo de vida futura. Ao utilizar este critério é possível determinar o mínimo de ganhos que se podem obter quando se utiliza um método de entrada realista.



(a) Tráfego de *upstream*



(b) Tráfego de *downstream*

Figura 6.11: Resultados obtidos utilizando escolha aleatória de nós semente e com escolha do pior filho na altura da promoção. Utilizando $\lambda_{entrada} = 1/4 \times 100$ o que equivale a um valor esperado de 4 chegadas por segundo e os parâmetros da tabela 6.1

6.4.3 Promoção do filho mais antigo

O critério de selecção aqui analisado consiste na promoção do nó que seja filho do super-nó há mais tempo. A ideia por detrás deste critério tem por base a convicção de que quanto mais tempo um utilizador estiver no sistema maior é a probabilidade de permanecer durante mais

tempo no sistema, se um nó for o filho que está há mais tempo no sistema, tem maior probabilidade de ficar mais tempo no sistema do que os outros filhos. No entanto, o modelo de simulação utilizado não leva em linha de conta este tipo de comportamento por parte dos utilizadores e assim é difícil de quantificar os melhoramentos que seria possível obter através do uso desta estratégia para seleccionar os nós a promover. Tal só seria possível se o modelo de simulação levasse em linha de conta informação sobre o comportamento dos utilizadores em sessões anteriores. Apesar disso, foram realizadas algumas simulações para testar o comportamento do sistema perante o referido critério. Como se pode observar pela figura 6.12 existe um melhoramento face às simulações que utilizavam um critério em que era escolhido o nó que se sabia ter menor tempo de vida futura, isto é, mesmo considerando que este critério não foi testado em condições que seriam óptimas neste caso, este critério de promoção é melhor do que o pior caso possível em que é escolhido o pior de todos os filhos, mesmo com escolha aleatória de nós semente, o que leva a crer que existe um bom potencial de melhoramento do algoritmo de super-nós.

6.4.4 Promoção aleatória de um filho

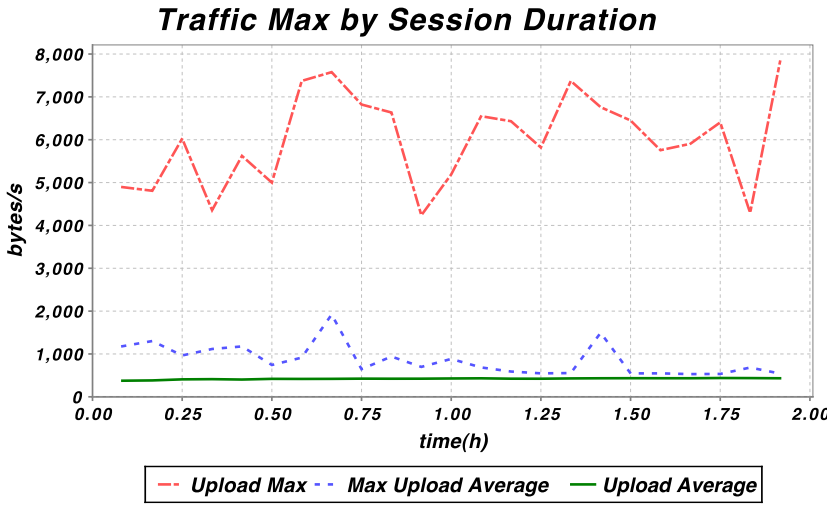
Na figura 6.13 são apresentados os resultados obtidos com simulações em que a escolha do nó semente é feita aleatoriamente e em que os filhos a promover são seleccionados aleatoriamente pelos super-nós. Pode-se verificar que existiu novamente um ganho no que se refere ao tráfego de *upstream* e *downstream* que cada nó despendeu para difundir a informação de filiação dos novos nós em comparação com o algoritmo de visibilidade completa.

De entre os critérios aqui analisados que podem ser implementados num cenário realista, a promoção aleatória de nós filhos parece ser a melhor opção. Como se pode verificar através da comparação dos gráficos referentes a este critério com os gráficos obtidos na secção 6.4.3, confirma-se que ao utilizar o critério de escolha aleatória de nós filhos se obtêm melhores resultados do que ao utilizar o critério de escolha do filho mais antigo. A diferença é no entanto muito diminuta e é preciso ter em conta que o método de escolha do filho mais antigo foi avaliado em condições longe de óptimas uma vez que o tempo de vida futura de um nó no simulador não tem qualquer relação com o tempo de vida passada do mesmo.

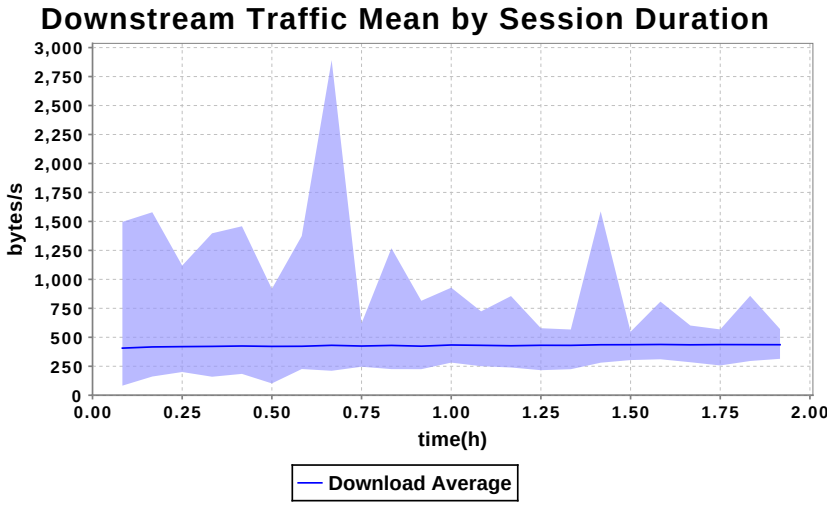
6.4.5 Conclusões sobre os critérios de promoção de filhos

Observando as simulações apresentadas neste capítulo, é possível tirar algumas conclusões sobre os critérios de escolha de filhos a promover e em relação ao método de escolha de super-nós semente.

Apesar de apenas ter sido equacionada uma forma realista de escolha de nós semente, a



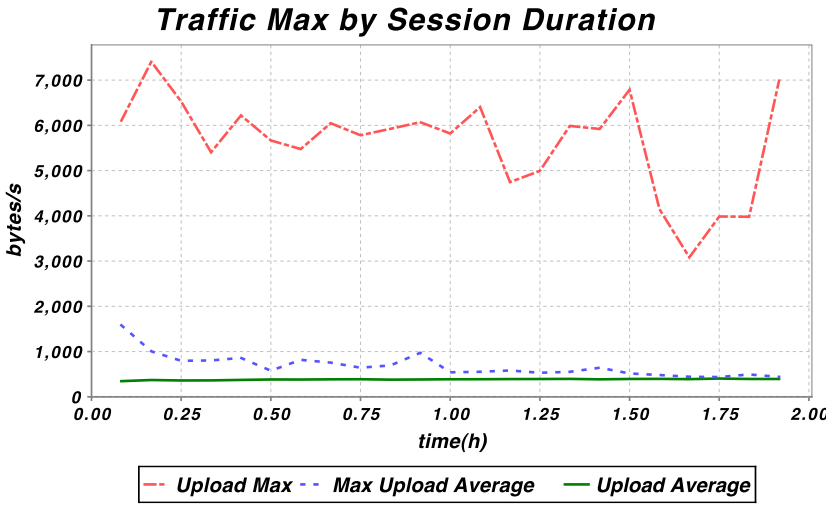
(a) Tráfego de *upstream*



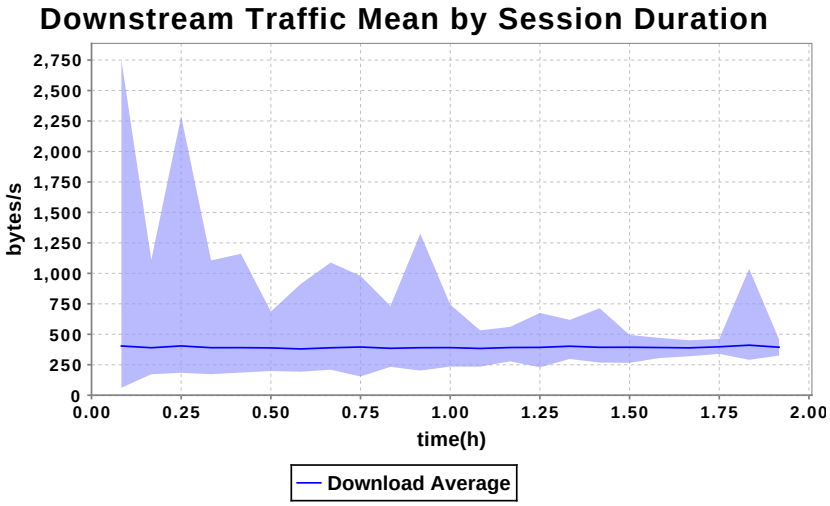
(b) Tráfego de *downstream*

Figura 6.12: Resultados obtidos utilizando escolha aleatória de nós semente e escolha do nó que é filho há mais tempo para ser promovido ao utilizar $\lambda_{entrada} = 1/4 \times 100$ o que equivale a um valor esperado de 4 chegadas por segundo e os parâmetros da tabela 6.1

escolha aleatória, é possível concluir que o uso deste tipo de aproximação não introduz penalizações significativas ao valor de tráfego de *upstream* e *downstream* despendido pelos nós do sistema face à situação irrealista em que se utiliza uma forma de entrada com informação perfeita sobre a disponibilidade dos super-nós. Prova disso é a comparação entre as duas simulações que utilizem o mesmo critério de selecção de filhos mas formas de entrada diferentes, com informação perfeita ou escolha aleatória (Secções 6.4.1 e 6.4.2). No caso em que foi testado o critério de promoção do filho com melhor tempo de vida futura (Secção 6.4.1) , a média do tráfego máximo de *upstream* despendido ("Max Upload Average") sobe de cerca de 250 B/s para um valor mais próximo dos 350 B/s, existindo no entanto um aumento não muito acentuado dos valores máximos observados ("Upload Max").



(a) Tráfego de *upstream*



(b) Tráfego de *downstream*

Figura 6.13: Resultados obtidos utilizando escolha aleatória de nós semente e escolha aleatória do filho a promover, utilizando $\lambda_{entrada} = 1/4 \times 100$ o que equivale a um valor esperado de 4 chegadas por segundo e os parâmetros da tabela 6.1

Estabelecida a subida de custos proveniente da utilização de uma forma realista de entrada, foram analisadas algumas simulações de forma a comparar os diversos critérios equacionados de selecção de filhos para promoção. Através desta análise, e nas condições de simulação já referidas, pode-se concluir que a escolha aleatória de filhos é a melhor opção. Este critério obtém resultados muito melhores do que a escolha do pior filho possível. De facto, é possível verificar que o valor do tráfego médio de *upstream* raramente é superior a 1000 B/s enquanto que no caso em que promove o pior nó possível esse valor se aproxima dos 2000 B/s. Este critério está no entanto um pouco longe do melhor possível, que foi estabelecido ao simular o uso da promoção do nó com melhor tempo de vida futura (secção 6.4.1), com um tráfego médio de *upstream* pouco superior a 250 B/s, tendo-se conseguido com o critério de promoção

aleatória um tráfego médio de *upstream* pouco superior a 500 B/s.

Os resultados obtidos utilizando o critério de selecção de filhos em que é seleccionado o filho mais antigo é no entanto promissor, uma vez que foi analisado em condições não óptimas e tendo conseguido mesmo assim um desempenho semelhante ao observado quando se utiliza uma selecção aleatória dos nós filhos para promoção. Este critério pode vir a oferecer resultados melhores do que a selecção aleatória caso se confirme que o comportamento dos utilizadores é consistente com a ideia de que quanto mais tempo um utilizador permanecer no sistema maior é a probabilidade de continuar no sistema por mais algum tempo. A tabela 6.3 sumariza os resultados obtidos pela análise das simulações realizada. Os valores da tabela são apresentados em termos dos ganhos obtidos e com base no valor de referência de 2000 B/s, que foi observado nas simulações em que não é utilizado o algoritmo de super-nós proposto neste capítulo.

Critério de selecção de filhos	Método de escolha de nó semente	
	Entrada com semente aleatória	Entrada com informação perfeita
Mais tempo	8×	8×, Melhor caso
Menos tempo	1,8×, Pior caso	2×
Mais antigo	5×	-
Aleatório	5×	-

Tabela 6.3: Diminuição obtida dos valores de "Upload Average" com a utilização dos diversos critérios e verificados pela observação dos gráficos das figuras apresentadas, com $\lambda_{chegadas} = 1/4 \times 100$ o que equivale a um valor esperado de 4 entradas/segundo.

6.5 Delegação de trabalho para os filhos

Utilizando o algoritmo analisado anteriormente, um super-nó fica responsável por um conjunto de filhos, o que leva a que os super-nós necessitem de despender de uma maior capacidade de *upstream* para que o algoritmo de super-nós possa funcionar correctamente, pois os filhos ficam dependentes dos seus super-nós para receberem mensagens referentes à difusão de notificações. Para reduzir a desvantagem que um super-nó tem ao ser responsável por outros nós, o trabalho de envio de mensagens a que um super-nó fica sujeito quando é escolhido para nó interior da árvore de difusão pode ser feito por um dos seus filhos. Nesta secção é proposto e analisado um melhoramento ao algoritmo de super-nós já apresentado que consiste na utilização de filhos para reencaminhar as mensagens no caso em que um super-nó é escolhido para ser nó interior da árvore de difusão.

Utilizando este mecanismo de delegação de trabalho para os filhos, ao receber uma mensagem com informação de filiação, um super-nó verifica se deve encaminhar essa mensagem para outros nós, caso em que é nó interior na árvore de difusão, ou se a mensagem não deve ser

reencaminhada, no caso em que o super-nó é folha na árvore de difusão. Quando um super-nó tem filhos e é escolhido para nó interior, necessita apenas de enviar uma mensagem para um dos seus filhos, contendo uma lista de nós para onde reencaminhar a mensagem e a informação de filiação a difundir. A capacidade de rede despendida pelos super-nós escolhidos para nó interior passa assim a ser apenas de uma mensagem enviada e não de G mensagens enviadas, como acontecia anteriormente. Esta diminuição do número de mensagens enviadas por super-nós interiores, e consequente diminuição da capacidade de rede de *upstream* despendida, leva a que exista uma vantagem em ser super-nó, passando a existir um incentivo forte para um nó filho querer tornar-se super-nó. Por outro lado, existe também um grande incentivo para que os super-nós sejam responsáveis por alguns filhos, pois esses filhos podem realizar trabalho útil que de outra maneira teria de ser desempenhado pelo super-nó. A combinação destes incentivos, poderá levar a que exista um aumento da rede de super-nós e consequente aumento do número total de filhos que podem ser alojados pela rede de super-nós, o que leva ao aumento do número total de nós dentro de todo o sistema distribuído.

Na secção 6.6 são apresentadas as simulações realizadas depois de implementado este melhoramento ao algoritmo inicial. Na secção 6.7 são analisados os resultados obtidos através das mesmas.

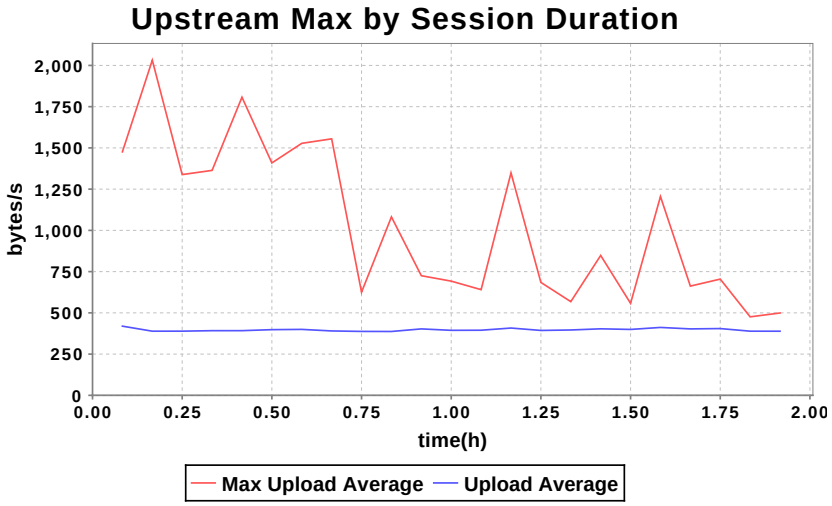
6.6 Simulações com delegação de trabalho para os nós filhos

De acordo com o método utilizado anteriormente para analisar o algoritmo de super-nós proposto, o simulador foi configurado para que o sistema utilizasse o melhoramento aqui proposto e foram recolhidas algumas simulações com o objectivo de quantificar os melhoramentos obtidos.

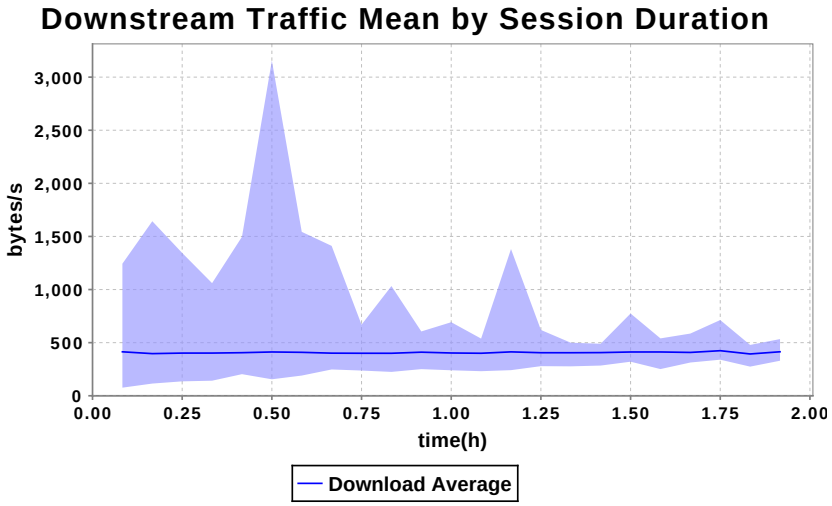
A métrica "Upload Max", utilizada nas simulações anteriores e que corresponde ao maior valor de banda passante de *upstream* alguma vez observado durante o tempo de amostragem, deixa de ser útil por corresponder à altura em que um super-nó não têm nós filhos e tem assim de enviar até G mensagens quando é escolhido para nó interior. Desta forma, nas simulações apresentadas seguidamente, apenas são utilizadas as métricas "Max Upload Average", que representa o valor máximo da média de banda passante despendida pelos nós e "Upload Average" que representa a média de banda passante despendida pelos nós. Estas métricas foram já apresentadas em maior detalhe na secção 5.2 do capítulo 5. A utilização de apenas estas duas métricas leva a que seja possível observar com maior detalhe os valores de "Max Upload Average" e "Upload Average", permitindo assim que se possa fazer uma análise mais rigorosa aos melhoramentos obtidos através da utilização do melhoramento estudado nesta secção.

Em primeiro lugar é apresentada uma simulação para estabelecer os valores de capacidade de *upstream* e *downstream* que os super-nós têm de despender quando não fazem qualquer

delegação de trabalho para os filhos. Foram utilizados os parâmetros das simulações realizadas nos capítulos anteriores, com $\lambda_{entrada} = 1/4 \times 100$, $G = 4$ e os parâmetros da tabela 6.1. Foram ainda utilizados os critérios de escolha aleatória de nós semente e escolha aleatória de filhos a promover.



(a) Tráfego de *upstream*



(b) Tráfego de *downstream*

Figura 6.14: Resultados obtidos com uma simulação em que não foi utilizada delegação de trabalho para os filhos

Como já foi apresentado anteriormente, a mudança do grau da árvore de difusão construída, G, não tem consequências na capacidade de rede despendida pelos nós quando não se utilizam os filhos para enviar mensagens atribuídas aos super-nós, pelo que apenas se apresentam os resultados de uma simulação com $G = 4$, onde se pode observar, pela figura 6.14, que os valores das duas métricas apresentadas referentes ao tráfego de *upstream* vão convergindo à medida que aumenta o tempo de sessão dos nós, tal resultado já foi discutido anteriormente. Pelo gráfico 6.14 (a) é ainda possível observar que o valor de "Upload Average" é de cerca

de 400B/s, enquanto o valor de "Max Upload Average" é de aproximadamente 500B/s, sendo estes valores referentes aos super-nós com maior tempo de sessão pois estes valores reflectem com maior exactidão os ganhos obtidos dado que o custo inicial decorrente de os super-nós não serem responsáveis por filhos no início da sua vida como super-nós já ter sido mais amortizado nos super-nós com maior tempo de sessão. Estes valores servem assim de referência para a análise dos ganhos obtidos através da utilização dos filhos para envio de mensagens com informação de difusão, pois são os valores de capacidade de rede de *upstream* que os super-nós têm de despender quando não podem utilizar os seus filhos para enviar mensagens quando os super-nós são escolhidos para nós interiores. Como já foi referido, espera-se uma diminuição de tráfego de *upstream* despendido, b_u, em função de G e B, segundo a expressão 6.2, em que G é o grau da árvore de difusão e B é o valor do tráfego de *upstream* despendido antes da implementação do mecanismo de delegação de trabalho para os filhos.

$$b_u = \frac{B}{G} \tag{6.2}$$

Uma vez definido um valor de referência, B, igual a 400B/s, através da simulação apresentada na figura 6.14, podemos definir a expressão 6.3, que representa os ganhos que seriam esperados teoricamente no sistema com os parâmetros apresentados ao utilizar os filhos para fazer algum trabalho que de outra maneira deveria ser feito pelos super-nós escolhidos para nós interiores.

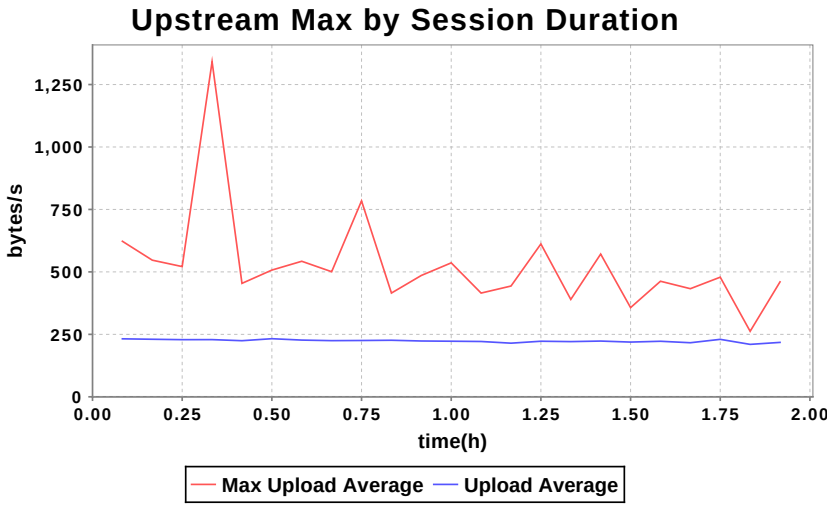
$$b_u = \frac{400}{G} \tag{6.3}$$

Nas figuras 6.15, 6.16, 6.17 e 6.18 são apresentadas simulações para diferentes valores de G, mas com os parâmetros anteriormente utilizados na simulação 6.14, o que permite avaliar os ganhos obtidos pela utilização do melhoramento que consiste na delegação de trabalho para os filhos.

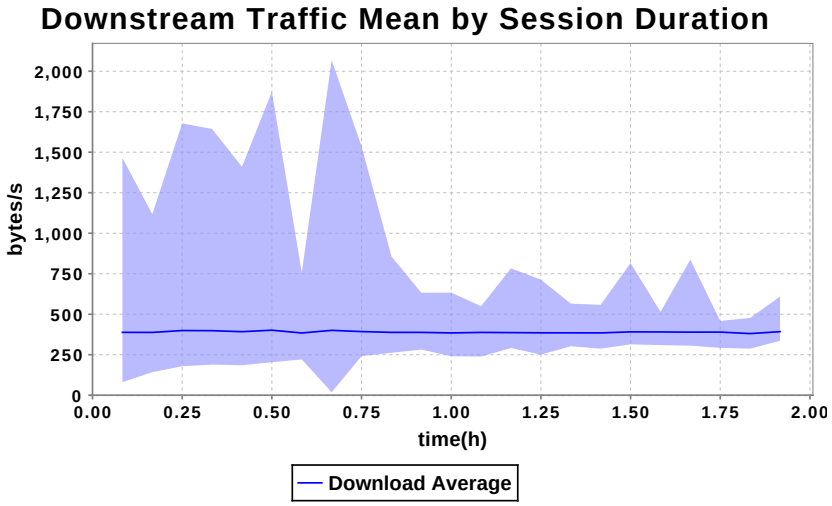
6.7 Resultados obtidos com delegação de trabalho para os filhos

Através das simulações apresentadas é possível retirar algumas conclusões. No início de vida de um super-nó, este não é responsável por nenhum nó filho, o que leva a que o super-nó necessite ele próprio de enviar G mensagens quando é escolhido para nó interior. Apesar de a percentagem global de super-nós sem filhos ser de aproximadamente 5%[1], todos os super-nós passam no início da sua promoção por um período em que não têm filhos e consequentemente têm de enviar mais mensagens do que o normal. Este factor explica o facto de ambas as métricas

[1]É possível obter este valor através da observação dos dados que são apresentados pelo simulador, tal como descrito na secção 1.3 do capítulo 1



(a) Tráfego de *upstream*

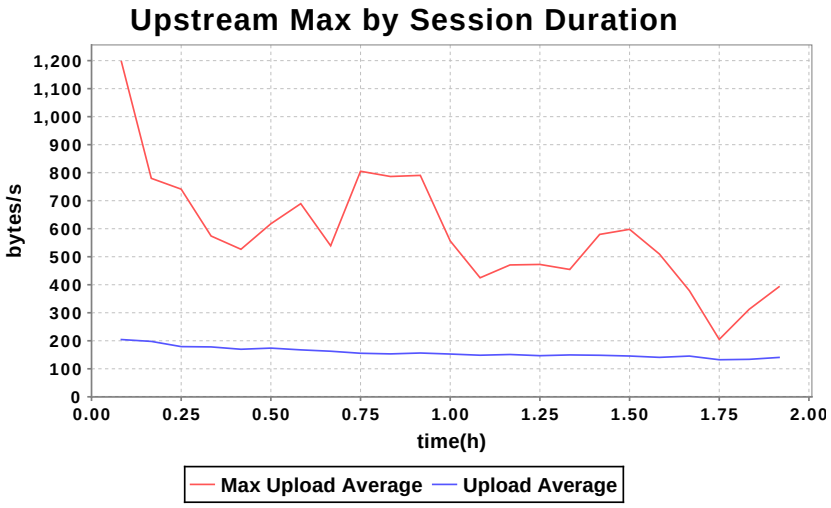


(b) Tráfego de *downstream*

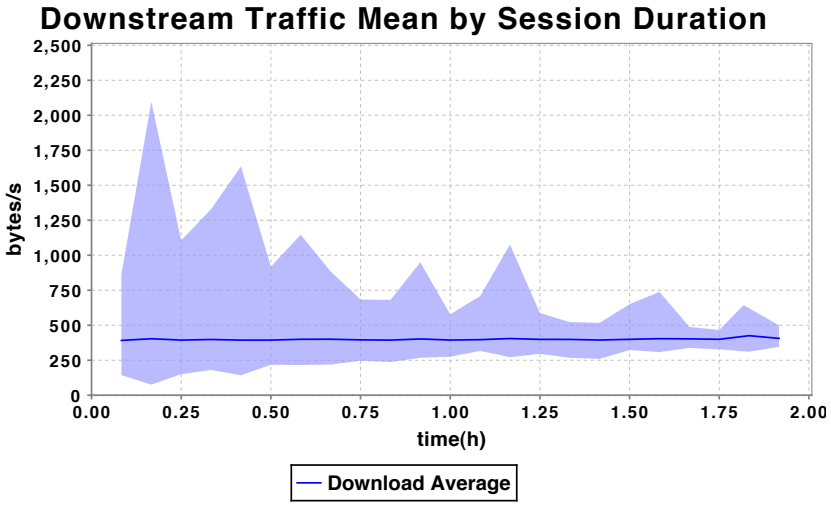
Figura 6.15: Resultados obtidos com delegação de trabalho para os filhos com $G = 2$

analisadas, "Max Upload Average" e "Upload Average" diminuírem com o aumento do tempo de sessão dos super-nós, pois quanto mais tempo um super-nó permanecer como super-nó, mais tempo tem para amortizar o custo em termos de capacidade de *upstream* com que necessitou de contribuir antes de ter pelo menos um filho. Note-se ainda que quanto maior for o grau, maior será esta capacidade requerida inicialmente.

Ainda através da análise das simulações, é possível desenhar o gráfico da figura 6.19, que apresenta duas curvas de onde se podem tirar algumas conclusões. A curva "Upload Average Real" corresponde aos valores observados nas simulações em relação à métrica "Upload Average". Tal como anteriormente, foram considerados os valores de "Upload Average" dos super-nós com maior tempo de sessão pois estes valores reflectem com maior exactidão os ganhos obtidos. A curva "Upload Average Teórico" corresponde aos valores obtidos através da expressão (6.3), $b_u = \frac{400}{G}$ e oferece assim uma base de comparação entre os ganhos realmente



(a) Tráfego de *upstream*

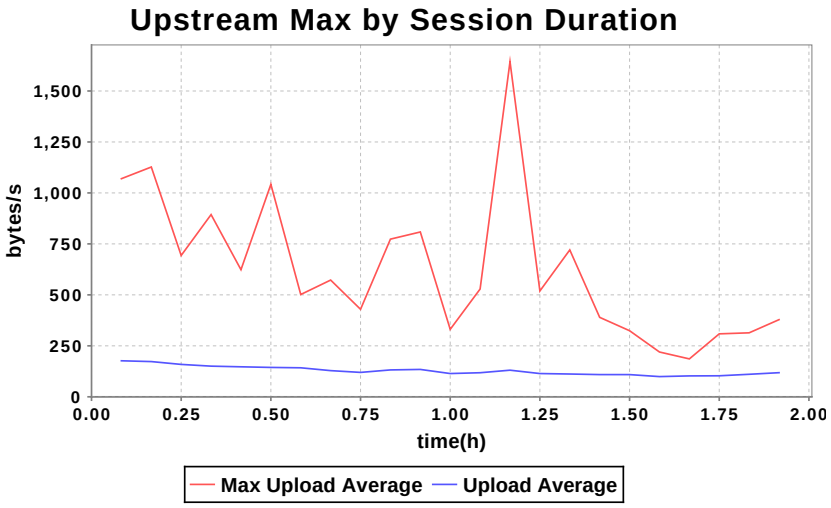


(b) Tráfego de *downstream*

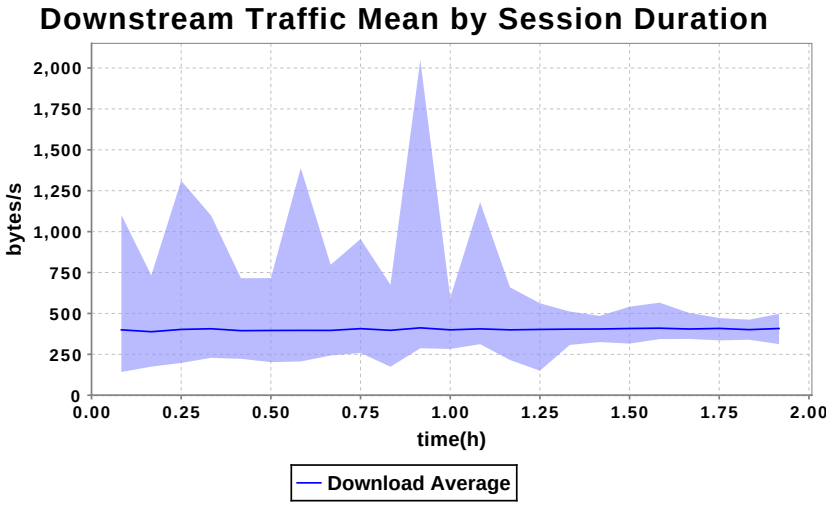
Figura 6.16: Resultados obtidos com delegação de trabalho para os filhos com $G = 4$

obtidos e os ganhos que se esperavam obter.

Como se pode observar pela figura 6.19, ambas as curvas tem a mesma forma, o que indica que os ganhos obtidos nas simulações configuram uma curva semelhante ao que seria de esperar quando o algoritmo é examinado de forma analítica. No entanto, os valores da curva "Upload Average Real" são maiores do que os valores esperados, representados pela curva "Upload Average Teórico", esta diferença pode ser justificada, como já foi dito, pelo facto de no início da sua vida como super-nós, estes terem de contribuir com uma maior capacidade de rede de *upstream*, pois não têm filhos e não podem assim delegar trabalho. De facto, quanto maior é o grau da árvore da difusão, G, maior é a diferença entre as curvas de "Upload Average Real" e "Upload Average Teórico", com o aumento de G, aumenta a banda passante de *upstream* que um super-nó necessita de utilizar quando é escolhido para nó interior da árvore de difusão e não tem filhos para o substituírem nas suas funções, o que leva a que a amortização deste custo



(a) Tráfego de *upstream*

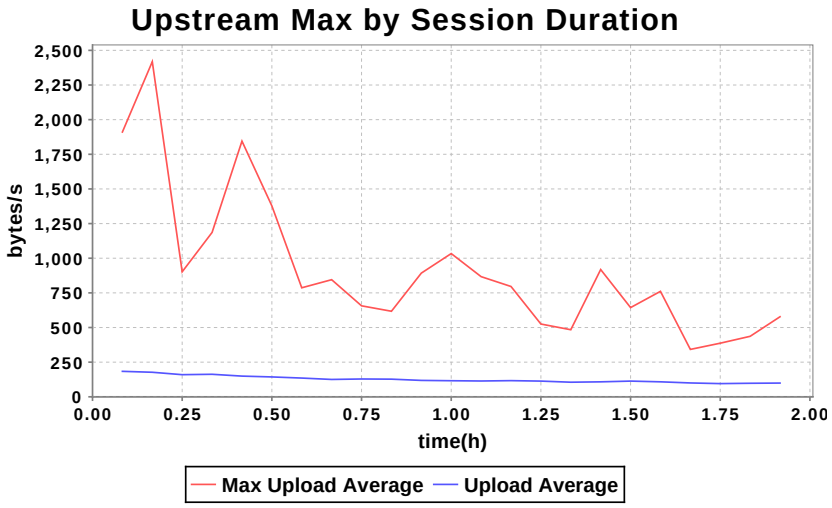


(b) Tráfego de *downstream*

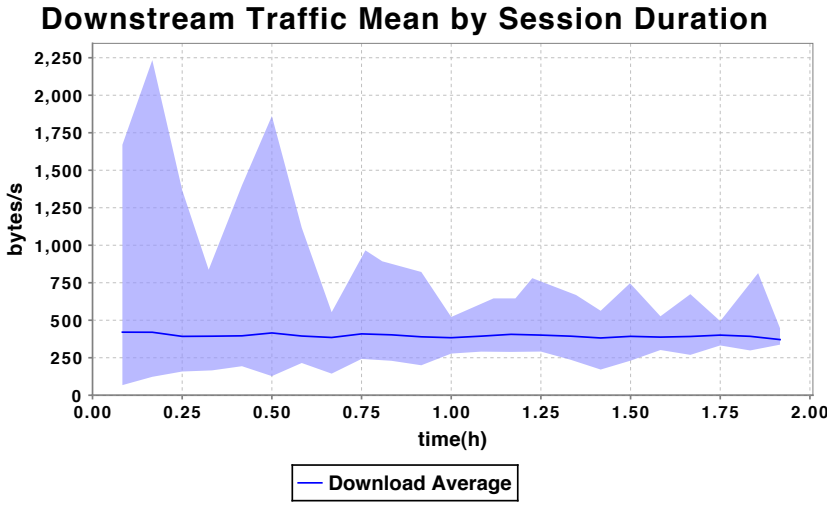
Figura 6.17: Resultados obtidos com delegação de trabalho para os filhos com $G = 8$

se reflicta de forma mais acentuada nos valores obtidos nas simulações.

Em relação aos valores observados nos gráficos referentes ao tráfego de *downstream* (figuras 6.15 (b), 6.16 (b), 6.17 (b) e 6.18 (b)), é possível verificar que não existem diferenças significativas em relação ao que já tinha sido observado nas simulações anteriores em que não era utilizada a delegação de trabalho para os filhos. Podemos então concluir, como era esperado, que este melhoramento, afecta apenas o tráfego de *upstream* dos super-nós e não tem qualquer influência no tráfego de *downstream*.



(a) Tráfego de *upstream*



(b) Tráfego de *downstream*

Figura 6.18: Resultados obtidos com delegação de trabalho para os filhos com $G = 10$

6.8 Outros melhoramentos

Para além dos critérios utilizados pelos super-nós para seleccionar os filhos a promover e da forma de entrada utilizada, existem outros melhoramentos que podem ser aplicados ao algoritmo de filiação com super-nós testado. Seria no entanto necessária mais informação de que não se dispõe para avaliar com algum rigor o interesse desses melhoramentos. São aqui pormenorizados alguns destes melhoramentos cujos resultados podem ser quantificados posteriormente.

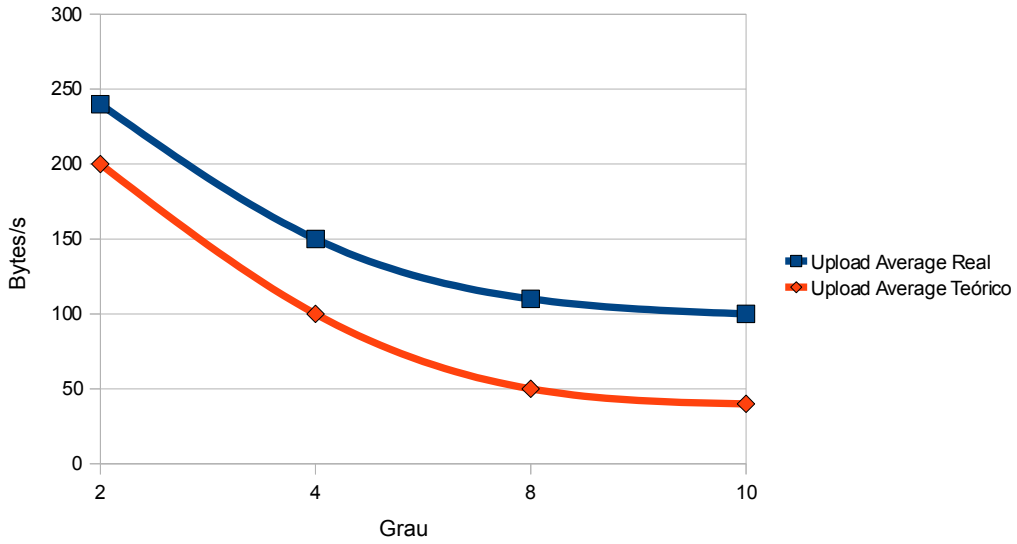


Figura 6.19: Comparação dos resultados obtidos face ao esperado. A curva referente à variável "Upload Average Real" corresponde aos valores nas simulações apresentadas. A curva da variável "Upload Average Teórico" toma os valores da expressão (6.3), $b_u = \frac{400}{G}$.

6.8.1 Outros critérios para promoção dos filhos

Além dos critérios já considerados e testados, podem ser equacionadas novas opções para fazer a selecção de filhos a promover

Um critério alternativo seria dividir os nós em classes, baseados em informação como a utilização de *NAT* por parte do nó, qual a capacidade estimada do nó, ou outros factores que possam ser determinantes e seleccionar o nó mais apto para ser promovido para super-nó.

A noção de classes pode ainda ser desenvolvida para um outro modelo mais perfeito em que cada nó utiliza um critério de ordenação (*ranking*) para decidir qual dos seus filhos deve promover a super-nó. A posição de um nó poderia ser determinada em função da classe do nó e através de conhecimento de sessões passadas do nó em questão. Nestes moldes, um nó precisaria de guardar de forma persistente vária informação sobre os nós, nomeadamente informação sobre duração de sessões passadas.

6.8.2 Adaptação ao ritmo de entrada

Além dos critérios para seleccionar um dos filhos a promover, um super-nó deve decidir quando deve promover um filho. Com o algoritmo de filiação testado neste capítulo, as promoções acontecem ao aceitar um novo filho, tendo em atenção um mecanismo de promoção antecipada segundo uma função $p(x)$.

A escolha do momento para promover um filho poderia ser feita não no momento em que é

aceite um novo filho, mas apenas quando um super-nó detectasse que a rede de super-nós estava estável e com um ritmo de entradas suficientemente baixo para admitir um novo elemento, pois como vimos, o tráfego de *upstream* depende linearmente da taxa de entradas na rede de super-nós.

Desta forma, seria possível estabelecer um limite para a taxa de entradas observadas e os super-nós seriam responsáveis por manter essa taxa de forma a que o custo de manutenção da filiação pudesse ser suportado.

Ao implementar este melhoramento, deve ser tomado em conta que esta aproximação pode levar a um efeito de *feedback* pois os super-nós ao detectarem ao mesmo tempo uma baixa taxa de entradas podem proceder à promoção de filhos na mesma altura levando a um aumento excessivo da taxa de entradas e posterior diminuição correspondente. Um dos métodos que podem ser utilizados para evitar o efeito de *feedback* é utilizar os *slice leaders* para controlarem o processo de promoção, avaliando a taxa de entradas observadas e difundido as ordens de promoção pelos nós da fatia pela qual cada *slice leader* é responsável.

6.8.3 Utilização de múltiplos pais

Este melhoramento consiste na implementação de um mecanismo que permita aos nós filhos a associação a mais do que um pai. Cada novo nó, ao entrar no sistema, poderia contactar mais do que um nó com o objectivo de ser filho desses nós, o que leva a que exista alguma redundância. Prevê-se que através deste melhoramento se possam obter melhoramentos significativos ao algoritmo de super-nós estudado neste capítulo.

Utilizado o algoritmo de super-nós apresentado, quando um super-nó saí da rede, seja devido a uma falha ou a uma saída normal, provoca a entrada abrupta no sistema dos nós filhos pelos quais o super-nó era responsável. Esta situação leva a que os super-nós tenham de encontrar novos nós semente imediatamente e recomeçar todo o seu processo de entrada na rede novamente, o que aumenta os custos em termos de capacidade de rede.

Este melhoramento poderá revelar-se especialmente interessante na diminuição dos custos referentes à distribuição de mensagens de difusão de conteúdos uma vez que os nós filhos poderiam escolher os seus pais com base na compatibilidade entre os seus filtros de conteúdos, o que permitiria distribuir pelos pais de um filho todo o trabalho de envio de mensagens de difusão de notificações, diminuindo assim os custos em termos de capacidade de rede de *upstream* com que cada super-nó tem que contribuir quando é responsável por um conjunto de nós filhos.

6.8.4 Comunicação entre filhos, compatibilidade de filtros e tempo antes de promoção

Utilizando uma aproximação em que os filhos dos super-nós possam comunicar entre si seria possível aligeirar a carga requerida ao super-nó, pois para o super-nó difundir uma mensagem

por todos os seus filhos bastaria enviar a mensagem a um dos seus filhos que ficaria encarregado de a enviar aos outros filhos.

Outro melhoramento a explorar é a possibilidade de implementar uma forma de os filhos ao entrarem na rede de super-nós serem reencaminhados para super-nós que possam ter uma maior compatibilidade de filtros com o filho de forma a facilitar a difusão de mensagens referentes à distribuição de conteúdos.

Finalmente, deve ser estudada uma forma de adaptar o tempo durante o qual um nó é filho em função do tempo necessário para transferir a tabela de encaminhamento do sistema.

7 . Conclusões e trabalho futuro

Neste capítulo são apresentadas as conclusões e o trabalho futuro a desenvolver em relação ao algoritmo de filiação do projecto LiveFeeds.

Este algoritmo tem a particularidade de ser um algoritmo de visibilidade completa, todos os nós conhecem todos os nós intervenientes no sistema bem como os seus filtros de subscrição de conteúdos, o que exige que exista uma replicação da informação de filiação. O algoritmo de filiação proposto para que esta replicação possa existir baseia-se na construção de árvores aleatórias. Como foi demonstrado, este tipo de aproximação permite que exista uma boa distribuição do custo em termos de capacidade de *upstream* e *downstream* por todos os nós do sistema. Através da replicação dos filtros de subscrições em que os nós estão interessados, juntamente com a informação de filiação, é possível simplificar o algoritmo de difusão de conteúdos. Ao dotar todos os nós com o conhecimento dos filtros de subscrições de todos os outros nós, ao receber uma mensagem de difusão de conteúdos, um nó sabe sempre quais os nós que estão interessados em receber a mensagem e a árvore de difusão pode ser construída de forma mais simples e eficiente e com a particularidade de que apenas os nós interessados em receber os conteúdos difundidos contribuem para a difusão da mensagem, tanto em termos de custos de capacidade de rede como em termos dos custos de computação associados à análise dos filtros de subscrição.

O estudo do algoritmo de filiação com visibilidade completa, recorrendo a métodos analíticos, permitiu que se verificasse a existência das propriedades de distribuição do custo de comunicação e computação. Esta análise proporcionou ainda um bom ponto de partida para determinar os custos em termos de capacidade de *upstream* e *downstream* necessários para utilizar este tipo de mecanismo de encaminhamento, tendo sido estabelecido que esse custo pode ser dado pela expressão $b_u = r \times m$, em que r representa a taxa de eventos por segundo observada na rede e m representa o tamanho da informação de filiação de um nó. Desta análise, foi possível ainda concluir que ao fim de um número suficiente de difusões de informação de filiação, o valor do tráfego de *upstream* é igual ao valor do tráfego de *downstream*, pois as vezes que um nó é escolhido para nó interior da árvore de difusão e tem de contribuir com mais tráfego são compensadas pelo número de vezes em que um nó é escolhido para nó folha e apenas tem de receber uma mensagem. Os resultados obtidos através desta forma de análise ao algoritmo foram ainda validados através de simulação antes de ser introduzido um modelo de *churn*.

Para que fosse possível fazer um estudo aprofundado ao algoritmo de filiação, foi desenvolvido um modelo de *churn* com recurso a trabalhos que estudaram o comportamento dos utilizadores em redes P2P cujos utilizadores tinham um comportamento semelhante ao que se espera dos possíveis utilizadores do LiveFeeds. Através deste modelo de *churn*, baseado numa distribuição de *Poisson* para modelar o tempo entre duas entradas consecutivas e numa distribuição

de *Weibull* para modelar o tempo de sessão dos nós, foi possível fazer um estudo detalhado ao algoritmo de filiação do LiveFeeds em ambiente de simulação sob condições próximas de uma implementação real.

Com os resultados destas simulações, foi possível concluir que os resultados anteriormente obtidos são válidos mesmo quando as condições de execução não são exactamente as mesmas do que as consideradas ao estudar o algoritmo de foram analítica, isto é, observa-se que existe uma distribuição dos custos de comunicação de *upstream* e *downstream*, ao fim de um número significativo de difusões da filiação, o tráfego de *upstream* é igual ao tráfego de *downstream*. Conclui-se ainda que a expressão anteriormente determinada, $b_u = r \times x$, é uma boa aproximação em relação ao valor de capacidade de rede de *upstream* e *downstream* com que cada nó deve contribuir para que o algoritmo de filiação considerado possa ser implementado num sistema com os parâmetros utilizados. Com base nestes resultados, foi possível tirar algumas conclusões relativamente à aplicabilidade do algoritmo de visibilidade completa. Concluiu-se que a utilização de árvores de difusão aleatórias tem como benefício que a capacidade de *upstream* necessária é proporcional à taxa de eventos por segundo observada no sistema e que a capacidade que cada nó deve suportar para que este mecanismo possa ser utilizado não depende do tamanho do sistema mas sim do comportamento dos utilizadores. Se este comportamento for estável o suficiente, o algoritmo de visibilidade completa com árvores aleatórias pode ser utilizado em sistemas com um grande número de nós. A aplicabilidade deste algoritmo depende assim da taxa de eventos por segundo, r, que se pretende suportar e do tipo de conectividade dos possíveis utilizadores do sistema. Fica assim demonstrado que para suportar utilizadores com um comportamento muito dinâmico será necessário utilizar diferentes soluções para os problemas considerados. Por outro lado, o algoritmo de filiação analisado pode ser utilizado num ambiente em que os utilizadores têm tempos de sessão muito elevados, o que leva a que a taxa de eventos seja suficientemente baixo para que o algoritmo possa ter custos aceitáveis, como é o caso de um sistema de *brokers* alojados em ISPs, instituições de ensino, etc, que servem os conteúdos obtidos através de *feeds RSS* aos utilizadores normais, que obtêm os conteúdos através do seu *broker* através do tradicional método *HTTP*.

Os custos para a utilização do algoritmo de visibilidade completa levam a que tivesse sido necessário encontrar outras soluções para o caso em que o comportamento dos utilizadores é muito dinâmico. Uma destas soluções foi investigada no decurso desta dissertação e consiste numa hierarquização do sistema através do uso super-nós. Foram estudados vários critérios de forma a afinar o algoritmo sugerido e foram feitas algumas recomendações relativas a estes critérios. Através da avaliação em ambiente de simulação, foi possível verificar que de entre os critérios estudados para promoção de filhos o critério mais adequado é a escolha aleatória de um nó filho, no entanto, os resultados obtidos mostram ainda que a utilização de um critério em que se escolhe o nó mais antigo pode trazer alguns ganhos ao sistema, caso se verifique que quanto maior for o tempo de sessão de um utilizador maior a probabilidade do mesmo permanecer

no sistema durante ainda mais tempo. Foi ainda estabelecido que o tipo de entrada em que é escolhido um nó semente aleatoriamente é um bom critério que não introduz custos de maior no sistema quando comparado com o caso óptimo em que é escolhido um super-nó semente com base em informação perfeita sobre os sistema. Apesar de a replicação da informação de filiação dos nós ser apenas parcial, pois os nós filhos não conhecem todos os super-nós, existe uma replicação total da informação de filiação dentro da rede de super-nós, dado que todos os super-nós conhecem todos os outros super-nós bem como os filtros subscritos por cada super-nó. Dentro desta rede de super-nós, os intervenientes continuam assim a usufruir das vantagens inerentes à utilização da replicação total da informação de filiação, como a distribuição de carga em termos de capacidade de rede e de esforço de computação e simplicidade conseguida aquando da difusão de conteúdos.

Depois de determinados os custos necessários para que o algoritmo de super-nós possa ser utilizado, foi sugerido um melhoramento que consiste na utilização dos filhos de um super-nó para substituir o super-nó quando este é escolhido para nó interior da árvore de difusão. Através do estudo pormenorizado dos ganhos obtidos através da implementação deste melhoramento, foi possível concluir que é possível fazer com que o custo em termos de capacidade de *upstream* diminua em função de G, o grau utilizado da árvore da difusão. Como foi demonstrado, quanto maior for G, menor será o tráfego de *upstream* com que cada super-nó tem de contribuir para que exista uma replicação total de filiação dentro da rede de super-nós.

Outro melhoramento que se prevê ter especial importância será a utilização de múltiplos super-nós pai para o mesmo filho. Através desta mudança ao algoritmo de super-nós inicialmente proposto, espera-se que se possa diminuir os custos associados à falha de super-nós e à difusão de conteúdos dos super-nós para os filhos.

Neste âmbito, foram ainda propostos outros melhoramentos para o algoritmo de super-nós que consistiam no uso de mecanismos que diminuem o custo necessário para utilizar este tipo de aproximação, como a adaptação dos super-nós ao ritmo de entrada observado, a delegação de trabalho para os nós filhos e a implementação de novos critérios de escolha de filhos a promover. Para trabalho futuro em relação ao estudo desta alternativa resta analisar os efeitos dos melhoramentos propostos.

Para além da implementação dos melhoramentos sugeridos para o algoritmo de super-nós, o trabalho futuro deve ainda incluir o estudo aprofundado dos resultados obtidos tendo em conta as falhas de nós e os efeitos que tais falhas possam introduzir no sistema. É ainda preciso estudar as questões relacionadas com a dimensão da informação de filiação que cada nó precisa de receber antes de entrar no sistema.

Complementarmente, pode ainda ser investigada uma outra alternativa ao algoritmo de encaminhamento com visibilidade completa que não inclua a noção de super-nós. Para tal pode ser equacionada uma versão do algoritmo em que os nós têm apenas uma visibilidade parcial da filiação e vão encaminhando as mensagens com base no conhecimento actual da filiação,

quando um nó recebe uma mensagem destinada a um nó que não é conhecido, reencaminha a mensagem para outro nó que possa ter uma visão da filiação que inclua o nó do destino, repetindo-se este processo até a mensagem ser entregue. As implicações, especificidades e melhoramentos deste tipo de aproximação devem ser alvo de estudo no futuro.

Ainda que depois de detalhadamente analisado o algoritmo de filiação com visibilidade completa, proposto como ponto de partida desta dissertação, se tenha concluído que tal aproximação pode ser utilizada apenas num cenário em que os utilizadores não tenham um comportamento demasiado dinâmico, foram equacionadas as vantagens e desvantagens da utilização de um mecanismo de visibilidade completa. Depois de determinados os limites de aplicabilidade do algoritmo de filiação com visibilidade completa, foi sugerida e estudada detalhadamente uma outra aproximação baseada na utilização de super-nós, que se espera resolver de forma eficiente os problemas colocados ao projecto LiveFeeds no âmbito do algoritmo de filiação.

A . Setting up Modelnet: A large-scale network emulator

A.1 Introduction

After the design and implementation phase, distributed systems need to be evaluated in order to improve them or validate the observed results. Network emulation is an essential tool for experimental testing and evaluation of distributed systems, algorithms and protocols. Network emulators subject traffic to the end-to-end bandwidth constraints, latency, and loss rate of a user-specified topology [28], this way an emulator allows researchers to subject their applications to controlled conditions enabling them to evaluate the behaviour and performance of the tested systems.

ModelNet is a scalable large-scale network emulation environment. Using ModelNet, unmodified applications run on edge nodes configured to route all their packets through a core cluster. This core is responsible for emulating the characteristics of a user-specified target topology, these core machines subject traffic to various constraints present in a real network and then re-route the traffic back to the edge nodes. A single PC can act as core machine, emulating properties and characteristics of a network with thousands of links. ModelNet is therefore a good tool for setting up an emulation testbed.

Available documentation for ModelNet is, however, sparse. The last official version of the ModelNet How To [1], documenting ModelNet 0.90, was released in 2003. Meanwhile ModelNet 0.99 and a Linux version for the core nodes were released. This lack of documentation was already addressed by [2] but it does not show how to setup ModelNet under a modern Linux distribution using an up to date kernel. Furthermore, [2] uses commercial virtualization tools[1] to set up a testbed using only one computer.

This paper provides an additional source of documentation for setting up a network emulator using ModelNet, supplying instructions on how to configure the core nodes and edge nodes to use the latest stable version of the Debian Linux distribution[2] using the Linux 2.6.18 kernel. Using a modern version of a Linux distribution as opposed to an old FreeBSD version, users setting up ModelNet can benefit from up to date software and better hardware support besides new features and improvements added to the linux kernel. The Debian Linux distribution was chosen as it is the distribution with official support from ModelNet developers, furthermore, this distribution has a large up to date software repository and a an extensive community providing an excellent support knowledgebase, also, the laboratories at Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, have Ubuntu Linux installed, which is based on Debian. Instructions on how to set up the network emulator under virtualization, using the free and

[1] VMWare - http://www.vmware.com/
[2] http://www.debian.org/

open source tool VirtualBox[3], are also provided allowing researchers to run ModelNet using only one computer for debugging or testing purposes over smaller networks.

The rest of this paper is organized as follows. In section A.2 the ModelNet architecture is presented. Instructions on how to setup ModelNet under Debian Linux 4.0r6 "Etch" are shown in section A.3. Steps on how to configure a small testbed environment using virtualization are provided in section A.4. In section A.5 we show how topologies can be deployed into ModelNet and prove that ModelNet is working as expected. The paper is concluded in section A.6.

A.2 ModelNet Architecture

ModelNet architecture is based on two groups of nodes, core and edge nodes, with each node being a computer in the cluster. Users run instances of the unmodified application being tested on edge nodes. Each instance of the application is contained in a virtual node with a unique IP address. When the application sends an IP packet through the corresponding virtual node interface, the packet is intercepted and routed to the core nodes. Core nodes, upon receiving these packets, emulate a user-specified topology, delaying or accelerating packets, according to the emulated network and then send them to the edge node containing the target virtual node. Therefore, the actual emulation of the specified topology is accomplished at the core nodes where network properties and characteristics such as bandwidth constraints, latency, loss rate, etc. are emulated.

A.3 Setting up ModelNet under Debian Linux

In this section instructions on how to set up ModelNet under Debian Linux are provided. Both core and edge nodes were installed over Debian GNU/Linux 4.0r6 "Etch", the stable branch of the Debian distribution at the time of writing. The Linux kernel 2.6.18 was used as it is the latest kernel at the software repositories for this Debian version.

Figure A.1 shows how the two machines are connected and the used hostnames and IP addresses. The private network 172.16.0.0/16 was used but other can be used if it does not overlap with network 10.0.0.0/8 which is used by ModelNet.

ModelNet 0.99 was released with a Linux version but it does not run using Linux kernel 2.6.18. A patch can be obtained at the ModelNet Wiki (`http://modelnet.ucsd.edu/patches/linux/sstv_7_08_linux.patch`) and applied to the Linux version of ModelNet using the following commands.

[3] http://www.virtualbox.org/

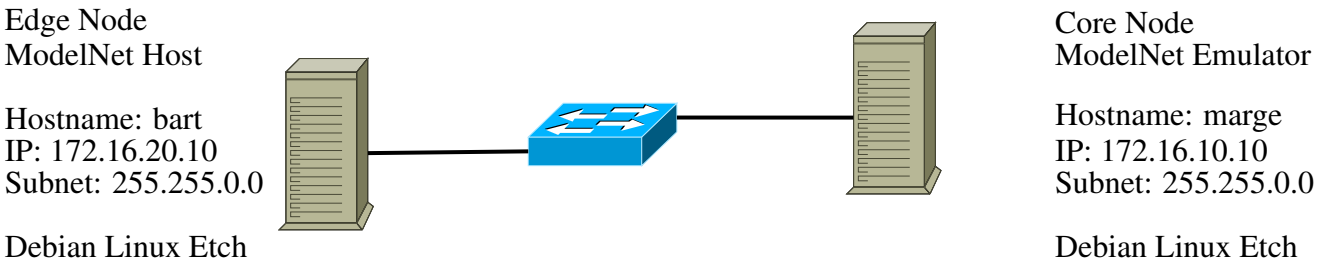


Figura A.1: Modelnet setup used

```
tar xvzf modelnet-linux-0.99.tar.gz
cd modelnet
patch -p0 < ../sstv_7_08_linux.patch
```

Listing A.1: Commands to patch modelnet to kernel 2.6.18

The following steps are different between core and edge nodes and depicted in the following sections.

A.3.1 Core nodes

After installing Debian in the core machines and configuring the corresponding network interfaces, the required software packages have to be installed. This can be done using the commands shown at listing A.2.

```
aptitude install kernel-package module-assistant build-essential \
libboost-graph-dev libxerces27-dev libssl-dev xinetd libxml-simple-perl
```

Listing A.2: Installing the required software packages

Most ModelNet scripts are written in Perl, and their dependencies can be installed using the following commands.

```
cpan install Graph
cpan install Heap
cpan install Xml::Simple
```

Listing A.3: Installing the required perl packages

[1] suggest that the core nodes machines kernel should be modified to use a clock rate set to 10000 HZ. Although the steps required to compile and install a modified Debian kernel are beyond the scope of this paper, they can easily be found at the WWW[4]. The kernel modification required by ModelNet can be done using the "menuconfig" mode to configure the kernel

[4]E.g. http://www.howtoforge.com/kernel_compilation_debian_etch

sources and modifying the option under “Processor type and features -> Timer Frequency -> 1000 HZ”.

With all dependencies satisfied and Modelnet patched to work with kernel 2.6.18 we can now proceed to install the ModelNet kernel module. The necessary steps are shown in listing A.4.

```
cd modelnet
./configure
cd linux_module
make
make install
cp linuxmodelnet.ko /usr/local/lib/linuxmodelnet.ko
cp modelload /usr/local/bin
cp modelstat /usr/local/bin
```

Listing A.4: Installing ModelNet at the core machines

In order to build topology files, the commands presented in listing A.5 must be executed.

```
cd modelnet
cp src/inet2xml /usr/local/bin
cp src/mkmodel /usr/local/bin
```

Listing A.5: Installing scripts to build topologies

After these steps the core machines are configured and topologies can be deployed into them.

A.3.2 Edge Nodes

In edge nodes, the ModelNet kernel module doesn't have to be installed, but a library, libipaddr, is used to interpose on certain Linux system calls in order to route all packets to core machines. This library is part of ModelNet and can be installed using the following commands.

```
cd modelnet
./configure
cd src
mv Makefile.orig Makefile
make
make install
```

Listing A.6: Installing libipaddr

Edge machines need to have the following scripts installed.

```
cd modelnet
cp src/deploy /usr/local/bin
cp src/deployhost /usr/local/bin
cp src/hostrun /usr/local/bin
cp src/vnrun /usr/local/bin
cp src/vnrunhost /usr/local/bin
cp src/libipaddr.so /usr/local/lib
```

Listing A.7: Installing scripts

Commands represented at listing A.5 can also be executed at edge machines.

Core and edge nodes are now configured and topologies can be deployed into them.

A.3.3 Installing gexec and authd

Although gexec and authd were successfully installed, using the instructions presented in [1], their correct operation could not be verified. Consequently, deployment and running of applications to virtual nodes becomes harder when using more than 2 or 3 machines to setup ModelNet, since deployment commands must be executed on each machine sequentially.

The impossibility of using gexec renders the following commands unusable.

deploy This command is used to access each machine in the ModelNet cluster and run the command `deployhost`. Thus, to overcome this difficulty, the command `deployhost` must be executed at each machine sequentially.

vnrun This command is used to run an application within one or more virtual nodes belonging to an emulation. Without the use of this command, bash syntax can be used to force an application to run within a virtual node, preloading the required libraries (libipaddr.so).

Instructions on how to do this are provided in section A.5. However, this requires one execution of this command for each virtual node, which can be cumbersome for network emulations with many virtual nodes.

In order to run these commands, scripts can be built that connect to each of the machines in use through ssh[5] and execute the required commands for each virtual node (library preloading) or host machine (`deployhost`), simulating the execution of `vnrun` and `deploy`. Although this solution is not as scalable as using `gexec`, it can be used until a better alternative can be found and is still scalable enough to be used on a laboratory with up to about twenty computers.

A.4 Setting up a testbed environment using virtualization with VirtualBox

Virtualization is a powerful tool and allows researchers to set up a small network emulation using only one computer. Multiple virtual nodes can be emulated using, for example, a laptop. One virtual machine can act as ModelNet emulator, a core node, and other machine can act as an edge node, this way, we obtain a simple and functional ModelNet setup that can be used for testing and debugging of smaller networks.

In this section we provide instructions for setting up a system similar to the one described in section A.3 and represented with figure A.1.

VirtualBox is used as it is a free and open source tool that efficiently provides virtualization facilities and runs over various host operating systems. We used Arch Linux[6] but the steps shown are identical for other linux distributions and can easily be adapted to other operating systems.

A.4.1 Configuring VirtualBox

Steps for creating the two used virtual machines are straightforward and for that reason they are not show here, for more info the VirtualBox manual [13] is available.

After creating the two virtual machines and installing Debian Linux in each of them, the virtual machines interfaces must be configured. The main interface of a virtual machine is configured using NAT, allowing the virtual machine network interface use the host operating system main network interface. The only problem with this approach is that the virtual machine network interface is by default configured to use the 10.0.0.0/8 network which ModelNet needs to use. This can be configured executing the command shown at listing A.8 for each virtual

[5] Secure Shell - http://www.openssh.com/
[6] Arch Linux - http://www.archlinux.org

machine. The name of the virtual machine, “ModelNet Host”, should be changed for each created virtual machine.

```
VBoxManage modifyvm "ModelNet Host" –natnet1 "192.168/16"
```

Listing A.8: Configuring the virtual machine network interface

After booting each of the virtual machines, they should be able to access the same network as the host operating system and have an IP address belonging to the 192.168.1.0/24 network.

The two virtual machines, however, cannot yet communicate between themselves, the configured network card only allows them to access the host operating system network interface. Another network interface has to be added to each virtual machine so they can join the same local network. This can be done using the command represented in listing A.9 for each VM.

```
VBoxManage modifyvm "Modelnet Host" –nic2 intnet
```

Listing A.9: Configuring the network interface of a VM for internal networking between VMs

After executing this command and rebooting the two VMs, they should now have two interfaces available and the IP address for each of them can be configured using the command `ifconfig`. In the ModelNet emulator VM we can use the following command.

```
ifconfig eth1 172.16.10.10/24
```

Listing A.10: Configuring internal networking of a VM

This command should be executed for each VM, the represented IP address must be different for each VM. To make these changes permanent, the lines shown in listing A.11 must be added to /etc/network/interfaces.

```
auto eth1
iface eth1 inet static
        address 172.16.10.10
        netmask 255.255.0.0
```

Listing A.11: Permanently configuring internal networking

To ease our work, the file /etc/hosts can be changed to include the VM’s internal network IP addresses, adding the lines represented in listing A.12.

```
172.16.10.10 bart
172.16.20.10 marge
```

Listing A.12: Added lines to /etc/hosts on both VM’s with hostnames marge and bart.

The output of `ifconfig` for each VM, after these steps, is shown in listings A.13 and A.14.

```
eth0      Link encap:Ethernet  HWaddr 08:00:27:5B:D1:2C
          Inet addr:192.168.20.15  Bcast:192.168.20.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:7 errors:0 dropped:0 overruns:0 frame:0
          TX packets:7 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1752 (1.7 KiB)  TX bytes:1042 (1.0 KiB)
          Interrupt:11 Base address:0xc020

eth1      Link encap:Ethernet  HWaddr 08:00:27:A7:68:8C
          Inet addr:172.16.10.10  Bcast:172.16.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:7 errors:0 dropped:0 overruns:0 frame:0
          TX packets:7 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:610 (610.0 b)  TX bytes:574 (574.0 b)
          Interrupt:10 Base address:0xc060

lo        Link encap:Local Loopback
          Inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:560 (560.0 b)  TX bytes:560 (560.0 b)
```

Listing A.13: ifconfig output at marge ModelNet emulator

The two configured virtual machines should now be able to communicate between themselves, as show in listing A.15.

ModelNet core and edge nodes can now be configured using the instructions presented in section A.3.

More host or emulator machines can be added to this set up using analogous steps.

A.5 Deploying topologies

As documented in [1], to run a network emulation with ModelNet, several files are needed.

graph.xml lists the nodes and links of the virtual network

route.xml contains route data for paths through the virtual network

machines.xml lists the machines that can be emulators or host virtual nodes

model.xml matches nodes and links to host machines and emulator machines

```
eth0      Link encap:Ethernet  HWaddr 08:00:27:EE:C1:03
          Inet addr:192.168.20.15  Bcast:192.168.20.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:7 errors:0 dropped:0 overruns:0 frame:0
          TX packets:7 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1748 (1.7 KiB)  TX bytes:1038 (1.0 KiB)
          Interrupt:11 Base address:0xc020

eth1      Link encap:Ethernet  HWaddr 08:00:27:60:52:1B
          Inet addr:172.16.20.10  Bcast:172.16.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
          Interrupt:10 Base address:0xc060

lo        Link encap:Local Loopback
          Inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:560 (560.0 b)  TX bytes:560 (560.0 b)
```

Listing A.14: ifconfig output at bart ModelNet host

```
bart:~# ping marge
PING marge (172.16.10.10) 56(84) bytes of data.
64 bytes from marge (172.16.10.10): icmp_seq=1 ttl=64 time=19.3 ms
64 bytes from marge (172.16.10.10): icmp_seq=2 ttl=64 time=6.39 ms
64 bytes from marge (172.16.10.10): icmp_seq=3 ttl=64 time=5.23 ms
64 bytes from marge (172.16.10.10): icmp_seq=4 ttl=64 time=17.6 ms
64 bytes from marge (172.16.10.10): icmp_seq=5 ttl=64 time=6.34 ms
--- marge ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4015ms
rtt min/avg/max/mdev = 5.231/10.983/19.317/6.150 ms

marge:~# ping bart
PING bart (172.16.20.10) 56(84) bytes of data.
64 bytes from bart (172.16.20.10): icmp_seq=1 ttl=64 time=2.55 ms
64 bytes from bart (172.16.20.10): icmp_seq=2 ttl=64 time=1.46 ms
64 bytes from bart (172.16.20.10): icmp_seq=3 ttl=64 time=8.81 ms
--- bart ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2006ms
rtt min/avg/max/mdev = 1.463/4.276/8.813/3.239 ms
```

Listing A.15: Execution of the ping command at each VM

To emulate a network, ModelNet requires route.xml and model.xml which are automatically generated from graph.xml and machines.xml using Perl scripts distributed with ModelNet.

The graph.xml file can be generated from an Inet[7] output converted to XML using the tool included with ModelNet, `inet2xml`. Other topology generators can be used with ModelNet, provided their outputs can be converted to a XML format that ModelNet can understand. machines.xml should be manually created by the user.

A.5.1 A simple topology

In this section, we start by emulating a small network in order to guarantee that our setup is working properly.

We use a graph.xml file based on [2, 1], shown in listing A.16, and machines.xml shown in A.17 representing our ModelNet setup configured using virtual machines.

```
<?xml version="1.0" encoding="UTF-8"?>
<topology>
  <vertices>
    <vertex int_idx="0" role="gateway" />
    <vertex int_idx="1" role="virtnode" int_vn="0" />
    <vertex int_idx="2" role="virtnode" int_vn="1" />
    <vertex int_idx="3" role="virtnode" int_vn="2" />
    <vertex int_idx="4" role="virtnode" int_vn="3" />
  </vertices>
  <edges>
    <edge int_dst="1" int_src="2" int_idx="0" specs="client-stub" int_delayms="1" />
    <edge int_dst="2" int_src="1" int_idx="1" specs="client-stub" dbl_kbps="768" />
    <edge int_dst="1" int_src="3" int_idx="2" specs="client-stub" dbl_kbps="768" />
    <edge int_dst="3" int_src="1" int_idx="3" specs="client-stub" />
    <edge int_dst="0" int_src="4" int_idx="4" specs="client-stub" />
    <edge int_dst="4" int_src="0" int_idx="5" specs="client-stub" />
    <edge int_dst="1" dbl_len="1" int_src="0" int_idx="0" specs="stub-stub" />
    <edge int_dst="0" dbl_len="1" int_src="1" int_idx="1" specs="stub-stub" />
  </edges>
  <specs >
    <client-stub dbl_plr="0" dbl_kbps="64" int_delayms="100" int_qlen="10" />
    <stub-stub dbl_plr="0" dbl_kbps="1000" int_delayms="20" int_qlen="10" />
  </specs>
</topology>
```

Listing A.16: graph.xml file used

```
<?xml version="1.0" encoding="UTF-8"?>
<hardware>
        <emul hostname="marge"/>
        <host hostname="bart"/>
</hardware>
```

Listing A.17: machines.xml file used based on our configured VM's

[7] Inet - http://topology.eecs.umich.edu/Inet/

The commands shown in listing A.18 show how to generate route.xml and model.xml from graph.xml and machines.xml.

```
allpairs graph.xml > route.xml
mkmodel graph.xml machines.xml > model.xml
```

Listing A.18: Commands to generate route.xml and model.xml

The files required by ModelNet were generated and the topology can now be deployed. In order to do that the command shown in A.19 must be executed for each core and edge machine. This command creates the necessary number of virtual Ethernet adapters and modifies the routing table of all machines involved [2].

```
deployhost model.xml route.xml
```

Listing A.19: Execution of the deploy command

The ModelNet emulator is now running an emulated network with a user specified topology. This can be shown by running `iperf` and confirming if the observed throughput measurements are consistent with the graph.xml file used.

All applications running over the network emulated by ModelNet have to preload a library that interposes on certain system calls to route IP packets to the emulator core nodes. This library preloading can be done using the following syntax.

```
bart:~# VN=0 SRCIP=10.0.0.1 <command to execute>
```

Listing A.20: Execution of a command within a virtual node in this case virtual node 0 with IP address 10.0.0.1

To assess that the emulated network is consistent with the network idealized with the used graph.xml file, the bandwidth between virtual nodes 0 (IP address 10.0.0.1) and 2 (IP address 10.0.0.3) was tested using iperf.

The `iperf` server can be started at VN[8] 3 with the following commands, causing `iperf` to be executed in background.

```
bart:~# export LD_PRELOAD=/usr/local/lib/libipaddr.so
bart:~# VN=0 SRCIP=10.0.0.1 iperf -s &
```

Listing A.21: Starting `iperf` server at VN 0

The `iperf` client can be started with the following command, obtaining the output shown.

[8] Virtual Node

```
bart:~# VN=2 SRCIP=10.0.0.3 iperf -c 10.0.0.1
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 16.0 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.3 port 4088 connected with 10.0.0.1 port 5001
[  3]  0.0-25.4 sec  2.27 MBytes    749 Kbits/sec
```

Listing A.22: Starting `iperf` client at virtual node 2

The value 749 Kbits/sec is coherent with the graph.xml file used, showing that the emulated network is working as intended.

For the same purposes, the symmetric test can also be done.

```
export LD_PRELOAD=/usr/local/lib/libipaddr.so
bart:~# VN=0 SRCIP=10.0.0.1 iperf -c 10.0.0.3
------------------------------------------------------------
Client connecting to 10.0.0.3, TCP port 5001
TCP window size: 16.0 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.1 port 2864 connected with 10.0.0.3 port 5001
[  3]  0.0-10.4 sec  80.0 KBytes  63.2 Kbits/sec
```

Listing A.23: Starting an `iperf` client at VN 0

Again, the results show that the network emulation is working properly with ModelNet treating symmetric links differently. The considered link uses 64 kbps from VN 0 to VN 2 and 768 kbps from VN 2 to 0, which is consistent with `iperf` outputs shown in listings A.21 and A.23.

A.5.2 Using Inet to generate a topology

ModelNet directly supports topologies generated with Inet. Using a script distributed with ModelNet, `inet2xml`, Inet outputs can be converted to a graph.xml file that in turn can be used to generate the model and route files.

A graph.xml file can be generated from an inet output using the following commands [2].

```
bart:~# inet -n 4000 | Inet2xml -p 100 among 1 stubs client-stub \
64 100 0 > graph.xml
```

Listing A.24: Generating a topology using Inet to deploy to ModelNet

This generates an Inet topology with 4000 nodes, inet2xml parses the generated topology using 100 of those nodes as virtual nodes spreaded across 1 stub. This command also defines a client-stub as a 64 kbps link with 100 ms delay.

For reference, the `inet2xml` script syntax is shown in listing A.25 [1].

```
usage: inet2xml [-l] [-q qcnt] -p <vncnt> among <stubcnt> stubs
            [<link type> <kbps bw> <ms delay> <drop fraction>]+ |
            [min-<link type> <kbps bw> <ms delay> <drop fraction>
             max-<link type> <kbps bw> <ms delay> <drop fraction>]
    -l    Set delay based on link lengths derived from inet node location
    -q qcnt
          Queue length for on all links. Default is  10
    -p <vncnt> among <stubcnt> stubs
          Create <vncnt> virtual nodes spread among <stubcnt> stubs
    <link type>
          Types can be: client-stub stub-stub stub-transit transit-transit
    <kbps bw>
          integer kilobits per second
    <ms delay>
          integer milliseconds of link latency
    <drop fraction>
          real value from fraction of packets dropped
```

Listing A.25: `inet2xml` syntax

Using the machines.xml file previously used, with the graph.xml file generated and following the steps depicted in section A.5.1, this topology generated by Inet can be deployed and analogous tests can be performed to evaluate the emulation.

A.6 Conclusion

We have shown that it is possible to install ModelNet using the latest stable version of the Debian Linux distribution, Debian 4.0r6 “Etch” using Linux kernel 2.6.18, on both core and edge nodes. We also provided instructions on how to accomplish such a setup, enabling researchers interested in using ModelNet to use a vast up to date software repository, up to date documentation and better hardware support.

Steps on how to setup a network emulation testbed with only one computer using virtual machines are also presented.

The successful installation of ModelNet was proved using `iperf` with the deployment of a simple topology and we have shown how an inet generated topology can be deployed into ModelNet.

Bibliografia

[1] David Becker and Ken Yocum. *Modelnet Howto*, 2003. A.1, A.3.1, A.3.3, A.5, A.5.1, A.5.2

[2] Patrik Bless. A peer-to-peer research framework via emulation in a cluster, 2006. A.1, A.5.1, A.5.1, A.5.2

[3] S. Chhabra, E. Damiani, S.D.C. di Vimercati, S. Paraboschi, and P. Samarati. A protocol for reputation management in super-peer networks. In *Proceedings of the 15th International Workshop on Database and Expert Systems Applications*, pages 979–983. 6.1

[4] Rui Pedro da Silva Lopes and J. Legatheaux Martins. Sistemas de localização e de medida de distância na internet – contribuição para a avaliação da sua integração com algoritmos p2p. Master's thesis, Universidade Nova de Lisboa, 2009. 3.3

[5] A.K. Datta, M. Gradinariu, M. Raynal, and G. Simon. Anonymous publish/subscribe in p2p networks. In *Proceedings of IPDPS*, pages 22–26, 2003. 6.1

[6] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003. 1.2, 3.1

[7] P. Garbacki, D.H.J. Epema, and M. van Steen. Optimizing peer relationships in a super-peer network. In *Distributed Computing Systems, 2007. ICDCS'07. 27th International Conference on*, pages 31–31, 2007. 6.1

[8] P. Brighten Godfrey, Scott Shenker, and Ion Stoica. Minimizing churn in distributed systems. *SIGCOMM Comput. Commun. Rev.*, 36(4):147–158, 2006. 3.3

[9] S. Guha, N. Daswani, and R. Jain. An Experimental Study of the Skype Peer-to-Peer VoIP System. In *Proceedings of The 5th International Workshop on Peer-to-Peer Systems (IPTPS'06)*, pages 1–6, 2006. 3.3, 5.1, 5.1, 5.1, 6.1

[10] A. Gupta, B. Liskov, and R. Rodrigues. Efficient routing for peer-to-peer overlays. *Proc. First Symposium on Networked Systems Design and Implementation*, 2004. 1.2, 2, 3.2, 3.3, 5.4

[11] Abhishek Gupta, Ozgur D. Sahin, Divyakant Agrawal, and Amr El Abbadi. Meghdoot: content-based publish/subscribe over P2P networks. In *Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, pages 254–273, New York, NY, USA, 2004. Springer-Verlag New York, Inc. 3.1

[12] Mike Hibbler, Robert Ricci, Leigh Stoller, Jonathon Duerig, Shashi Guruprasad, Tim Stack, Kirk Webb, and Jay Lepreau. Large-scale Virtualization in the Emulab Network Testbed. In *Proc. of the 2008 USENIX Annual Technical Conference*, June 2008. 3.4, 3.4

[13] Sun Microsystems Inc. *Sun xVM VirtualBox User Manual*, 2008. A.4.1

[14] R. Dean Malmgren, Daniel B. Stouffer, Adilson E. Motter, and Luís A. N. Amaral. A Poissonian explanation for heavy tails in e-mail communication. *Proceedings of the National Academy of Sciences*, 105(47):18153–18158, 2008. 3.3, 5.1, 5.1

[15] J. Legatheaux Martins and Sérgio Duarte. Routing Algorithms for Content-based Publish/Subscribe Systems. *Accepted for publication in IEEE Communications Surveys & Tutorials*, 2009. 1.1, 1.2, 2, 3.1

[16] Simão Mata, J. Legatheaux Martins, Sérgio Duarte, and Margarida Mamede. Análise do custo e da viabilidade de um sistema P2P com visibilidade completa. In *INForum Simpósio de Informática 2009*, pages 333–344. Faculdade de Ciências da Universidade de Lisboa, September 2009. 1.2

[17] Sun Microsystems. Java website. `http://java.sun.com/`, July 2009. [Online; accessed 19-July-2009]. 1.3

[18] S.H. Min, J. Holliday, and D.S. Cho. Optimal super-peer selection for large-scale p2p system. In *Hybrid Information Technology, 2006. ICHIT'06. International Conference on*, volume 2, 2006. 6.1

[19] AT Mizrak, Y. Cheng, V. Kumar, and S. Savage. Structured superpeers: Leveraging heterogeneity to provide constant-time lookup. In *Internet Applications. WIAPP 2003. Proceedings. The Third IEEE Workshop on*, pages 104–111, 2003. 6.1

[20] Venugopalan Ramasubramanian and Emin Gün Sirer. Beehive: O(1)lookup performance for power-law query distributions in peer-to-peer overlays. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pages 8–8, Berkeley, CA, USA, 2004. USENIX Association. 3.2

[21] S. C. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. *Handling Churn in a DHT*. Computer Science Division, University of California, 2003. 1.2, 3.3

[22] Rodrigo Rodrigues and Charles Blake. When Multi-Hop Peer-to-Peer Lookup Matters. In *In Proc. of IPTPS*, pages 112–122, 2004. 1.2, 3.2

[23] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. *Lecture Notes In Computer Science*, 2218:329–350, 2001. 3.2

[24] S. Singh, S. Ramabhadran, F. Baboescu, and A.C. Snoeren. The case for service provider deployment of super-peers in peer-to-peer networks. In *Proc. of International Workshop of Economics of Peer-To-Peer Systems*, 2003. 6.1

[25] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160, New York, NY, USA, 2001. ACM. 3.3

[26] Daniel Stutzbach and Reza Rejaie. Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 189–202, New York, NY, USA, 2006. ACM. 3.3

[27] Amin Vahdat, Ken Yocum, Kevin Walsh, Priya Mahadevan, Dejan Kostić, Jeff Chase, and David Becker. Scalability and accuracy in a large-scale network emulator. *SIGOPS Oper. Syst. Rev.*, 36(SI):271–284, 2002. 3.3, 3.4, 3.4

[28] Amin Vahdat, Ken Yocum, Kevin Walsh, Priya Mahadevan, Dejan Kostić, Jeff Chase, and David Becker. Scalability and accuracy in a large-scale network emulator. *SIGOPS Oper. Syst. Rev.*, 36(SI):271–284, 2002. A.1

[29] Z. Xiao, L. Guo, and J. Tracey. Understanding instant messaging traffic characteristics. In *Proceedings of the 27th International Conference on Distributed Computing Systems*, page 51. IEEE Computer Society Washington, DC, USA, 2007. 3.3, 5.1, 5.1, 5.1

[30] B. Yang and H. Garcia-Molina. Designing a super-peer network. In *Proceedings of the International Conference on Data Engineering*, pages 49–62. IEEE Computer Society Press; 1998, 2003. 6.1, 6.1