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

Dissertação de Mestrado em Engenharia Informática
1º Semestre, 2008/2009

Sistemas de localização e de medida de distância na Internet – Contribuição para a avaliação da sua integração com algoritmos P2P

Nº 26220 – Rui Pedro da Silva Lopes

Orientador
Prof. Doutor José Augusto Legatheaux Martins

20 de Fevereiro de 2009

Nº do aluno: 26220
Nome: Rui Pedro da Silva Lopes

Título da dissertação:
Sistemas de localização e de medida de distância na Internet –
Contribuição para a avaliação da sua integração com algoritmos P2P
Palavras-Chave:

- Sistemas de coordenadas na rede
- Localização na Internet
- Distância na rede
- "Geolocalização" IP
- Desempenho de algoritmos P2P

Keywords:

- Network coordinate systems
- Internet Location
- Network distance
- IP Geolocation
- P2P algorithms performance

Agradecimentos

Gostaria de dar uma palavra de agradecimento a todos os que de alguma forma contribuíram positivamente para o desenrolar desta dissertação.

Em primeiro lugar quero agradecer à minha família, nomeadamente aos meus pais, que sempre me apoiaram em tudo o que foi necessário, não a nível do projecto propriamente dito, mas sim a nível pessoal, o que é sempre muito importante.

À minha ex-namorada e amiga Raquel, agradeço o facto ter compreendido quando, por demasiadas vezes, não pude estar com ela, por estar dedicado a esta dissertação.

Ao professor Sérgio Marco Duarte, autor do simulador e do algoritmo LiveFeeds, que foram utilizados na parte prática desta dissertação, agradeço a disponibilidade em termos de tempo e toda a ajuda que foi prestando ao longo da mesma e que foi preciosa.

Por último, um especial agradecimento ao meu orientador, o professor José Augusto Legatheaux Martins, pela enorme ajuda, pela dedicação e interesse, pelo acompanhamento incansável e permanente e pela compreensão demonstrada pela minha situação de trabalhador-estudante.

A todos, muito obrigado.

Resumo

Com o crescimento da utilização de sistemas distribuídos e nomeadamente dos sistemas de difusão de conteúdos que têm por base sistemas "peer-to-peer" torna-se importante valorizar as questões relacionadas com a forma como este tipo de sistemas e algoritmos podem ser optimizados de modo a melhorar a experiência dos utilizadores, se possível minimizando a carga gerada sobre as redes de uma forma geral. De entre os aspectos nos quais podem ser desenvolvidas melhorias encontra-se a forma como são escolhidos parceiros de comunicação. Nem sempre a escolha de parceiros de comunicação privilegia aqueles que estão mais próximos em termos de latência de comunicação. Pretende-se estudar se existe uma mais valia ao empregar métodos que permitam conhecer em cada momento quais os nós mais próximos em termos de latência através de uma função de distância virtual e ainda, de entre os diferentes métodos disponíveis para esse fim quais os que poderão fornecer melhores resultados a um custo aceitável. Assim, foram estudados diferentes tipos métodos que podem ser usados neste contexto e, de entre esses foram escolhidos métodos considerados adaptáveis ao sistema de difusão de conteúdos em estudo, sendo integrados nesse mesmo sistema, neste caso o sistema LiveFeeds. Estes métodos fornecem uma função de distância virtual baseada em latência entre os nós participantes, sendo avaliado o seu desempenho e implicações no sistema referido, de modo a poder extrair conclusões que possam contribuir para um melhor entendimento desta problemática.

Abstract

With the increasing demand for distributed systems and particularly peer-to-peer based content broadcasting systems, it becomes imperative to value questions related to the way in which these kinds of systems and algorithms can be optimized in order to improve user's experience and possibly reducing overhead on networks in a general way. Between the various aspects that can undergo any improvements we can find the way in which the communication peers are chosen. The choice of communication peers does not always privilege those that are in close proximity in terms of communication latency. The main objective is to study if there is any kind of benefit in applying methods that allow knowing at any instant which of the network nodes are closest in terms of communication latency through a virtual distance function, and from all of the different available methods, which can provide the better results on an affordable cost. Thus, there were studied different kinds of methods in this context and from those, there were chosen some that were considered to be adaptable to the content broadcasting system in study, being incorporated in that same system, in the case, the LiveFeeds system. These methods can provide a virtual distance function based on the communication latency between the participating network nodes, being evaluated the performance and implications of these methods in the referred system, in a way in which is possible to extract any conclusions that can contribute to a better understanding of this issue.

Índice

Índice de tabelas

Índice de figuras

1. Introdução

Nos últimos anos tem-se vindo a assistir a um crescimento da utilização de redes de disseminação de conteúdos em larga escala baseadas na *web*, nomeadamente sistemas que permitem obter actualizações relativas ao conteúdo de sítios, *blogues*, ou outras fontes de informação baseadas na *web*.

Muitos dos actuais sistemas de disseminação de conteúdos baseiam-se em algoritmos e redes "peer-to-peer" (P2P). Algumas das propostas não têm em conta a latência da comunicação ou a localização, na escolha de parceiros. Quando tal é o caso, o modelo usado por esses sistemas é o de uma rede homogénea em que todas as ligações teriam os mesmos custos. Como isso não corresponde à realidade, estes tendem a degradar a experiência dos utilizadores, ao sofrerem de atrasos e velocidades de transferência mais reduzidas do que seria possível.

Mas não é só ao nível dos utilizadores que efeitos menos desejáveis se fazem sentir, a nível global, a rede é utilizada de forma não óptima, o que provoca atrasos e carga nas ligações da periferia e do núcleo da rede. Por exemplo, por vezes acontece que determinado conteúdo é transferido de um nó muito "distante", mas existir uma cópia do mesmo disponível num nó mais "próximo", tendo neste caso atravessado um canal de longa distância sem que na realidade houvesse necessidade que tal acontecesse. Também se verifica frequentemente que o mesmo conteúdo atravessa várias vezes os mesmos canais de longa distância, porque os utilizadores interessados no mesmo não se conhecem uns aos outros, ou mesmo que se conheçam, desconhecem que utilizadores interessados nesse mesmo conteúdo se encontram "próximos" [1].

Assim sendo, tanto os utilizadores como os provedores de acesso à Internet têm interesse em que a sua experiência melhore e o tráfego desnecessário seja reduzido, respectivamente, aplicando algoritmos que evitem que os conteúdos tenham de percorrer distâncias maiores quando tal não é estritamente necessário.

Para tal, os algoritmos de escolha de parceiros deveriam tomar em consideração a distância, em termos de latência, a que os diferentes nós se situam. Isso pode ser feito medindo essa latência através de pacotes de teste ("probing packets" ou "ping packets"). No entanto, se o número de nós for muito elevado, por exemplo, centenas, milhares ou centenas de milhares, a carga que tal geraria, não seria suportável, tornando o seu custo demasiado grande face ao benefício de daí se obteria, não sendo portanto realista. Em alternativa seria desejável dispor a priori dessa informação, por exemplo, na forma de acesso a uma matriz de custos de comunicação entre quaisquer dois nós do sistema.

Têm sido investigados métodos que tornam possível obter funções de distância na rede, que normalmente têm por base as latências na rede. Estas funções de distância podem ser obtidas através de medições, sistemas de coordenadas virtuais ou sintéticas e até mesmo pela correspondência com a topologia terrestre, através do cálculo de uma estimativa da localização física dos nós.

Desta forma, é fundamental estudar se esses sistemas de localização ou de medida de distâncias na rede (latência) permitem melhorar os métodos de escolha de parceiros e se o podem fazer de uma forma rentável, ou seja, se a aplicação de métodos que permitam de alguma forma calcular uma função distância, não têm um custo demasiado elevado, quando comparado com o seu benefício.

Partindo da hipótese, já demonstrada em alguns casos, que este tipo de sistemas podem ser benéficos, pelo menos em alguns casos, torna-se imperativo estudar de que modos podem ser usados esses sistemas de localização ou medida de distâncias na rede, para melhorar o desempenho e eficiência de algoritmos distribuídos, nomeadamente no contexto de sistemas distribuição de conteúdos e em redes *peer-to-peer*. E destes, quais os que fornecem os melhores resultados segundo uma lógica de custo/benefício.

O objectivo deste trabalho é testar esta hipótese no quadro de um sistema e um algoritmo de difusão de conteúdos particular, em desenvolvimento no projecto de investigação LiveFeeds, o qual é descrito no capítulo 3.

Para testar esta hipótese, torna-se necessário efectuar experimentação prática sobre algoritmos ou sistemas que forneçam funções ou matrizes de distâncias na rede, integrando esses mesmos sistemas no algoritmo de difusão em estudo de modo a poder obter resultados que possam contribuir para um melhor entendimento desta problemática e até mesmo

contribuindo para a evolução futura de sistemas que tirem partido de sistemas de localização na rede.

De modo a facilitar a sua leitura, apresenta-se em seguida estrutura e organização dos restantes capítulos presentes neste documento.

No capítulo dois são descritos sistemas de localização na rede, começando pelos sistemas de localização geográfica IP, passando em seguida aos sistemas de localização virtual.

No capítulo três, são descritos o sistema LiveFeeds o seu algoritmo de difusão, assim como os seus possíveis melhoramentos através de utilização de técnicas de coordenadas virtuais, sendo ainda discutidos alguns dos algoritmos estudados no contexto do sistema LiveFeeds e como estes podem contribuir para a sua melhoria, e a que níveis.

No capítulo quatro são apresentadas ferramentas de teste e analisada a relevância da sua utilização neste estudo, nomeadamente técnicas de experimentação na rede real, emulação e simulação e de sistemas distribuídos e redes. É também descrito o simulador usado, o seu modelo de funcionamento e as suas características específicas com maior impacto sobre os objectivos do estudo apresentado nesta dissertação. É ainda descrita a integração do algoritmo de difusão do sistema LiveFeeds, bem como a integração dos diversos algoritmos de coordenadas virtuais em estudo no contexto do simulador.

No capítulo cinco são apresentados os critérios de comparação e planificação de testes considerados durante o estudo.

No capítulo seis são apresentados os resultados obtidos através dos testes descritos anteriormente e sua análise.

No capítulo sete são apresentadas as conclusões e considerações sobre possíveis evoluções futuras.

Finalmente, no capítulo oito são apresentadas as referências bibliográficas consultadas durante a elaboração deste documento.

2. Sistemas de localização na rede

O problema de estimar a localização e assim poder estimar a distância para um dado nó na Internet pode ser visto por duas perspectivas distintas, uma que usa uma perspectiva de localização geográfica e outra em que a localização é de acordo com a topologia abstracta de rede, sendo neste caso apenas virtual e cuja relação com a localização física é deveras pobre.

A localização geográfica ou física de um nó da Internet, sabendo apenas o seu endereço IP é o alvo da secção seguinte onde é feita uma análise de várias abordagens conhecidas para este problema.

Os sistemas de coordenadas virtuais propriamente ditos e os sistemas de localização relativa na rede possuem também algumas abordagens conhecidas, sendo as consideradas mais relevantes, tratadas na secção 2.2.

2.1 Geolocalização IP

Estimar a localização física de um nó da Internet conhecendo apenas o seu endereço IP, também referida como "Internet Geolocation" ou "Geolocalização[1] IP", é um problema que tem vindo a ser estudado há alguns anos. Dada a sua natureza e dificuldade tem causado as mais diversas abordagens, nem sempre bem sucedidas. A grande dificuldade na resolução deste problema, com algum grau de exactidão, prende-se com o facto de não existir qualquer informação geográfica associada directamente ao endereço IP, sendo que as abordagens existentes apenas se podem basear nas latências de rede, em informação conhecida relativa ao domínio associado ao endereço IP, como os sistemas Whois e também informação obtida através de Reverse DNS Lookup [2]. Destas informações básicas partem as principais e mais estudadas abordagens para este problema. As quais são descritas nesta secção.

[1] Assume-se um neologismo pela junção do prefixo "geo" de geográfico à palavra localização, cujo significado é equivalente à expressão "localização geográfica".

Nas subsecções seguintes são introduzidas as principais abordagens conhecidas para este problema, subdivididas em conjuntos lógicos. Nomeadamente, os sistemas que se baseiam em bases de dados, em captura de informação, em medições de latência, um sistema que usa várias abordagens em simultâneo, concluindo com uma referência a sistemas comerciais ou proprietários.

2.1.1 Sistemas baseados em bases de dados públicas

Este tipo de sistema de localização não requer qualquer tipo de infra-estrutura especial, ao contrário dos sistemas baseados em pontos de referência ou "landmarks"[2] que efectuam medidas de latência, analisados em 2.1.3. Estes sistemas baseiam-se em bases de dados que são constantemente actualizadas e que recorrem a informação disponível em sistemas de consulta Whois [3], ou que usam a informação existente nos registos DNS LOC [4]. Os sistemas que se baseiam em bases de dados podem induzir em erros, na medida em que podem conter informação desactualizada ou em falta e podem também ser limitados à cobertura de zonas geográficas específicas, sendo neste caso incompletos.

A informação sobre um endereço IP pode ser facilmente obtida a partir de consulta a bases de dados Whois públicas. Estas bases de dados fazem a correspondência entre os endereços e IP ou espaço de endereçamento e entidades como os ISP, sistemas autónomos, organizações ou empresas. Pode então ser feita uma correspondência entre a informação de contacto dessas mesmas entidades (Moradas, números de telefone…) e o endereço IP, gamas de endereços, ou o(s) nome(s) associados a domínios.

O sistema DNS possui um tipo de registo, o DNS LOC, que pode ser usado voluntariamente para publicar informação rigorosa (coordenadas geográficas) sobre a localização de um determinado nó. Este tipo de registo é, no entanto, muitíssimo raro e pode conter informação errada, na medida que não possui qualquer tipo de validação.

A informação que consta geralmente das referidas bases de dados Whois é muito genérica, referindo-se normalmente a grandes organizações e que corresponde apenas a alguns locais reais como as sedes dessas mesmas organizações. Para além disto, essa informação

[2] "landmarks" pode ser traduzido do inglês por marcos territoriais ou pontos de referência. No que se segue será usado o termo original em inglês por facilidade de relação com os artigos originalmente publicados em inglês e que se referem desta forma ao termo.

pode estar desactualizada em alguns casos e até mesmo incompleta ou incorrecta, não existindo qualquer tipo de validação para estes dados.

2.1.2 Sistemas baseados em captura de informação

Este tipo de sistema pode ser subdividido em três categorias distintas. A primeira depende da informação existente na infra-estrutura de rede, como por exemplo, regras que por vezes são usadas pelos ISP para nomearem os nós das suas infra-estruturas e que podem conter informação geográfica. A segunda aproximação baseia-se em informação geográfica submetida por utilizadores ou aplicações em sistemas de uso alargado por milhares ou milhões de utilizadores. A terceira faz uso das duas anteriores para fazer a generalização a blocos de endereços.

Inferência a partir dados de encaminhamento

Uma forma usada para determinar a localização geográfica de um nó de rede é através da análise dos nomes associados aos nós da infra-estrutura de rede que se encontrem perto do nó de destino. Tradicionalmente os ISP nomeiam os nós das suas infra-estruturas por meio de regras internas aos mesmos, que variam de ISP para ISP mas que muitas vezes contêm informação relativa à localização geográfica dos nós. Essa informação encontra-se sobre a forma de nomes de cidades, regiões ou locais, que se encontram regularmente sobre a forma de abreviaturas, ou códigos internacionais de países, e até mesmo códigos de aeroportos. O modo de obter os nomes dos nós que se encontram perto do nó de destino é normalmente através do programa Traceroute. Este tipo de estratégia pode não ser muito precisa, pois para tal seria necessário conhecer as regras de nomeação de todos os ISP, que nem sempre usam regras de nomeação baseadas na localização das infra-estruturas e nem sempre é possível determinar o nome de uma infra-estrutura de rede, ou porque um nó não responde a ICMP (protocolo usado na ferramenta Traceroute) ou porque esta não tem nome atribuído. Uma ferramenta conhecida que usa este tipo de estratégia é a ferramenta GeoTrack [5].

Informação geográfica submetida por utilizadores

Este tipo de informação é obtido tipicamente em aplicações de correio electrónico, redes sociais, sítios *web* onde é feita a previsão do tempo local, guias de TV na Internet ou qualquer outro tipo de aplicação *web* onde o utilizador seja convidado a inserir a sua loca-

lização geográfica [5], [6]. Outra forma de obter este tipo de dados, desta vez involuntariamente, é através de aplicações que fornecem informação que pode estar relacionada com a localização do nó, por exemplo, um browser *web*, ao pedir um recurso *web* a um servidor, tipicamente uma página, fornece geralmente num dos cabeçalhos do pedido HTTP, a linguagem de preferência ou outras definições locais definidas na máquina local. Este tipo de dados apesar de poder fornecer pistas, não fornece dados fiáveis visto que não existe qualquer validação, sendo que os dados poderão ser falsos ou erróneos em qualquer dos casos.

GeoCluster

A técnica denominada por GeoCluster [5] baseia-se em dados de encaminhamento para fazer uma generalização da suposta localização de um conjunto de nós a blocos de endereços IP consecutivos, identificados por prefixos de encaminhamento. Existem listas de blocos de endereços IP associados a sistemas autónomos nas tabelas de endereçamento BGP. No entanto, estes blocos de endereços podem-se distribuir por extensões territoriais muito grandes [7]. Inferências são feitas de modo a subdividir grandes blocos de endereços IP, constituídos por gamas agrupadas por prefixos de endereços, nomeadamente a partir de dados obtidos com recurso aos métodos descritos nas duas subsecções anteriores. Nestes casos, assume-se que se uma determinada amostra de endereços IP pertencentes a um bloco mais alargado se encontra numa localização específica, então todos os IP's dessa gama também se encontram nessa mesma zona. No caso contrário, uma gama de IP's consecutivos pode ser subdividida em intervalos menores, se existir evidência de que estes se localizam em locais distintos. Foi demonstrado que este tipo de generalização incorre em imprecisão e a grandes erros de geolocalização [7], [8].

2.1.3 Técnicas baseadas em medições de latência

Este tipo de abordagem depende da utilização dos chamados *landmarks*. Os *landmarks* ou pontos de referência, são entidades de rede, cujas coordenadas no mundo físico são conhecidas. Esses *landmarks* são habitualmente usados como pontos de partida para medições de latências entre eles próprios e o nó destino, mas também de uma forma mais passiva, sendo eles próprios os alvos de medições. Deste modo, é feita, portanto, uma correspondência entre a latência e a distância física, cuja relação é considerada pobre devido à topologia da rede, em que uma ligação entre dois nós raramente é uma linha recta e cuja

latência depende de vários factores, contendo circuitos e caminhos que nem sempre são os mais curtos possíveis, por vezes devido a políticas dos ISP ou outras situações, como ligações via satélite ou devido às filas de espera existentes para os pacotes de rede, enquanto estes são encaminhados através dos vários pontos intermédios, o que introduz sempre algum atraso. É facilmente demonstrado que este tipo de técnica obtém melhores resultados se aplicada a nós que se encontrem perto de um ou mais *landmarks* devido à acumulação de erro de medição (causada pelos factos descritos) ser menor. Seguidamente são descritas algumas técnicas conhecidas baseadas em *landmarks* que fazem uso deste tipo de aproximação.

GeoPing

A aproximação GeoPing [5] usa um conjunto de *landmarks* para estimar a localização de um nó de destino através da medição da latência entre os *landmarks* e esse mesmo nó. Segundo esta abordagem a localização aproximada do nó será igual à localização do *landmark* ao qual se mediu o mais aproximado padrão de latência, em relação ao padrão de latência medido ao nó de destino, explorando assim uma possível relação entre a distância física e a distância em termos de latência. A forma de obter o padrão de latência mais aproximado para um determinado nó, faz-se através da construção de um vector de latências medidas entre os *landmarks* e esse nó. É então calculada a distância Euclidiana entre esse vector e vectores que constam de um mapa que representa as distâncias entre *landmarks*. O padrão mais aproximado corresponderá assim à menor distância. Este tipo de aproximação muito simples apresenta muitas limitações, nomeadamente a fraca relação entre a distância física e a distância em termos de latência, sendo facilmente dedutível que seria necessária uma distribuição de *landmarks* muito elevada de forma a obter resultados satisfatórios para todas as zonas abrangidas.

Constraint-Based Geolocation

O método Constraint-Based Geolocation (CBG) [9] possui parecenças com o GeoPing, também usa *landmarks* que fazem medidas de latência ao nó de destino, no entanto, o método de estimativa da localização é semelhante a uma técnica de triangulação com os vários *landmarks* sendo estabelecida uma região contínua que é tida como sendo a região geográfica que contém o nó de destino, como ilustrado na Figura 1. De forma a estabelecer esta região, cada *landmark* efectua uma medição de latência ao nó de destino, sendo

esta convertida em distância. Considera-se a velocidade de propagação do sinal em fibra óptica como o factor de conversão, não sendo tidos em conta os atrasos com processamento em nós intermédios e filas de espera, nem a topologia do caminho. A distância estimada é considerada um limite superior em relação à distância real a que se encontrará o nó de destino. Para cada *landmark* é então desenhado um círculo cujo centro se encontra no *landmark* respectivo e cujo raio é a distância calculada. A intercepção desses mesmos círculos definirá a região onde se encontrará necessariamente o nó destino. A localização é então estimada pelo centro dessa mesma região. A relação entre a distância e a latência possui, segundo este método, um factor de correcção de modo a minimizar o erro, ao ter em conta a configuração e carga da rede. O cálculo desse factor é feito, tendo em consideração medições efectuadas entre os diferentes *landmarks*. O erro pode ser estimado a partir do tamanho da região definida.

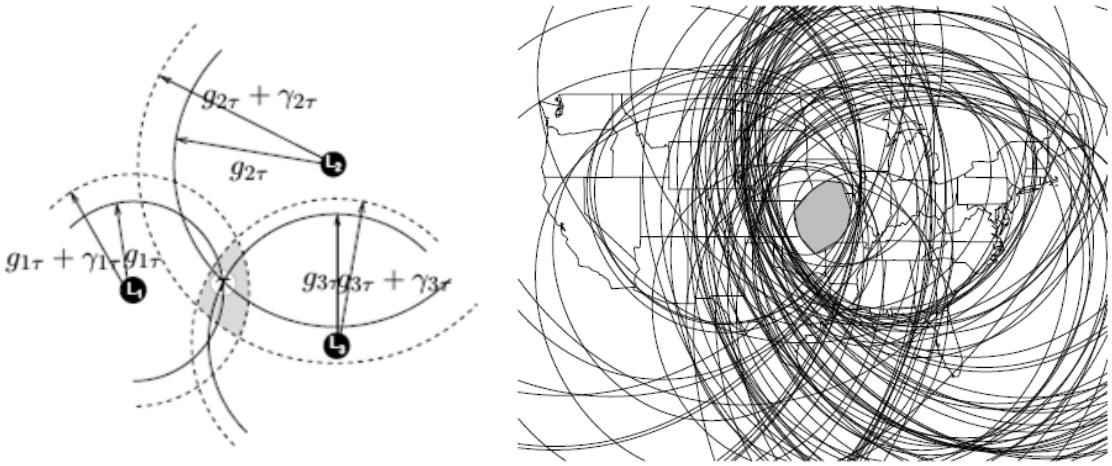


Figura 1 – Figura ilustrativa das técnicas de triangulação do método CBG.

Foi demonstrado [6] que este método é mais preciso que o GeoPing e que métodos baseados em DNS, não sendo, mesmo assim, muito elevada a sua precisão. Verifica-se que a precisão do método GeoPing, quando em comparação com o CBG, é menor quanto menor for o número e boa distribuição de *landmarks*. Para uma probabilidade cumulativa de 80% verificou-se que para dados de um ensaio no território dos Estados Unidos, a distância de erro encontra-se nos 277 km, enquanto que para um ensaio na Europa ocidental se verificou para a mesma probabilidade cumulativa um erro de 134 km.

O método usado para fazer as medições de latência aos nós é através do uso do tradicional ICMP, tal como para a técnica GeoPing, tratada anteriormente. Este tipo de protocolo nem sempre está disponível em todos os nós, sendo por vezes bloqueado como medida de

segurança. Logo, os métodos que dependem primariamente deste recurso, não podem ser considerados completos por não serem aplicáveis nestes casos. Um conjunto elevado de pedidos ICMP a um determinado nó, num curto espaço de tempo, pode ser visto como um ataque. Mesmo que esses pedidos ICMP tenham resposta, de modo a obter uma amostra de latência aceitável é necessário efectuar na ordem da dezena de pedidos. Multiplicando este número pelo número de *landmarks* é facilmente visível que o custo será elevado e uma estimativa final não será obtida de forma imediata, como seria desejável numa aplicação em tempo real.

2.1.4 Topology-based Geolocation

Esta aproximação insere-se numa categoria à parte pois tem a particularidade de fazer uso de diferentes técnicas descritas anteriormente. É tida em conta a topologia da rede ao fazer a estimativa para a localização do nó destino [10]. Pode ser descrita como uma extensão das técnicas vistas anteriormente, nomeadamente Constraint-Based Geolocation, usando as melhores propriedades dessas mesmas técnicas e variações das mesmas de modo a obter melhores resultados. Este método pode ter várias variantes: Usando apenas *landmarks* activos, que efectuam medições, e usando *landmarks* activos e passivos, sendo que estes últimos não efectuam medições. Podem ainda ser usados *landmarks* activos e passivos em conjunção com dados de inferência de localização obtidos a partir de dados de encaminhamento, tais como os descritos na subsecção: "Inferência a partir dados de encaminhamento", que são validados segundo medições de latência. O modo usado para determinar a topologia da rede é através do uso da ferramenta Traceroute, através da qual são feitas estimativas de latência das ligações entre os nós que se ligam ao nó de destino. Segundo [10], esta técnica obtém melhores resultados que as descritas anteriormente, tendo no entanto um custo que é previsivelmente bastante mais elevado para a rede.

2.1.5 Sistemas Comerciais

Para além dos sistemas anteriormente enunciados, existem ainda sistemas comerciais, geralmente proprietários, cuja natureza e modo de funcionamento total ou parcial, são de um modo geral desconhecidos. As aplicações destes sistemas são as mais variadas e podem ser vistas pelo comum utilizador da Internet enquanto navega normalmente pelo espaço virtual. Procurando de uma forma mais direccionada é simples encontrar sistemas que de uma forma gratuita nos indicam qual o nosso país, por vezes até a nossa cidade,

com alguma exactidão usando apenas o endereço IP. Apesar de os métodos usados por estes sistemas serem no seu geral desconhecidos, pelos seus resultados, bem como pela informação que por vezes recolhem, é possível inferir alguns dos mecanismos a que estes recorrem. Os métodos usados por estes sistemas vão desde os mais simples e menos precisos (normalmente gratuitos), recorrendo a bases de dados *whois* ou a informação contida nos cabeçalhos HTTP dos browsers, até a um conjunto de sistemas mais complexos que fornecem localizações bastante fiáveis através do "clustering"[3] de endereços IP, bem como sistemas que previsivelmente usarão alguns dos algoritmos vistos anteriormente ou variantes dos mesmos, bem como uma união de vários sistemas de forma a obter dados mais fiáveis, numa lógica de validação em várias frentes. Para além dos aspectos já referidos, existem empresas, normalmente de carácter multinacional, que fornecem este tipo de serviços, que têm relações privilegiadas com alguns ISP e empresas que detêm espaços de endereçamento próprios e que por isso conseguem obter informação importante sobre a infra-estrutura da rede e sobre os seus próprios métodos de organização espacial do espaço de endereçamento, que de outra forma lhes estaria vedada. Esta posição privilegiada é normalmente obtida em troca de serviços que estas empresas disponibilizam, como a distribuição de conteúdos e os serviços de geolocalização que permitem, entre outros, a filtragem potenciais atacantes tendo por base países ou localizações conhecidas por terem índices de perigosidade acrescida.

2.1.6 Balanço

Sobre as técnicas de localização espacial ou geolocalização é possível inferir que de entre os sistemas e técnicas vistos não é possível determinar um que constitua uma solução infalível para este problema, visto que seria muito complicado obter uma solução que fosse fiável de uma forma simples, em tempo útil, baseado em aplicações em tempo real, e de carácter seguro. Os sistemas referidos anteriormente como sistemas baseados em bases de dados públicas estão disponíveis de uma forma gratuita e de fácil consulta, no entanto não possuem fiabilidade suficiente para serem usados de uma forma sistemática e rigorosa. Os sistemas de captura de informação, como os sistemas que usam informação submetida por utilizadores e aplicações também não são fiáveis por potencialmente darem respostas falsas e/ou incompletas. Sistemas que se baseiam em informação que consta dos caminhos de rede também falham em dar informação fiável, pelo menos a um nível glo-

[3] Termo em inglês cuja tradução directa para português é "agrupamento"

bal, pois seria necessário conhecer as regras de nomeação dos diferentes provedores de acesso à Internet. Mesmo quando esta informação existe pode estar sujeita a ambiguidades e incorrecções. Por outro lado, o Traceroute e o Reverse DNS Lookup nem sempre estão disponíveis, o que faz com que os sistemas que dependem de informação por estes fornecida sofram de considerável incompletude. O agrupamento de endereços IP em grupos, o chamado *clustering*, de acordo com regras que se baseiam nas técnicas anteriores, sofre dos mesmos males que foram descritos para essas mesmas técnicas, podendo essa generalização ser indutora de erros suplementares. No entanto, este agrupamento quando validado por uma técnica que se baseie em medições poderá ser validado de uma forma mais aceitável, ou seja, fica desta forma demonstrado que quando se juntam estes dois tipos de técnica podem ser obtidos resultados bastante aceitáveis, tal como no sistema Topology-based Geolocation (2.1.4). As técnicas baseadas em medições de latências possuem sobretudo limitações ao nível de rigor da localização, devido ao facto de a as características da rede não permitirem fazer uma correspondência directa entre o tempo de propagação e a distância real, mas também ao nível do custo deste tipo de medições para a rede a nível global e o tempo necessário para obter uma estimativa aceitável. Os sistemas comerciais, por beneficiarem de informação restrita e por usarem técnicas anteriormente descritas mas validadas com informação suplementar, possuem provavelmente os melhores resultados, de forma imediata. É de notar que para a generalidade das técnicas estudadas, o uso de *proxies* e equipamento de tradução de endereços (e.g. NAT) contribui para a sua imprecisão, sendo que os sistemas que usam medidas de latência são os menos afectados por estas condições.

2.2 Sistemas de localização virtual na rede

Os sistemas de localização na rede tentam obter localizações relativas num espaço virtual, que de uma forma bem definida relaciona os nós entre si em termos de latência, dado apenas o seu endereço IP. Distinguem-se dos sistemas de localização espacial, ou geolocalização, por não terem como objectivo o cálculo ou estimativa da localização física de um nó arbitrário, não tentando fazer a ligação, sempre difícil, entre a rede global e o mundo físico. Em vez disso, estes sistemas tratam a rede como um sistema independente da geografia. A sua representação tem em conta factores como a topologia da rede, nomeadamente a latência que dela surge, como um factor de proximidade ou uma medida para estimar uma localização num sistema de coordenadas próprio, as chamadas coordenadas

virtuais. Ao longo do tempo em que tem sido estudado este problema, a evolução tem sido notória, no entanto existe ainda um caminho a percorrer, visto que os sistemas até agora estudados poderão ser melhorados no sentido de obter melhores resultados, nomeadamente ao nível da sua exactidão. Tal como no problema da geolocalização, este problema tem sofrido várias abordagens, nas quais se distinguem sistemas diversos para atacar o problema. Segue-se uma secção dedicada às mais relevantes abordagens e sistemas culminando numa síntese crítica sobre os mesmos.

Nesta secção são identificadas as duas abordagens mais estudadas para este problema: a que se baseia em coordenadas virtuais; e a que dispensando essas coordenadas, faz apenas a estimativa da proximidade entre nós da rede.

2.2.1 Sistemas baseados em coordenadas virtuais

Os sistemas baseados em coordenadas virtuais fazem geralmente uma representação da Internet como um espaço virtual com múltiplas dimensões e formas (coordenadas planas, esféricas, hiperbólicas, etc.). Em oposição às técnicas de localização geográfica, este espaço que pode assumir várias dimensões, é apenas virtual, e não tenta reproduzir ou localizar os nós segundo a sua localização física real, desligando-se do espaço físico. Seguem-se descrições de alguns sistemas que se destacaram nesta subcategoria.

GNP, NPS e Lighthouse

Neste capítulo é feita uma pequena análise do sistema GNP [11] e de dois sistemas que podem ser considerados derivados deste: O NPS [12] e o Lighthouse [13].

O sistema GNP (Global Network Positioning) é baseado em coordenadas, calculadas tendo por base uma modelação da Internet como um espaço geométrico. Neste sistema os nós mantêm as suas próprias coordenadas, o que lhes permite calcular facilmente a sua distância a qualquer outro nó nesse espaço geométrico virtual (Figura 2). A arquitectura deste sistema é composta essencialmente por duas partes. A primeira parte é composta por um conjunto de nós distribuídos, designados por *landmarks*. Os *landmarks* calculam as suas próprias coordenadas num espaço geométrico escolhido. A segunda parte é constituída por todos os restantes nós, que podem calcular as suas próprias coordenadas em relação às coordenadas calculadas pelos *landmarks* e que servem de base de referência para o sistema.

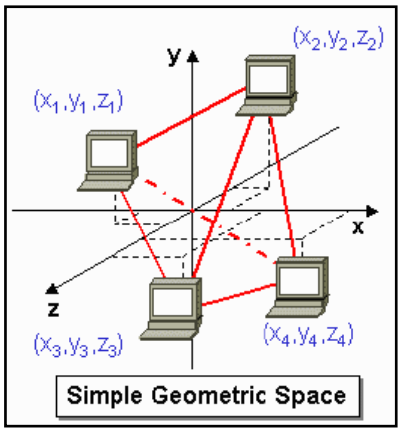


Figura 2 – Esquema ilustrativo do espaço geométrico empregue pelo sistema GNP.

A forma de cálculo destas coordenadas usa medições de latências entre os *landmarks*, que corresponderão às distâncias calculadas neste espaço virtual, sendo o objectivo principal encontrar as coordenadas que minimizem o erro entre as distâncias medidas e as distâncias segundo esse mesmo espaço de coordenadas. O problema da derivação das coordenadas pode então ser visto como um problema genérico de minimização global em múltiplas dimensões, que pode ser resolvido de forma aproximada por vários métodos disponíveis, incluindo o método que os autores usaram na sua demonstração original: *Simplex Downhill*.

Ao contrário do pioneiro sistema IDMaps (subsecção 2.2.2), este sistema não se baseia num esquema cliente-servidor o que, segundo os autores, tem vantagens em termos de escala, podendo, por exemplo, ser usado numa arquitectura *peer-to-peer*. Foi ainda demonstrado [11] que este sistema é mais robusto e possui uma maior exactidão que o sistema IDMaps descrito na subsecção 2.2.2.

O sistema NPS (Network Positioning System) é uma versão do GNP com maiores preocupações em relação a sobrecarga que possa ser causada a nível dos *landmarks*, assim sendo, este é constituído por um sistema hierárquico construído de forma a reduzir a carga nos *landmarks*. Além desta modificação é também incluído um sistema que tenta minimizar os efeitos de nós maliciosos, bem como um mecanismo de controlo de congestionamentos. As coordenadas neste sistema estão constantemente a ser recalculadas tendo em conta as frequentes actualizações que são efectuadas através de comunicações com os *landmarks*. No caso do NPS, o algoritmo é descentralizado. Ao invés de enviar as latências medidas para nós centralizados que correm o algoritmo (tal como é feito no GNP), cada *landmark* faz os seus próprios cálculos sempre que são efectuadas novas medições.

O sistema Lighthouse pode ser visto como uma extensão do sistema GNP, com maiores preocupações em termos de escala. Tal como o GNP também possui um conjunto de *landmarks*, no entanto, sempre que um novo nó inicia a participação no sistema, este não necessita de interrogar as *landmarks* de modo a obter as suas coordenadas, podendo apenas interrogar qualquer conjunto de nós, obtendo as coordenadas relativas àquele conjunto, transformando-as posteriormente em coordenadas relativas aos *landmarks*, ou seja, neste caso qualquer conjunto de nós com coordenadas atribuídas pode funcionar como um *landmark*, tal como já acontecia com o NPS. Em relação ao GNP, o modelo matemático usado para computar as coordenadas é diferente, bem como o modo como um nó escolhe os seus *landmarks*.

Vivaldi

O sistema Vivaldi [14] baseia-se num algoritmo que atribui coordenadas virtuais aos nós participantes de modo a que essas mesmas coordenadas, quando aplicadas a dois nós, reflictam de uma forma aproximada a latência esperada entre eles, fazendo simultaneamente uma adaptação dinâmica a mudanças que ocorrem na rede. Ao contrário dos sistemas da família GNP, descritos anteriormente, o mecanismo por de trás deste sistema não pressupõe uma infra-estrutura com nós especiais. Um nó ao entrar no sistema escolhe coordenadas arbitrárias e à medida que são trocadas mensagens com os outros nós, são medidas as latências entre nós e trocadas as coordenadas virtuais. À medida que este processo avança, os nós vão-se posicionando de forma que as coordenadas virtuais reflictam a latência de rede. Este reposicionamento ocorre em pequenos passos controlados por um factor baseado no erro com que as coordenadas correntes foram estimadas. Sempre que os nós comunicam entre si estes medem a latência e obtêm as coordenadas do outro nó, bem como o erro relativo calculado por este. Estes dados simples permitem reajustar as coordenadas pelo processo anteriormente descrito, tendo em conta os erros relativos medidos no nó local e no nó remoto de modo a que no fim seja minimizado o erro global. O constante reposicionamento dos nós do sistema, pode ser comparado com uma estrutura de massas ligadas por molas (Figura 3), em que as massas são os nós, cada um com o seu peso relativo, isto é, o seu erro relativo associado, equivalendo as molas às forças que reajustam constantemente as coordenadas virtuais dos nós.

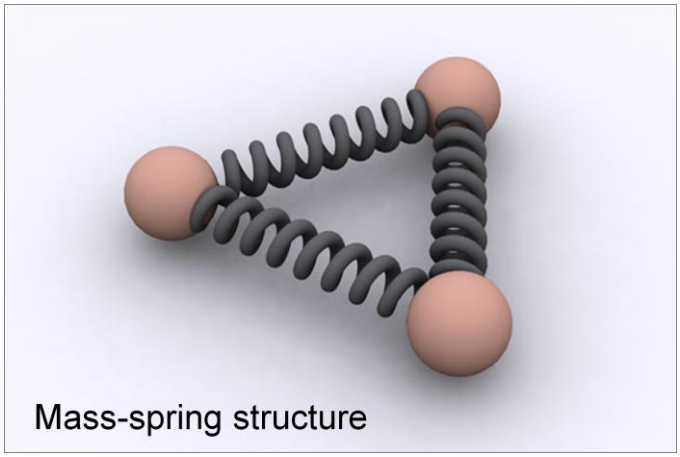


Figura 3 – Analogia do sistema Vivaldi com uma estrutura de massas ligadas por molas.

De modo a obter uma melhoria na precisão do método, foi proposta [14] a introdução da noção de altura nas coordenadas, por complementaridade a sistemas que possuem apenas coordenadas em duas ou em três dimensões. Um vector de altura consiste numa coordenada Euclidiana em duas dimensões, incrementada de uma altura. A porção Euclidiana modela um núcleo da Internet a alta velocidade, com latências proporcionais à distância geográfica, enquanto a altura modela o tempo necessário para um pacote viajar desde o nó de origem até ao núcleo de rede. Os resultados obtidos quando usadas coordenadas de duas dimensões e altura, por oposição à utilização de apenas duas ou três dimensões demonstraram ter um menor erro cumulativo.

O algoritmo é completamente distribuído, correndo versões iguais em todos os nós, sendo facilmente incluído num sistema *peer-to-peer*. As coordenadas virtuais de um nó estão em constante actualização, o que faz com que este possa reagir a mudanças na rede. A partir do momento em que um nó inicia a sua participação no sistema, este possui um tempo de convergência, que consiste no tempo que um nó demora até obter coordenadas consideradas estáveis. Foi verificado que usando um intervalo de tempo menor entre actualizações a convergência é mais rápida. No entanto, mesmo com convergências rápidas, o tempo que um nó demora a obter coordenadas estáveis é da ordem das dezenas de segundos. Ou seja, usando este algoritmo não é possível obter estimativas, cujo erro seja mínimo, de uma forma imediata.

Foi demonstrado em ambiente de simulação que este sistema atinge uma taxa de erro equivalente ao sistema GNP, que é um sistema que possui uma infra-estrutura que faz uso de *landmarks*, o que pode ser considerado um progresso, visto que o sistema Vivaldi não

possui essa noção, usando, no entanto, medições entre a generalidade dos nós participantes, não dependendo de *landmarks* como nos sistemas semelhantes ao GNP. Isto pode ser visto como um factor gerador de sobrecarga para este sistema, visto que é esperado que o conjunto de medições efectuadas no sistema Vivaldi seja substancialmente superior. No entanto, no contexto de um sistema *peer-to-peer*, em que por si só os nós já comunicam entre si de forma frequente, sendo que isso faz parte do algoritmo normal de actualizações deste tipo de sistema, a sobrecarga gerada não seria significativa, podendo facilmente fazer parte integrante desse mesmo algoritmo *peer-to-peer*.

Tal como o sistema Vivaldi, outros sistemas existem que fazem um ajuste constante das coordenadas de acordo com uma noção equivalente à de um campo de forças no qual os nós se posicionam e se ajustam relativamente. De entre estes sistemas destaca-se o sistema Big-bang Simulation [15], um sistema anterior ao Vivaldi, no qual, a cada nó, correspondente a uma partícula, é associado um momento, que possui uma energia cinética e uma noção de atrito ou fricção, de acordo com o referido campo de forças. No estado inicial todos os nós se encontram no mesmo ponto, divergindo de acordo com a energia cinética dada pelo campo de forças, como numa explosão. Posições mais estáveis são atingidas quando a força de atrito se sobrepõe à energia cinética. Os autores do sistema Vivaldi argumentam que o sistema Big-bang Simulation tem uma maior complexidade e possui uma necessidade de um maior conhecimento global do sistema, não sendo clara a forma de o descentralizar.

Apreciação crítica

Os sistemas de coordenadas virtuais tentam modelar a rede segundo um espaço de coordenadas que se baseia nas distâncias relativas estimadas entre nós. Estes sistemas deparam-se com vários problemas de fundo nessa sua tentativa. A ideia de tentar inserir cada nó num espaço de coordenadas é muito poderosa pois sendo conseguida de forma fiável é uma maneira manter sempre distâncias relativas segundo um espaço global tal como qualquer sistema de coordenadas no mundo real. Tanto os sistemas da família do GNP como o sistema Vivaldi fazem este tipo de modelação, no entanto, o sistema Vivaldi não possui uma infra-estrutura de nós especiais, o que por um lado é uma vantagem mas por outro lado trás uma desvantagem associada, que é o facto de um nó participante não poder obter coordenadas virtuais aceitáveis de uma forma imediata, por se basear num algoritmo que vai convergindo até estabilizar, tendo por base um número substancial de medi-

ções de latência, que quando efectuadas no contexto de um sistema *peer-to-peer*, não teriam um peso demasiado elevado para a rede, sendo apenas uma das componentes desse sistema e fazendo parte integrante do seu normal regime de actualizações entre nós, que qualquer sistema deste tipo possui. As versões do algoritmo GNP que possuem maiores preocupações em termos de descentralização e de escala (NPS e Lighthouse) podem ser consideradas verdadeiras melhorias, por minimizarem algumas fragilidades do sistema GNP original.

Para os sistemas de coordenadas virtuais estudados, como a função distância é uma função da latência, esta pode ser considerada adequada. Tal como no mundo real, a distância real é uma função da velocidade e do tempo, na rede a distância apenas pode ser vista como uma função da latência. No entanto, a característica variável e imprevisível da latência na rede global é o factor determinante para que modelar a rede com um modelo fixo de coordenadas, seja um problema difícil. O sistema Vivaldi sendo adaptativo e dinâmico tende a ser o que melhor replica as características da latência.

2.2.2 Sistemas de proximidade relativa

Ao contrário dos sistemas anteriormente tratados, estes dispensam as chamadas coordenadas virtuais, centrando-se apenas no cálculo ou estimativa da distância em termos de rede, ou seja, a latência. Esta grandeza representa na rede o que se pode designar por proximidade, sendo assim, os nós são classificados segundo a sua proximidade ao nó de origem. De seguida são destacadas as características principais de algumas das técnicas mais relevantes que se inserem nesta categoria.

IDMaps

O sistema IDMaps [16], [17] foi o sistema pioneiro de entre os sistemas de cálculo de proximidade relativa. Este sistema tem como uma das principais características o funcionamento em complementaridade com um sistema composto de servidores pertencentes a uma infra-estrutura designada de HOPS ou SONAR. Estes servidores mantêm uma topologia virtual da Internet, efectuando cálculos com base em dados de distância fornecidos por nós especiais do sistema IDMaps, os quais, os autores designaram de “tracers” (Figura 4). A distância estimada entre um par de nós é então calculada somando a distância entre o nó de origem e o *tracer* mais próximo, a distância entre o nó destino e o *tracer* mais próximo, e a menor distância entre os dois *tracers* usados. Os endereços IP são divi-

didos em conjuntos de prefixos, sendo que os *tracers* serão localizados de modo a que haja sempre uma relativa proximidade entre pelo menos um *tracer* e um prefixo de endereços. Assim sendo, a exactidão deste método aumenta com o aumento do número de *tracers* usados mas também com a correcta localização destes em relação aos conjuntos de endereços IP, definidos pelos seus prefixos, ou seja, os *tracers* deverão estar distribuídos pela rede de modo a que a distância máxima de um *tracer* ao conjunto de endereços definidos pelo seu prefixo, seja minimizada. A função principal dos *tracers* é fazer estimativas de distância para os outros *tracers* e para os prefixos de endereços IP. Toda essa informação é então propagada para os "clientes" do sistema, neste caso, os servidores HOPS ou SONAR que constroem e mantêm uma topologia virtual da rede, baseada nessa mesma informação. Estes servidores são interrogados pelos nós de modo a obterem a estimativa de distância, constituindo-se como a interface visível de todo o sistema.

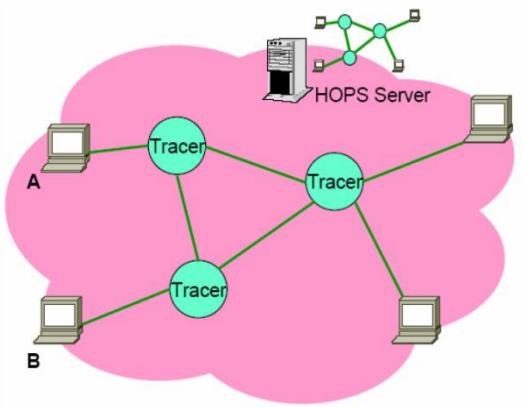


Figura 4 – Esquema básico da estrutura do sistema IDMaps.

King

A ferramenta King [18] emprega consultas DNS recursivas para fazer uma estimativa da latência entre dois nós arbitrários, baseando-se no princípio de que um servidor DNS para um determinado domínio de destino se encontra geograficamente/topologicamente próximo dos nós desse mesmo domínio (Figura 5). Esta estimativa pode ser feita sem usar demasiados recursos, usando a infra-estrutura existente do sistema DNS, podendo atingir níveis de precisão bastante elevados, isto porque foi verificado que o princípio anteriormente enunciado é verificado na maioria dos casos. No entanto, nos casos em que o princípio enunciado não se verifica os resultados podem ser decepcionantes. Este método apenas pode ser utilizado quando o nó possui um nome DNS (e.g., www.fct.unl.pt) ou quando é possível efectuar um Reverse DNS Lookup [2] do endereço IP respectivo de

modo a obter esse mesmo nome DNS. Esta constitui-se como uma clara limitação do método. Uma particularidade desta ferramenta é o facto de poder ser utilizado para estimar a latência entre dois nós, sem a participação de nenhum deles.

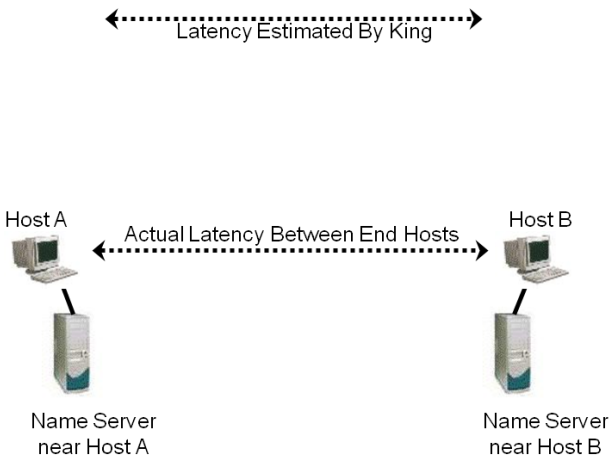


Figura 5 – Forma como a ferramenta King estima a latência.

O método usado pela ferramenta King para estimar a latência entre dois nós usa medições de latência entre servidores de nomes DNS próximos dos nós em causa, baseando-se no princípio anteriormente referido. Para medir a latência entre dois servidores de nomes DNS, esta ferramenta efectua uma consulta recursiva ao a um dos servidores DNS pedindo para resolver um nome associado ao domínio de autoridade do segundo servidor DNS, medindo desta forma o tempo necessário para resolver esta consulta. A este tempo total é subtraída a latência entre o cliente e o primeiro servidor de nomes, sendo esta medida através da utilização da ferramenta Ping, que usa o protocolo ICMP, ou de uma consulta DNS iterativa.

A utilização da infra-estrutura do sistema DNS, apesar de ter a vantagem de não serem necessárias novas infra-estruturas, pode ter alguns inconvenientes que no limite podem ser gravosos. Nomeadamente, sendo esta uma utilização não prevista desse sistema, pode ser vista como abusiva, podendo levar a que sejam tomadas medidas restritivas, nomeadamente impedindo que os servidores usem consultas DNS recursivas, que não são necessárias ao bom funcionamento do sistema DNS. Como este tipo de utilização não foi prevista, uma utilização em larga escala poderia levar a uma sobrecarga dos servidores DNS, o que poderia ter consequências graves no desempenho da Internet como um todo, sendo que nesse caso o método teria mais inconvenientes que benefícios.

Meridian

O sistema Meridian [19] tem como principal objectivo proporcionar um *framework* que providencia um método para a selecção de nós, baseado na sua localização relativa em termos de latência. Este mecanismo pode ser incluído num sistema *peer-to-peer* como uma ferramenta de localização. O método pelo qual é obtida esta localização é através de medições de latência. Cada nó mantém um conjunto de *m* anéis concêntricos. Cada anel representa uma distância, que aumenta com a distância ao centro de uma forma exponencial (Figura 6). Um número pequeno e fixo de nós, *O(log N)* (sendo *N* o número de nós do sistema), fica organizado segundo esses anéis, ou seja, de acordo com a latência medida. O algoritmo de selecção de nós tenta manter uma distribuição de nós pelos anéis, evitando a concentração de nós apenas em alguns anéis. Deste modo é obtida uma maior dispersão de distâncias. Para cada anel, o número fixo de nós é escolhido, sendo seleccionados os anéis de acordo com a sua distribuição espacial. Com alguma frequência os membros de cada anel são reavaliados segundo a sua distribuição espacial, sendo escolhidos preferencialmente nós que possuam uma maior distribuição geográfica ou espacial relativamente aos restantes nós do anel. A forma de propagar a informação sobre os membros, bem como a descoberta dos mesmos, é através do uso de um algoritmo escalável para notificar os nós sobre os membros do sistema. O protocolo usado baseia-se num algoritmo de "gossip". Esse algoritmo escolhe aleatoriamente um nó de cada um dos anéis e envia a sua informação "gossip" sobre um nó escolhido aleatoriamente em cada anel. O nó destino faz medidas de latência a esses nós, incluindo o nó de origem, colocando-os como membros secundários dos seus anéis, consoante a latência medida.

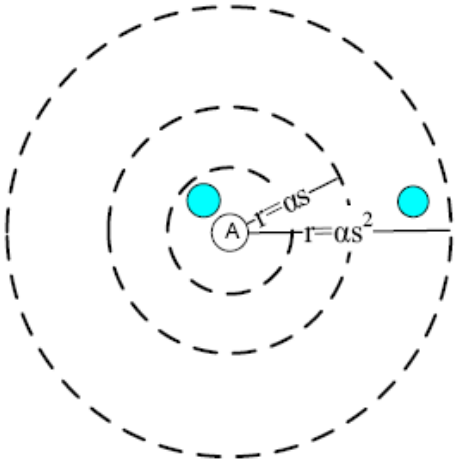


Figura 6 – Estrutura de anéis concêntricos.

Este sistema não usa qualquer infra-estrutura especial. Deste modo, todos os nós pertencentes, usam o mesmo algoritmo, não existindo nós com um papel especial como se verifica em alguns métodos descritos neste documento. Sempre que um novo nó inicia a sua participação deve conhecer o endereço de algum nó participante. Neste caso, a periodicidade de envio de pacotes de "gossip" é maior para permitir que mais nós possam incluir o novo nó no sistema.

Netvigator

O sistema Netvigator (Network Navigator) [20] faz a estimativa de proximidade e latência medindo a latência a um conjunto predefinido de *landmarks*, mas também a nós que ficam no caminho para essas *landmarks*, os chamados "milestones" (Figura 7). Em cada nó, os nós a localizar são ordenados segundo a sua proximidade estimada. O uso dos *milestones* tem como ideia principal evitar que seja feito um falso agrupamento de nós, apenas por terem latências semelhantes, tal como acontece no sistema Meridian. Desta forma é incrementada a precisão do método Netvigator. Os autores deste método fizeram uma análise comparativa com os métodos GNP e Vivaldi, que usam coordenadas virtuais. Estas análises, feitas numa rede empresarial e na rede global, mostram que o sistema Netvigator apresenta melhores resultados que estes dois sistemas. Esta análise comparativa foi considerada em certa medida inadequada, pois o Netvigator usa um número bastante superior de *landmarks* (considerando também como *landmarks* os *milestones*). As comparações em termos de sobrecarga também podem ser consideradas inadequadas, pois não têm em conta a sobrecarga causado pelas medições de latência aos *milestones*. É ainda de apontar que uma comparação com o sistema Meridian seria mais desejável, visto que ambos os métodos têm a abordagem de não usar coordenadas virtuais, mas sim estimativas de proximidade.

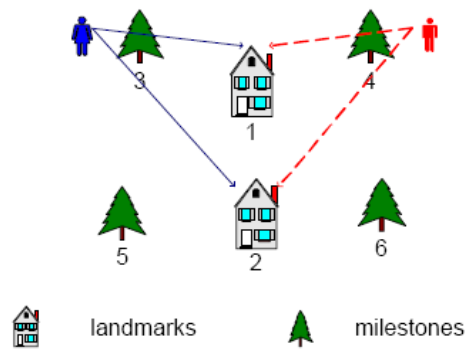


Figura 7 – Estrutura composta por *landmarks* e *milestones*.

Apreciação crítica

Os sistemas de proximidade relativa vistos ao longo desta subsecção foram seleccionados tendo em conta a sua importância no estudo dos sistemas de proximidade relativa que dispensam coordenadas virtuais. O sistema IDMaps foi um sistema que serviu como uma base de estudo e referência para os sistemas que se seguiram, no entanto, as suas características pouco adaptativas e muito dependentes da localização dos seus nós especiais, os "tracers", em função da localização dos prefixos de rede, de modo a poder ter uma performance aceitável, tornam a sua usabilidade real pouco provável.

O sistema King, de cariz mais *ad-hoc*, sendo bem pensado a nível do seu funcionamento e forma de aproveitamento da infra-estrutura DNS existente, obtendo resultados que apesar de não serem de uma precisão muito elevada podem servir de ponto de referência em testes de outros sistemas. O seu princípio de funcionamento no qual os servidores DNS para um determinado domínio, se encontrarem topologicamente próximo da generalidade dos nós desse domínio nem sempre se verifica e a utilização do sistema DNS, o qual não foi construído com este fim, não é uma solução que possa ser adoptada a nível global.

Os sistemas Meridian e Netvigator constituem soluções por si só, sem contar com o uso de infra-estruturas existentes como o sistema King, sendo os primeiros de implementação mais simples bem definida que o sistema IDMaps. Estes dois sistemas constituem-se como soluções de proximidade relativa entre os nós participantes, sendo estes organizados segundo essa proximidade em ambos os casos, mas de forma diversa. O sistema Meridian constitui-se como a solução mais descentralizada, sendo que ambos os sistemas obtêm uma performance, ao nível de precisão, mais elevada que o sistema King, mas com um custo para a rede superior no que diz respeito ao número de medições a efectuar de modo a manter a sua organização de acordo com as distâncias relativas dado que fazem as medições no momento. O sistema Meridian é o que efectua um maior número de medições, devido ao facto de as efectuar de cada vez que procura uma resposta, mas provavelmente obtém os melhores resultados. Esta assumpção não possui, no entanto, qualquer validação, visto que não foi efectuada qualquer comparação prática entre os dois métodos. Por não tentarem construir um sistema genérico que obtenha estimativas para latência entre dois nós arbitrários, estes sistemas centram-se no problema da proximidade, sendo ignorado o problemático e complexo problema associado à estimativa de distâncias, para nós cuja maior distância possa impedir uma boa precisão dos métodos, como no caso dos

sistemas de coordenadas virtuais. Essa especialização tendo em vista menores distâncias permite que estes sistemas possam obter melhores resultados nestas situações em particular.

2.3 Balanço e perspectivas de utilização dos algoritmos estudados

Neste capítulo foram apresentados diversos sistemas segundo duas vertentes principais: Os sistemas de localização geográfica de nós na rede ou sistemas de geolocalização IP e os sistemas de localização relativa e virtual na rede. Apesar de estes dois grupos de sistemas serem relativos a localizações para nós da rede, é necessário distingui-los e definir quais as contribuições que sistemas de um e de outro tipo podem dar.

Os sistemas de localização geográfica de nós da rede permitem inferir uma localização física estimada dos nós participantes, dado o seu endereço IP, estimativa essa, cuja precisão depende sobretudo do método usado, mas também das condições relacionadas com a topologia da rede em que os nós se encontram.

Devido ao facto de os sistemas de localização geográfica IP relacionarem os endereços IP com a suposta localização física dos nós em causa, estes apenas servem esse mesmo propósito especificamente, não sendo os métodos mais indicados para obter uma função de distância válida na rede. Isto deve-se ao facto de a topologia de rede não se poder relacionar directamente com a topologia física terrestre, sendo que essa relação é de facto bastante pobre. A situação concreta na qual dois nós se encontram fisicamente próximos mas cuja distância em rede, ou seja, a latência, é grande ocorre de uma forma frequente, o que acontece devido a elementos variados: pouca capacidade das ligações, tempo perdido em filas de espera, ou simplesmente porque apesar de os nós poderem estar próximos fisicamente, estes poderem pertencer a domínios diversos. Tendo em conta estes factores, é possível inferir que não será simples obter uma função de distância adequada para estabelecer relações entre localizações virtuais ou sintéticas, que reflictam a topologia de rede através dos sistemas de geolocalização IP.

Para além destes factores, os sistemas de geolocalização IP não podem facilmente ser testados em ambientes que não tenham alguma associação com a rede real, isto é, com localizações físicas reais associadas aos nós participantes, ou em alternativa usando conjuntos de dados reais onde essa associação esteja presente, sendo que nesse caso as localizações físicas devem ser consistentes com as latências entre os nós. Isto obrigaria à existência de

vastos e rigorosos conjuntos de dados, recolhidos na Internet, o que não existe disponível ou cuja veracidade não pode ser confirmada facilmente.

Os sistemas de localização virtual na rede (*virtual network location systems*) podem ser vistos segundo duas perspectivas distintas mas que visam providenciar algum tipo de localização relativa na rede, havendo uma divisão entre os que produzem uma modelação aproximada da rede usando coordenadas virtuais e os que apenas se focam em distâncias relativas tendo em conta uma função de proximidade em termos de latência que geralmente se baseiam em medições no momento.

No caso dos sistemas de proximidade relativa que foram estudados existem alguns impedimentos à integração em ambientes de teste, nomeadamente no caso do sistema IDMaps. Como já foi referido, este sistema depende de uma infra-estrutura de implementação difícil e que não está disponível, pelo que não seria possível replica-lo. Em relação ao sistema Meridian e ao sistema Netvigator, cujas naturezas possuem algumas semelhanças, a dificuldade não se prende com o ambiente de teste ou com infra-estruturas, até porque tal como os sistemas de coordenadas virtuais, estes também fazem uso das latências entre os nós para estimarem a proximidade. Neste caso, é a integração com o sistema LiveFeeds (que será descrito no capítulo seguinte) que é mais difícil. O sistema Meridian, por exemplo, apenas permite obter em cada momento, um subconjunto de nós que se encontram próximo de um dado nó, no entanto, o sistema LiveFeeds funciona como um sistema *full mesh* em que cada nó pode comunicar directamente com qualquer outro nó, sendo que, no caso do algoritmo de difusão do LiveFeeds os nós com os quais se vai comunicar em seguida são determinados por uma selecção de entre um intervalo de identificadores (estes processos são descritos em mais pormenor no capítulo seguinte), pelo que seria necessário ter um modo de relacionar todos os nós entre si, o que não vai de encontro ao modo de funcionamento de um sistema com as características, quer do sistema Meridian como do Netvigator, estando mais de acordo com os sistemas de coordenadas virtuais, que permitem em cada momento e de forma imediata obter um relacionamento entre quaisquer nós participantes, dado pelas suas coordenadas virtuais estimadas.

A modelação da rede pelos sistemas de coordenadas virtuais tem o objectivo simples de para cada par de nós, em cada momento, ser possível conhecer uma estimativa da latência sem ter de efectuar medições caso a caso, isto porque esse tipo de medições seria muito caro para a rede ou até mesmo impossível, pela sobrecarga que causaria.

Assim, tendo em conta os impedimentos verificados para os restantes sistemas, restam os sistemas baseados em coordenadas virtuais, que podem ser considerados como as alternativas mais consistentes para serem testadas neste contexto, visto que apenas baseiam a sua estimativa de localização virtual em medições de latência, posicionando os nós entre si de acordo com essas mesmas medições.

Nos capítulos seguintes são apresentados, o sistema LiveFeeds e os resultados da sua integração neste sistema de alguns dos algoritmos estudados neste capítulo, sendo finalmente apresentados resultados sobre os melhoramentos que possivelmente estes possam trazer ao algoritmo de difusão do sistema LiveFeeds. Como é possível verificar nos capítulos seguintes, dos vários algoritmos apresentados neste capítulo, apenas dois destes foram integrados e fazem parte do estudo que se segue, neste caso, versões dos algoritmos GNP e Vivaldi. As razões que levaram à escolha de apenas estes e não de outros algoritmos, prendem-se com vários factores, os quais passarei a enunciar em seguida.

Tanto o algoritmo GNP como o Vivaldi, são sistemas que produzem coordenadas sintéticas ou virtuais tendo por base as latências entre os diversos nós. Devido a esse facto, estes sistemas permitem uma fácil integração em ambientes de teste diversos.

Mesmo usando apenas sistemas baseados em coordenadas virtuais, existiriam outros sistemas que poderiam fazer parte do estudo que se segue, no entanto, tal seria muito complicado, dado o tempo que existe disponível para completar esta dissertação. Daí, ter-se optado por escolher dois sistemas representativos sistemas de coordenadas virtuais, mas de paradigmas diferentes: o sistema GNP, mais estático e que usa nós especiais que não fazem parte dos nós participantes e o sistema Vivaldi, um sistema mais orientado ao paradigma *peer-to-peer*, sendo um sistema dinâmico e que não possui nós especiais para além dos que são participantes no próprio sistema *peer-to-peer* no qual esteja integrado.

Tendo em conta o que foi referido, o estudo que foi levado a cabo ao longo desta dissertação culmina, nos capítulos finais, com a análise dos resultados obtidos neste contexto e para os referidos sistemas, bem como a apresentação das conclusões que foi possível retirar desses mesmos resultados.

3. Algoritmo Livefeeds e seus melhoramentos através de técnicas de coordenadas virtuais

Neste capítulo é apresentado de forma breve o projecto LiveFeeds, com especial ênfase no seu algoritmo de difusão, cujo melhoramento através de sistemas de coordenadas virtuais constitui o principal objecto de estudo desta dissertação.

São apresentadas considerações sobre hipóteses de melhoramentos desse algoritmo tendo em consideração a latência da comunicação. Finalmente discute-se de que forma os algoritmos para estimativa de coordenadas virtuais podem ajudar nesse objectivo.

3.1 Apresentação do modelo do sistema

O objectivo do projecto LiveFeeds é desenvolver técnicas para disseminação de conteúdos de Web Syndication (e.g. RSS feeds) segundo um princípio de disseminação de eventos, baseado no paradigma *peer-to-peer*. O modelo do sistema em estudo é apresentado a seguir.

Um conjunto de nós (*hosts*) com conectividade na Internet forma um sistema *peer-to-peer*. O conjunto de nós do sistema está sempre a variar porque existem nós que entram e nós que saem. O número total de nós presentes em cada momento pode variar, pois as saídas são independentes das entradas. Por hipótese, pretende-se que o algoritmo seja capaz de funcionar com um grande número de nós presentes em simultâneo. Por exemplo, entre 10.000 e 100.000. Em média cada nó que entra no sistema permanece um tempo razoável, por exemplo algumas horas, mas nada impede que esse tempo de permanência seja inferior ou superior a estes intervalos.

O sistema *peer-to-peer* em estudo é da filosofia One Hop Routing, isto é, cada nó tem uma tabela, ou base de dados, de encaminhamento, que contém o identificador e o endereço IP de todos os outros nós presentes no sistema. Assim, um nó pode enviar uma mensagem a qualquer outro nó visto que conhece o seu endereço IP. As diferentes instâncias das tabelas dos diferentes nós não são uma cópia exacta umas das outras e até podem con-

ter momentaneamente informação errada. Com efeito, quando um nó entra, ele não passa a ser conhecido instantaneamente por todos os outros, e quando um nó sai, isso não é repercutido instantaneamente nas tabelas de todos os outros.

Este aspecto da inconsistência das réplicas das bases de dados não é tratado nesta dissertação. O que se pretende é que um nó que entre no sistema seja conhecido o mais rapidamente possível por todos os outros. Quanto às saídas de nós, esta questão também não é tratada nesta dissertação. Assume-se que se o nó A tenta contactar o nó B e não consegue, A não sabe se B já saiu, ou se momentaneamente está inacessível. Podemos admitir que A marca B na sua tabela como estando inacessível no momento $T1$ e que periodicamente vai vendo se vale a pena manter B na sua tabela. Em resumo, o algoritmo tenta propagar rapidamente as entradas de nós, mas gere as saídas de forma descentralizada (cada nó suprime entradas da base de dados de forma independente) e quando tiver disponibilidade para isso. Isto é, cada nó vai fazendo alguma limpeza ("garbage collection") da sua tabela, quando tem tempo para isso.

Assim, existem N nós, cada um dos quais tem uma base de dados dos outros nós do sistema e pretende-se que essa base de dados reflicta, em cada momento, o conjunto de nós presentes (ou que pelo menos estiveram recentemente). Os identificadores dos nós têm um número elevado de bits (por exemplo, 128) e são gerados aleatoriamente (por hipótese sem colisões e com uma distribuição uniforme no espaço de todos os identificadores possíveis).

Segue-se uma descrição do algoritmo preliminar[4] de gestão da filiação e difusão *(broadcasting)* de interesses do sistema LiveFeeds.

3.2 Algoritmo de *broadcasting*

O universo de todos os identificadores está dividido em fatias. Cada fatia corresponde a uma zona contígua de identificadores. Por exemplo, se há S fatias, cada fatia corresponde a $(2^{128}/S)$ identificadores. Em cada fatia, o nó presente na fatia com o menor identificador designa-se por mestre ou coordenador da fatia, ou simplesmente coordenador.

[4] O algoritmo LiveFeeds encontra-se ainda em desenvolvimento, afinação e avaliação, pelo que neste trabalho utilizou-se uma versão preliminar do mesmo.

Os coordenadores assumem a responsabilidade de acumular as alterações que tiveram lugar na sua fatia e de periodicamente propagarem-nas de forma agrupada para todos os nós do sistema. Isto é, a propagação de alterações acumuladas correspondentes a uma fatia é sempre iniciada pelo seu coordenador. Esta opção tem a ver com a necessidade de dar hipóteses ao coordenador de comprimir mensagens e transmitir vários actualizações em simultâneo, para diminuir o impacto do desperdício ("overhead") de comunicação do algoritmo.

Quando um qualquer nó pretende difundir uma mensagem por todo o sistema, este divide o espaço dos identificadores em B blocos de igual tamanho, e envia uma mensagem para um nó de cada um desses blocos, escolhido aleatoriamente. O número B de blocos diz-se o "fanout factor". Assim, o emissor divide o espaço dos identificadores em B blocos de $(2^{128}/B)$ identificadores contíguos, escolhe um nó aleatório em cada um desses blocos e passa-lhe a mensagem com uma indicação do bloco (intervalo) por que o nó seguinte assume a responsabilidade de continuar a difusão. De forma recursiva, cada nó que recebe uma mensagem, guarda uma cópia e divide de novo o seu intervalo em B blocos (agora mais estreitos) e recomeça a difusão. Este processo continua até que todos os nós que recebem uma mensagem, recebem a responsabilidade de a continuar a difundir num bloco com um número de elementos entre 0 (terminação final) e B. O algoritmo, pela sua natureza, não introduz duplicados.

Um nó A que entra no sistema deve começar por obter um identificador e contactar um qualquer nó que já esteja no sistema, por exemplo J. A recebe de J uma cópia da base de dados e, quando esta estiver completa, transmite ao coordenador da sua fatia uma mensagem a dizer que entrou no sistema. Enquanto A não receber uma mensagem com uma actualização em que a sua entrada figura, ele mantém-se a receber e actualizações de J e periodicamente reenvia a notificação da sua entrada ao coordenador corrente (o antigo pode ter saído no entretanto).

O algoritmo simplificado que acabou de ser descrito é, no essencial, um algoritmo de difusão de mensagens para um conjunto de nós, através de árvores aleatórias de grau B para um sistema *peer-to-peer* em que todos os nós conhecem todos os outros nós.

Dada a natureza do estudo que se pretende realizar, no que se segue vamos admitir que não se perdem mensagens, que a filiação é estável durante a difusão de uma mensagem,

isto é, cada difusão é atómica no sentido em que não existe concorrência entre difusões e alterações da filiação e respectiva propagação.

3.3 Considerações sobre a relação do algoritmo com a latência e o objectivo da investigação

Se todos os nós estiverem à mesma distância na rede uns dos outros, o esboço de algoritmo apresentado parece ser especialmente adequado para distribuir a carga das difusões de forma aleatória pelos diferentes nós do sistema. Na verdade, cada mensagem de um coordenador é difundida por uma árvore aleatória, em que os nós têm grau B e, se as árvores forem equilibradas e o número de nós (N) for da forma: $N = \sum_{i=0}^{P} B^i$, em que P corresponde à profundidade da árvore, as mesmas terão profundidade $P = log_B\ N$ num sistema com N nós. No caso geral, a árvore pode não ser completa e mesmo não equilibrada devido a uma distribuição não completamente uniforme dos identificadores.

Numa rede real, as latências não são iguais e portanto podem-se escolher os nós sem ser aleatoriamente, mas de forma a privilegiar outros critérios. Por exemplo, escolher a árvore que assegure que cada nó recebe a mensagem por um caminho aproximadamente de menor custo, ou privilegiar qualquer outro critério baseado na distância na rede.

Para este efeito é necessário fazer em cada passo uma escolha dos B nós seguintes tendo em consideração a latência para comunicar com cada um deles. O ideal seria portanto que os nós tivessem acesso, de alguma forma, a uma versão aproximada da matriz dos custos de comunicação entre os nós presentes no sistema.

Existem muitas aproximações possíveis para a estimação dessa matriz. O objectivo é estudar diversas versões das mesmas e fazer um estudo comparado, do comportamento de diferentes versões do algoritmo acima descrito mas tomando em consideração aproximações diversas da matriz de custos de comunicação.

3.4 Apresentação dos algoritmos estudados: GNP e Vivaldi

Perante o problema em análise torna-se necessário encontrar uma forma escalável de deduzir qual a matriz de custos para um sistema com as características descritas anteriormente e com um número de nós considerável. De entre os diversos tipos de sistemas que poderiam fornecer uma aproximação de uma matriz de custos para os diversos nós, os sis-

temas de coordenadas virtuais são os que se adaptam de uma forma mais escalável ao problema, por estarem mais orientados ao paradigma *peer-to-peer*. Dos vários sistemas de localização virtual, que funcionam através da utilização de coordenadas virtuais, disponíveis e que poderiam ser considerados como alternativas a integrar no sistema LiveFeeds, foram escolhidos os sistemas GNP e Vivaldi, que apesar de fornecerem resultados com o mesmo formato, isto é, coordenadas num sistema de várias dimensões, pertencem a duas correntes diversas que importa comparar neste contexto. Teria interesse efectuar um estudo mais alargado, que permitisse a integração e estudo outros tipos de algoritmos e sistemas, diferentes dos que aqui se apresentam. Por exemplo, sistemas de coordenadas geográficas e sistemas de proximidade relativa. No entanto tal não foi realista, como já foi explicado, pelo que o estudo ficou reduzido à integração dos dois sistemas acima referidos.

De seguida apresenta-se uma síntese sobre estes dois sistemas de coordenadas virtuais e algumas considerações sobre o que é estudado a propósito da integração dos mesmos no sistema LiveFeeds.

3.4.1 GNP

O sistema GNP, apresentado no capítulo anterior, permite, no contexto do algoritmo LiveFeeds, ter um sistema de coordenadas virtuais baseadas no posicionamento relativo na rede de nós especiais, os chamados *landmarks*. O posicionamento das *landmarks* é baseado em medições de latências entre estas. Este posicionamento relativo, reduzido a coordenadas num espaço de várias dimensões, permite extrapolar as coordenadas virtuais dos nós do sistema e assim estimar uma matriz de custos entre os diversos nós.

3.4.2 Vivaldi

O Vivaldi, sistema que foi também apresentado no capítulo anterior, permite estimar o posicionamento relativo dos nós através de coordenadas virtuais que são calculadas através de medições de latências entre os diversos nós. À semelhança do GNP, as coordenadas virtuais reflectem o posicionamento relativo dos vários nós. No entanto, ao contrário do GNP, a aproximação tomada pelo sistema Vivaldi dispensa a utilização de *landmarks* e é uma aproximação mais dinâmica ao estar em permanente adaptação a possíveis

mudanças no comportamento da rede. Estas características tornam o Vivaldi mais flexível e mais orientado ao paradigma *peer-to-peer* quando comparado com o GNP.

Desta forma, pretende-se avaliar se existe benefício na utilização de sistemas de obtenção de posicionamento relativo ou de coordenadas virtuais no contexto do algoritmo Live-Feeds, se esse benefício é significativo e qual dos sistemas, Vivaldi ou GNP, serve melhor o desígnio descrito anteriormente. Para isso, foi feita uma análise comparativa entre os resultados apresentados com a utilização de ambos os sistemas e também dos resultados obtidos através da utilização da abordagem original empregue pelo algoritmo LiveFeeds, na qual, a escolha dos nós dentro das fatias, aos quais serão entregues as mensagens seguintes, não tem em conta qualquer matriz de custos, sendo os nós escolhidos de forma aleatória. De modo a poder estabelecer um modo de comparação entre as diversas abordagens foi estabelecida uma escala na qual se podem colocar os resultados obtidos através das várias abordagens estudadas. Estas opções de estudo estão patentes nos capítulos seguintes desta dissertação.

4. Ferramentas de teste e sua utilização no estudo realizado

Neste capítulo é feito um estudo sobre os métodos de avaliação de desempenho de algoritmos e sistemas para a Internet dependendo dos recursos disponíveis e da natureza do objecto de estudo e da experimentação. É ainda descrita a utilização específica de métodos e ferramentas no contexto do trabalho de experimentação prático, realizado ao longo desta dissertação.

A avaliação do desempenho de algoritmos e sistemas distribuídos é efectuada através de análises formais, geralmente restritas a casos simplificados, mas sobretudo através da experimentação empírica. Este argumento é tanto mais verdade no caso de sistemas de larga escala com funcionamento na Internet, por exemplo. Os métodos experimentais e as plataformas de experimentação baseiam-se em redes reais, em emulação de redes ou em simulação. Existem vários tipos de ferramentas que melhor servem cada forma específica de experimentação.

4.1 Experimentação com base em plataformas na rede global

A experimentação com base em plataformas na rede global é a que fornece resultados mais próximos dos que se verificariam numa situação real, por usar nós da rede global que por consequência se encontram em condições reais. Isto é, usam a pilha de protocolos de comunicação, estando ainda expostos às condições inerentes à rede, como sejam, o *jitter*, as filas de espera nos nós internos da rede, as latências, o chamado tráfego de *background*, taxas de perca de pacotes, entre outras. Estas condições que podem ser difíceis de emular ou de simular têm influência nos resultados obtidos a partir das aplicações de teste, tendo mais ou menos relevância consoante o objectivo do estudo.

No entanto, mesmo sendo possível fazer a instalação de aplicações de teste em diversos nós, existe normalmente um problema prático, que consiste em conseguir obter um número suficiente de máquinas ligadas à rede global, de modo a poder efectuar os testes necessários.

O exemplo mais importante de uma plataforma de experimentação sobre a rede global é o sistema *PlanetLab* [21] [22]. Este sistema constitui-se como uma plataforma composta por um conjunto de nós acessíveis na Internet. Estes nós encontram-se à disposição para a realização de experimentação, resolvendo assim o problema prático descrito anteriormente.

O sistema *PlanetLab* constitui-se como uma rede "overlay" global, cujas centenas de nós de diversos tipos se encontram nas mais diversas localizações na Internet, principalmente em instituições de ensino superior e de investigação. Esta plataforma permite testar as mais diversas classes de serviços, aplicações e algoritmos de rede através do seu mecanismo de virtualização que tem por base a utilização de "fatias" (*slices*) que podem ser vistas como um subconjunto de máquinas virtuais, possuindo um conjunto de recursos à sua disposição. Em cada "fatia" residem e são testadas aplicações independentemente das restantes "fatias", onde, por sua vez, estão a ser testadas outras aplicações de forma paralela. Estas características de paralelismo e virtualização constituem-se como aspectos distintivos e de grande relevância no sistema *PlanetLab*.

Para resolver o problema de conseguir obter um número significativo de máquinas ligadas à Internet, de modo a poder efectuar os testes necessários sobre as aplicações, serviços ou algoritmos, permitindo obter condições de teste semelhantes às que se encontrariam em situação real, o *PlanetLab* tem vindo a constituir-se como uma plataforma muito usada a nível de investigação nesta área por permitir obter resultados considerados mais próximos dos que se obteriam na Internet.

Os resultados obtidos não podem, no entanto, ser considerados completamente fiáveis ou iguais aos que se obteriam num sistema a operar na Internet visto que, os nós que pertencem ao sistema estão geralmente bem ligados, isto é, as suas condições de ligação à Internet são melhores em média que a generalidade dos nós que estão ligados à rede global. Isto deve-se ao facto de os nós que pertencem ao sistema *PlanetLab*, como foi referido anteriormente, serem cedidos por instituições de ensino superior e de investigação. Este facto pode fazer com que haja uma distorção associada aos resultados obtidos visto que em condições reais uma amostra aleatória de nós da Internet teria possivelmente condições de ligação à rede consideravelmente piores que as dos nós usados no sistema *PlanetLab*. Outra razão para os resultados obtidos não poderem ser considerados completamente fiáveis é o facto de o número de nós pertencentes ao sistema *PlanetLab*, apesar de sig-

nificativo, ser apenas uma ínfima fracção da totalidade de nós que se encontram a cada momento ligados à Internet. Mesmo tendo em conta estes argumentos é possível afirmar que o *PlanetLab* é o sistema que fornece possivelmente melhores aproximações aos resultados que se poderiam encontrar num sistema a operar na Internet.

4.2 Emulação de rede

A emulação de rede é efectuada através de uma infra-estrutura de rede localizada, composta por conjuntos ou *clusters*[5] de nós. Idealmente, o modo de interligação entre os nós, será de forma a emular o comportamento de uma rede global. Deste modo, as ligações entre os vários nós emulados possuem latências e taxas de perca de pacotes, que serão semelhantes às de uma rede real. Em geral, os emuladores usam topologias simplificadas de redes que são geradas através de geradores de topologias.

Por usarem *clusters* de nós os emuladores permitem testar o código das aplicações a testar, utilizando a pilha de protocolos real de rede existente nos sistemas de operação. Deste modo, os nós emulados estarão em condições aproximadas às dos nós de uma rede não emulada com excepção da topologia de rede.

Em seguida são apresentados dois exemplos considerados representativos de emuladores de rede.

O emulador de rede Dummynet [23] é um emulador de redes simples que funciona através da intercepção da comunicação entre a camada protocolar em análise e a camada subjacente, emulando assim a presença de uma rede real com filas de espera de pacotes finitas, largura de banda limitada, atrasos de comunicação e até mesmo perda de pacotes. Este emulador foi um dos primeiros a funcionar desta forma, isto é, como um verdadeiro emulador.

O sistema de emulação ModelNet [24] é um outro exemplo de um emulador de redes, no entanto, este apresenta-se como sendo mais escalável e tendo um ambiente de emulação de rede em larga escala mais abrangente em relação a sistemas simples, como por exemplo, o Dummynet. A arquitectura deste sistema consiste num conjunto de nós de periferia, nos quais correm as aplicações a testar, sendo que esses nós estão configurados para encaminhar todos os seus pacotes para um *cluster* no núcleo da rede. Esse *cluster* é res-

[5] Tradução para português: Agregados

ponsável por emular as características da topologia de rede especificada, como filas de espera de pacotes finitas, largura de banda limitada, atrasos de comunicação, congestionamento de rede, perda de pacotes, entre outros factores.

4.3 Simulação

Os simuladores são plataformas centralizadas que tentam modelar o comportamento da comunicação através dos canais físicos, protocolos e nós da rede, ou simplesmente o comportamento de sistemas específicos.

Os simuladores que contemplam na sua implementação um maior nível de detalhe da rede, são normalmente orientados aos eventos e conseguem apenas simular sistemas com algumas centenas de nós.

Outros simuladores existem que abstraem muitos detalhes de mais baixo nível, conseguindo simular sistemas de maior escala, na ordem dos milhares de nós.

Finalmente, existem simuladores especializados, cuja concepção está intimamente ligada ao ambiente e sistema que pretendem simular, baseando-se apenas em modelos simplificados de rede e dos sistemas, mas que permitem o teste de sistemas com centenas de milhares de nós.

4.3.1 Ns-2

O simulador Ns-2 [25] é um conhecido simulador de redes de baixo nível, orientado aos eventos, disponível em domínio público e orientado para a investigação. Este simulador suporta a simulação de TCP, encaminhamento de pacotes e protocolos de *multicast* nos mais variados tipos de rede.

4.3.2 OPNET

O simulador de redes OPNET [26],[27] é um produto comercial que possui um ambiente de desenvolvimento abrangente para especificação, simulação e análise de desempenho de redes. Este é composto por um conjunto de editores textuais e gráficos que permitem criar e editar topologias de rede, bem como os nós que a constituem e as suas características e ainda efectuar controlo de processos e dinâmica da rede. Possui ainda ferramentas para efectuar e depurar simulações bem como para definir, analisar e filtrar os dados e

resultados obtidos. Possui ainda uma interface gráfica animada onde é possível visualizar aspectos dinâmicos de rede.

4.4 Comparação e discussão das aproximações em estudo

De entre as três aproximações apresentadas: sistemas com nós espalhados pela Internet, emulação de rede e simulação, é possível verificar que todas têm a sua importância no contexto de sistemas específicos e dependendo dos resultados que se esperam obter. A aproximação que permite obter resultados mais próximos dos que se verificariam na Internet é o teste a partir de sistemas que possuem nós espalhados pela Internet, como o *PlanetLab*. No entanto, este tipo de solução nem sempre está disponível e nem sempre é aplicável. De modo a poder usar esta plataforma, uma entidade precisa de ser aceite na mesma. Para isso é necessário ter à disposição máquinas ligadas ao sistema, sendo que essas máquinas e a infra-estrutura de ligação devem cumprir um conjunto de pressupostos. Para ter à disposição um conjunto significativo para teste, isto é, um conjunto significativo de recursos, é necessário contribuir de forma significativa para o sistema, pois este baseia-se na partilha de recursos. Por outro lado é difícil realizar testes preliminares pois é necessário dispor já de uma versão executável do sistema e as ferramentas de análise dos resultados são pouco desenvolvidas, sendo principalmente constituídas por *logs* textuais. A recolha desses mesmos resultados também é uma tarefa difícil visto que os nós participantes se encontram distribuídos, assim como os dados recolhidos nestes.

Uma alternativa ao teste sobre sistemas reais é a utilização de emuladores de rede. Esta aproximação não é tão rigorosa devido ao facto de as topologias de rede artificiais usadas pelos emuladores serem apenas aproximações que tentam modelar a rede global. A emulação permite, no entanto, que com poucos recursos seja possível obter resultados que serão semelhantes aos que se obteriam na rede global.

Quanto à simulação, por ser normalmente centralizada, tenta modelar diversos aspectos da rede. Aspectos estes que já estão disponíveis à partida nas restantes aproximações. Quando o nível de detalhe da modelação dos aspectos de rede é muito elevada apenas é possível simular algumas centenas de nós, pelo que por vezes o detalhe de rede é desprezado. O que permite ter alguma flexibilidade não só no número de nós que é possível ter numa simulação mas também nos diversos aspectos do detalhe de rede, que podem ou não ser relevantes no que se pretende testar. Os simuladores especializados, por estarem

completamente ligados com o sistema a testar permitem abstrair certos detalhes da rede simulada subjacente. Estes detalhes poderão não ter uma relevância significativa para o funcionamento específico do sistema a ser testado, pelo que em certos casos podem ser ignorados. Ao abstrair detalhe, está-se a ganhar em desempenho de simulação, por isso é possível aumentar o número de nós presentes na mesma, mesmo em sistemas centralizados. A simulação permite ainda testar sistemas que estão em construção ou cuja implementação não esteja pronta para ser testada em ambientes que assentam na rede global ou ambientes de emulação, pois no caso destas duas últimas aproximações é necessário que o sistema ou algoritmo a testar esteja preparado para usar a pilha de protocolos de rede usuais. É possível ainda testar apenas alguns aspectos de um sistema, sem ter obrigatoriamente toda a lógica do sistema a funcionar no simulador. Estas características conferem bastante flexibilidade aos simuladores. Por outro lado, os simuladores são mais intrinsecamente ligados com o sistema ou algoritmo a testar, permitindo por isso uma recolha e análise mais fácil de resultados e observações.

Tanto os emuladores como os simuladores recorrem a topologias de rede sintéticas de modo a modelar ou aproximar a topologia da Internet num número reduzido de nós. Para isso, existem à disposição geradores de topologias de rede que normalmente são integrados nos sistemas de emulação ou simulação. A secção que se segue é dedicada aos geradores de topologias de rede, sendo descritas as suas características principais e descritos alguns dos mais representativos.

4.5 Geradores de topologias de rede

O objectivo dos geradores de topologias de rede é gerar topologias ou modelos de rede, que reproduzem num número reduzido de nós, as propriedades de uma rede com a dimensão da Internet.

Alguns dos geradores de topologias de rede disponíveis em domínio público são apenas integráveis com alguns dos simuladores e emuladores mais usados e de uso genérico. Mas normalmente os geradores de topologias de rede permitem "exportar" topologias geradas de acordo com alguns parâmetros da rede a modelar, sendo que o resultado dessa exportação poderá ser então usado de uma forma mais genérica por emuladores ou simuladores que possam usar o formato exportado. Essas topologias geradas podem então ser usadas por simuladores especializados ou feitos à medida.

Segue-se uma breve descrição de alguns geradores de topologias de rede conhecidos e de algumas das suas principais características.

4.5.1 GT-IMT

O gerador de topologias de rede GT-IMT [28] é um dos geradores de topologias de rede pioneiros. Este gerador de topologias de rede incorpora estruturas hierárquicas presentes na Internet, apesar de não reproduzir de forma exacta a distribuição observada entre os sistemas autónomos. Este gerador de topologias de rede é integrável com vários tipos de simuladores conhecidos.

4.5.2 Brite

O Brite [29] é um gerador de topologias de rede completo e integrável com alguns dos simuladores mais conhecidos. Foi desenvolvido tendo como principais preocupações, a flexibilidade, interoperabilidade, portabilidade e usabilidade. Sendo um desenvolvimento posterior ao GT-IMT, este gerador incorpora uma distribuição de grau dos nós que reproduz de forma mais aproximada a distribuição entre sistemas autónomos.

4.5.3 Orbis

O Orbis [30] é um gerador de topologias de rede simples, mais recente que os anteriores, que tenta modelar a estrutura hierárquica da Internet. A aproximação que é feita pelo Orbis é baseada em observações sobre as topologias de rede reais e a evolução das mesmas. De modo a obter modelos de dimensões diversas são usadas técnicas de mudança de escala em grafos numa tentativa de manter a coerência e as propriedades das topologias originais. Para gerar modelos próximos da rede real é tomada uma aproximação na qual se parte de um nível elevado até chegar gradualmente aos nós "folha". Isto é, começando pela construção das relações entre sistemas autónomos (AS[6]), passando pelas relações entre "routers" da rede (intra-AS), as ligações entre nós de fronteira (inter-AS), até à construção das ligações entre os "routers" e os diversos nós "folha".

Ao permitir obter modelos aproximados da realidade, o Orbis possibilita aos emuladores e simuladores que usam as topologias por si geradas, obter resultados mais aproximados aos que seriam obtidos na Internet.

[6] do inglês "Autonomous System".

4.6 Simulador e suas características específicas

Pelas razões anteriormente expostas é fácil reconhecer que a forma mais flexível de conduzir o estudo objecto desta dissertação é através de simulação. Dos simuladores disponíveis foi escolhido um desenvolvido no contexto do próprio projecto LiveFeeds.

O simulador usado na experimentação relativa à parte prática desta dissertação foi construído no contexto do projecto LiveFeeds e passou a ser também usado como uma ferramenta de apoio à aprendizagem, em disciplinas de redes e sistemas distribuídos da licenciatura e do mestrado em Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa. O algoritmo de disseminação do sistema LiveFeeds tem sido construído e testado usando esta plataforma de simulação. Nesta secção são descritas características deste simulador e do seu modelo de simulação de modo a poder, nas secções seguintes, discutir a implementação do algoritmo no contexto deste simulador e do seu modelo de simulação específico.

4.6.1 Descrição do modo básico de funcionamento do simulador (funcionalidades/framework)

O simulador usado é um simulador centralizado e orientado aos eventos, construído usando a linguagem de programação orientada a objectos Java [31]. O Simulador oferece uma interface genérica que permite simular diversos tipos de algoritmos e sistemas que podem ser integrados directamente com o simulador ao estender a sua classe principal, a classe abstracta “Simulation” (Figura 8) que constitui o ponto de início da simulação e como tal possui algumas características internas, como o tempo de simulação.

```
abstract public class Simulation implements Displayable {
   public static Gui Gui ;
   public static Random rg ;

   public static Network Network ;
   public static Scheduler Scheduler ;
   public static Simulation Simulation ;

   private static TimeWarpTask timeWarpTask = null;

   protected Simulation(double fps, EnumSet<DisplayFlags> flags) { … }
   public double currentTime() { … }
   protected void start() { … }
   protected void stop() { … }
   …
}
```

Figura 8 – Resumo da classe Simulation.

Um conceito central no simulador é o conceito de nó. Um nó pode ser visto como uma abstracção a muito alto nível de um computador que faz parte da rede que está a ser simulada. Este conceito de nó é representado no simulador pela classe "AbstractNode" (Figura 9), que sendo uma classe abstracta deve ser estendida por classes que implementem um ou mais tipos de nós com características e comportamentos específicos do sistema a simular. A classe "AbstractNode" implementa a interface "MessageHandler" (Figura 10). Esta interface contém os métodos ligados ao tratamento dos eventos de recepção de mensagens. Uma mensagem é representada no simulador pela classe abstracta "Message" (Figura 11). Para uma mensagem ser trocada entre dois nós da rede é invocado um método de entrega de mensagem a um endereço de rede. De modo a ser endereçável, um nó possui um endereço de rede na rede simulada. Este endereço pertence à classe "NetAddress", a qual será melhor descrita na secção dedicada à rede no contexto do simulador (4.6.5). Para além dos métodos de tratamento de recepção de mensagens, a classe "AbstractNode" contém ainda métodos para enviar mensagens através de canais com semântica próxima dos protocolos UDP e TCP.

```
abstract public class AbstractNode implements MessageHandler {
   public EndPoint endpoint;
   public NetAddress address;

   protected AbstractNode() { … }
   protected boolean udpSend(EndPoint dst, Message m) { … }
   protected boolean udpSend( NetAddress dst, Message m ) { … }
   protected TcpChannel tcpSend( EndPoint dst, Message m ) { … }
   protected TcpChannel tcpSend( NetAddress dst, Message m ) { … }

   public void onReceive( EndPoint src, Message m ) { … }
   public void onReceive( TcpChannel ch, Message m ) { … }
   …
}
```

Figura 9 – Resumo da classe AbstractNode.

```
public interface MessageHandler {
   public void onSendFailure( EndPoint dst, Message m ) ;
   public void onReceive( EndPoint src, Message m ) ;
   public void onReceive( TcpChannel chn, Message m ) ;
}
```

Figura 10 – Resumo da interface MessageHandler.

```
abstract public class Message implements EncodedMessage {
   protected Message() { … }

   public int length() { … }
   public void deliverTo(EndPoint src, MessageHandler handler) { … }
   public void deliverTo(TcpChannel ch, MessageHandler handler) { … }
   …
}
```

Figura 11 – Resumo da classe Message.

4.6.2 Configurações globais de simulação e rede

Antes de iniciar uma simulação devem ser configurados os vários parâmetros globais de simulação. Estes parâmetros podem ser, por exemplo, as sementes aleatórias. Neste caso, existe uma semente aleatória de simulação e uma semente aleatória de rede. Estas sementes aleatórias permitem que uma dada simulação seja repetida de forma exacta, bastando para isso usar as mesmas sementes aleatórias em cada caso. Outros parâmetros podem ser definidos que têm que ver com a rede, como por exemplo, os tamanhos dos cabeçalhos TCP ou UDP. Existem ainda parâmetros mais específicos relacionados com o tipo de rede que esteja a ser usado. A comunicação entre os nós é simulada através de uma abstracção de canais TCP ou de troca de pacotes UDP, em que poderá ocorrer perda de mensagens.

4.6.3 Tempo de simulação e eventos

Como já foi referido, o simulador é orientado aos eventos. Existe uma variável de simulação que é precisamente o tempo de simulação desde o seu início que faz parte de classe abstracta "Simulation", já referida. Esta variável é controlada pelos eventos que decorrem no simulador. Estes eventos podem ser, por exemplo, as trocas de mensagens UDP. Neste caso, quando uma mensagem é enviada por um determinado nó, a recepção da mensagem só será tratada pelo nó receptor no momento exacto (tempo simulado) em que a mensagem chega a esse receptor. A recepção de uma mensagem é considerada como uma tarefa que vai decorrer no momento futuro $T + L$, em que T é o tempo de simulação em que a mensagem é enviada e L corresponde à latência (somada do *jitter*) entre o nó emissor e o nó receptor. Quando uma mensagem é enviada, é imediatamente calculado o tempo que esta demorará a chegar ao nó receptor. Esta tarefa é então colocada num fila de tarefas a executar, ordenada pelo tempo de execução. Ao ser retirada a próxima tarefa da fila, o tempo de simulação é ajustado para o tempo definido na tarefa. Nesse momento é disparado o evento que faz com que a mensagem seja tratada pelo receptor.

4.6.4 Interface gráfica

O simulador é dotado de uma interface gráfica (Figura 12) que pode ser usada para demonstrar de forma gráfica alguns dos aspectos de simulação. No simulador, virtualmente todos os elementos (nós, mensagens, etc.) possuem uma representação gráfica que pode ser usada de várias formas. Esta ilustração bastante útil em termos pedagógicos, em situações em que a ilustração e visualização dinâmica de aspectos dos algoritmos em estudo facilita o entendimento dos mesmos. No entanto, para a problemática em estudo nesta dissertação, a interface gráfica não possui uma grande relevância devido ao facto de os dados a serem medidos e estudados não serem facilmente visualizáveis, no entanto, esta pode ser usada a nível estatístico para mostrar e obter gráficos.

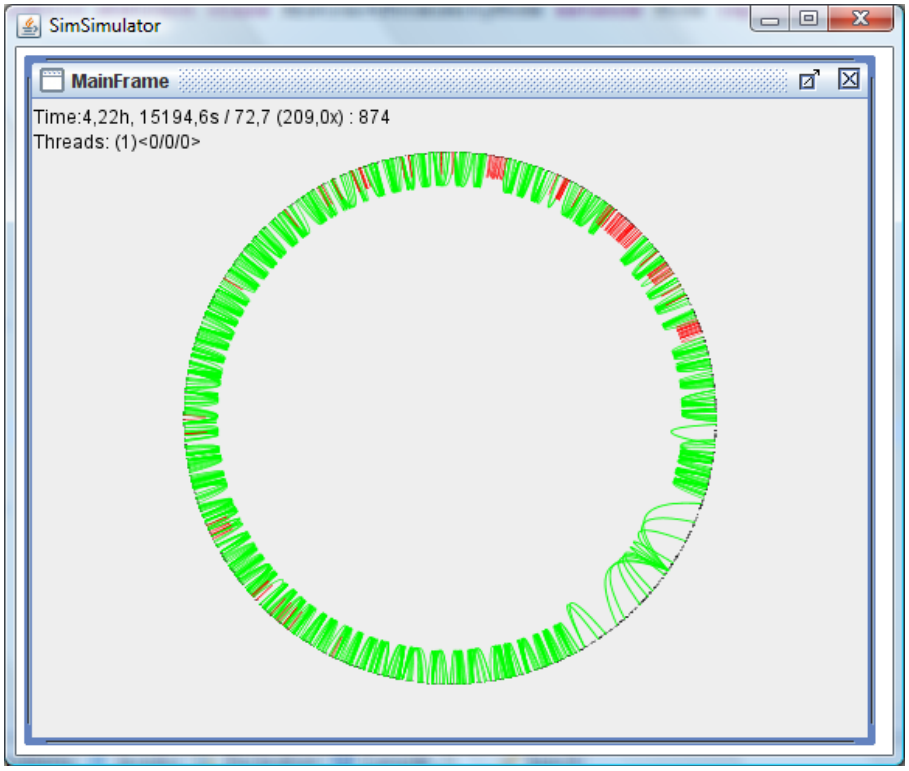


Figura 12 – Interface gráfica do simulador.

4.6.5 A Rede

Uma rede é representada no simulador pela classe abstracta "Network" (Figura 13). Esta classe tem como principal característica o conjunto de endereços que são conhecidos na rede. Estes endereços pertencem à classe "NetAddress" (Figura 14), também uma classe abstracta. A classe "Network" permite a criação de um endereço pelos nós, que assim passam a pertencer à rede. Existe ainda um método que permite obter todos os endereços pertencentes à rede. De modo a poder usar tipos de rede específicos, isto é, com comportamentos e modelos específicos para o fim que se queira testar, é necessário estender a classe "Network" e a classe "NetAddress". A classe "NetAddress" possui um método que permite obter a latência entre dois endereços de rede. Como seria de esperar este método assume uma grande importância na parte prática desta dissertação, devido aos métodos que tiram partido da latência entre os nós para estimar o posicionamento relativo dos mesmos através de coordenadas virtuais. Dependendo do modelo de rede implementado assim poderá ser a latência entre dois endereços. Durante a experimentação prática desta dissertação foram usados dois modelos diferentes de rede que são descritos nos parágrafos seguintes.

```
abstract public class Network implements Displayable {
   final double Jitter = 0;

   abstract public void setRandomSeed( long seed );
   abstract public Network init();

   abstract public Set<NetAddress> addresses();
   abstract public NetAddress createAddress(MessageHandler handler);
   …
}
```

Figura 13 – Resumo da classe Network.

```
abstract public class NetAddress implements Displayable {
   public EndPoint endpoint ;

   protected NetAddress(MessageHandler defaultHandler) { … }

   public double latency(NetAddress destination) { … }
   …
}
```

Figura 14 – Resumo da classe NetAddress.

O tipo de rede básico que existe definido à partida no simulador é o que replica uma rede Euclidiana a duas dimensões. Neste caso, a rede está definida de forma que cada nó possui ligação directa a todos os outros nós da rede e em que os nós se encontram distribuídos aleatoriamente num plano. A latência entre os nós é proporcional à distância em linha recta no plano. A colocação aleatória dos nós no plano é baseada na semente aleatória de rede que é definida nas configurações globais. Para a rede Euclidiana é possível definir alguns parâmetros simples de simulação, como sejam, a distância mínima entre nós no plano e outros parâmetros relacionados com a representação gráfica. Este tipo de rede muito básico, não segue um modelo que se aproxime minimamente do que se encontra na Internet, no entanto, serve para poder ter uma aproximação inicial que poderá indicar desde logo resultados tendenciais. Este tipo de rede é representado pela classe “EuclideanNetwork” (Figura 15).

```
public class EuclideanNetwork extends Network implements Displayable {
   private Random random ;
   private double costFactor;
   private double nodeRadius ;
   private boolean displayLabels ;
   private double minNodeDistance;
   private boolean toggleNodeShape;
   private double squareSideLength ;
   private final WeakHashMap<EuclideanAddress, Integer> nodes =
      new WeakHashMap<EuclideanAddress, Integer>();

   public Network init() { … }
   public void setRandomSeed( long seed ) { … }
   public NetAddress createAddress( MessageHandler handler) { … }
   public NetAddress replaceAddress( NetAddress other) { … }
   public Set<NetAddress> addresses() { … }

   class EuclideanAddress extends NetAddress {
      EuclideanAddress(MessageHandler handler, XY pos) { … }

      public double latency(NetAddress other) { … }
      public NetAddress replace() { … }
   }
   …
}
```

Figura 15 – Resumo da classe EuclideanNetwork.

Além da rede Euclidiana a duas dimensões que acabou de ser descrita, é possível definir outros tipos de rede no simulador. Um tipo de rede que foi integrado no simulador, já durante a fase de experimentação desta dissertação, foi o tipo de rede que é possível gerar a partir do gerador de topologias de rede Orbis, descrito anteriormente. Tal como para a rede Euclidiana, é possível definir alguns parâmetros de simulação específicos da rede Orbis. Um dos parâmetros que é possível definir é um factor de latência do "local loop". Outro parâmetro que é possível definir é número de classes de nós de "local loop". A latência de "local loop" de um nó será o produto do factor de latência pela classe a que o nó pertence. É possível ainda definir como parâmetro a latência correspondente às liga-ções no núcleo da rede. Para criar uma topologia de rede pelo Orbis é ainda definido o número de nós do núcleo da rede a ser gerada. A latência entre dois nós será então a soma das latências de "local loop" de cada um dos nós somada com as latências entre nós do núcleo de rede pertencentes ao caminho entre os dois nós. Com a utilização de topologias de rede geradas pelo gerador Orbis obtém-se um aumento em termos de qualidade do modelo de rede que não é comparável com a simplicidade da rede euclidiana de duas dimensões. As topologias geradas por este gerador de topologias são aproximadas em termos de estrutura à topologia da Internet. Deste modo é possível obter resultados mais

aproximados dos que se podem obter na Internet. O tipo de rede Orbis é representado pela classe "OrbisNetwork" (Figura 16).

```
public class OrbisNetwork extends Network implements Displayable {
   private String filename;
   private int localLoopClasses;
   private double corePerHopLatency;
   private double localLoopPerClassLatencyFactor;
   private Random random ;
   private final WeakHashMap<OrbisAddress, Integer> nodes =
      new WeakHashMap<OrbisAddress, Integer>();

   public Network init() { … }
   public void setRandomSeed( long seed ) { … }
   public NetAddress createAddress( MessageHandler handler) { … }
   public NetAddress replaceAddress( NetAddress other) { … }

   class OrbisAddress extends NetAddress {
      int stubRouter ;
      double localLoopLatency;

      OrbisAddress( MessageHandler handler) { … }
      OrbisAddress( OrbisAddress other) { … }

      public double latency( OrbisAddress other) {
         double ll_lat =
            this.localLoopLatency + other.localLoopLatency;
         double cn_lat =
            hopCount[this.stubRouter][ other.stubRouter]
            * corePerHopLatency;
         return cn_lat + ll_lat ;
      }
      public double latency( NetAddress other) {
         return this.latency((OrbisAddress) other ) ;
      }
      public NetAddress replace() { … }
      public Set<NetAddress> addresses() { … }
   }
   …
}
```

Figura 16 – Resumo da classe OrbisNetwork.

Independentemente do tipo de rede que é usada, é possível ainda definir um valor máximo de *Jitter*. Esse valor máximo é então usado para gerar um valor de *Jitter* aleatório artificial entre zero e esse mesmo valor máximo definido. Este factor de *Jitter* é o único elemento de simulação, em termos de rede, que se pode considerar um agente de instabilidade. Outros factores que estão presentes nas redes reais, como as filas de espera de pacotes, o tráfego de *background*, entre outros não são modelados pelo simulador.

4.7 Caracterização do modelo de simulação usado

A experimentação prática com base na Internet é sempre necessária de modo a tirar conclusões mais aproximadas da realidade sobre qualquer sistema que opere de forma distribuída na rede, sendo de facto o único teste real, isto é, a única forma de confirmar aquilo que apenas se pode supor em ambientes de emulação, simulação ou análise simples. No entanto, mesmo a experimentação com base na Internet global pode levar a conclusões que podem não se verificar na realidade, principalmente devido ao carácter de grande dinamismo da Internet, que pode levar a que o que se verifica no presente possa não ser verificado no futuro. Ao efectuar análise e experimentação sobre um modelo simplificado da Internet é possível obter um entendimento dos elementos básicos que podem influenciar em grande medida resultados e conclusões. No entanto, este estudo com base num modelo simplificado contém alguns riscos implícitos. Ao simplificar ou diminuir no modelo alguns aspectos da Internet, existe a possibilidade de os resultados que possam decorrer de estudos efectuados com base nesse modelo sejam inúteis. A simulação pode ser considerada complementar à análise fornecendo meios para testar cenários complexos que seriam difíceis ou impossíveis de analisar. A simulação permite ainda o desenvolvimento de intuição acerca do objecto de estudo. Em termos simulação, é cada vez mais importante a questão da escala. Tornam-se assim importantes questões como a topologia de rede, a forma como o tráfego é gerado e as múltiplas camadas protocolares, pelo que é importante ter simuladores capazes de modelar cenários que tenham em conta as variações destes parâmetros e cenários possíveis. Ao apresentar resultados sobre experimentação baseada em simulação é sempre difícil demonstrar que um determinado resultado não seria completamente diferente se apenas fosse efectuada uma variação de um parâmetro, no entanto, quando o objecto de estudo é algo tão complexo, heterogéneo e dinâmico [32] como uma rede global, seria impossível prever todas as variações possíveis, o que faz com que os resultados obtidos em simulação tenham obrigatoriamente de estar ligados apenas ao contexto do modelo de simulação usado, fornecendo apenas indicações.

Quanto ao modelo de simulação implementado pelo simulador que foi usado no contexto desta dissertação, é um modelo bastante simplificado. Este modelo funciona em níveis de abstracção elevados que vou passar a explicitar. Existe uma abstracção ao nível da camada de transporte de rede, nomeadamente o estabelecimento de canais TCP e envio de pacotes UDP, pelo que todas as camadas protocolares de mais baixo nível são ignoradas, existindo apenas uma noção de endereço em cada nó, que poderá ser vista como uma abstracção muito reduzida do nível IP. No sistema LiveFeeds as trocas de mensagens efec-

tuam-se através de pacotes UDP, no simulador é usada a abstracção de envio de pacotes UDP anteriormente referida, no entanto, não existe perda de pacotes neste contexto, pois esse problema escapa ao âmbito deste estudo. Esta abstracção dos protocolos e a não existência de perda de pacotes podem ser considerados como factores não determinantes para o âmbito deste estudo, no entanto, a perda de pacotes é um problema que não pode ser ignorado no caso do algoritmo de *broadcast* do sistema LiveFeeds.

O simulador não utiliza nenhum gerador de tráfego específico, o tráfego que se verifica é apenas resultante do envio de pacotes pelos nós, pelo que não existe o chamado tráfego de *background*, não existindo também modelação de filas de espera nos nós do núcleo da rede. No caso da versão usada do sistema LiveFeeds, cada nó difunde mensagens aquando da recepção de uma mensagem proveniente de outro nó, de modo a continuar a difusão, neste caso, no máximo tantas mensagens quanto o *fanout factor*. Para iniciar uma difusão, cada nó envia uma nova mensagem num intervalo fixo de tempo. Portanto, não existe neste caso uma modelação de quaisquer padrões de actividade de rede, isto é, de uma forma geral, a carga sobre a rede simulada é sempre pouco variável, não sujeita, por exemplo, a picos de tráfego como os que se verificam normalmente na Internet em períodos de maior e menor congestionamento na rede, regulados principalmente pelos ciclos de actividade humana. Para os casos referidos neste parágrafo, a sua não modelação confere à simulação um carácter pouco ou nada dinâmico, o que vai contra o carácter de grande dinamismo da Internet global que como se sabe, está sujeita a todos estes parâmetros.

Para o que se pretende estudar, será de prever que um algoritmo mais dinâmico como o Vivaldi seria menos prejudicado pelas situações descritas, em relação por exemplo, ao algoritmo GNP, que possui um carácter menos adaptativo. Uma das principais características do algoritmo Vivaldi é a constante medição de latência efectuadas entre os diversos nós, adaptando as coordenadas virtuais às mudanças verificadas. Num cenário em que a as características da rede se aproximem das condições que se verificam na Internet, em que existem todos os factores de dinamismo e instabilidade descritos, é previsível que o algoritmo Vivaldi seja capaz de manter as suas coordenadas consistentes com o estado da rede em constante mudança. A adaptação ao carácter dinâmico de rede foi um dos principais problemas que foram tidos em conta na construção deste algoritmo. Pelo contrário, o algoritmo GNP é mais estático, não existindo nenhum tipo de definição no algoritmo original que permita a sua adaptação. Uma versão mais dinâmica do algoritmo GNP, na qual

os *landmarks* e os restantes nós efectuassem medições de latência periodicamente adaptando as suas coordenadas virtuais, teria provavelmente um melhor desempenho face às características dinâmicas da Internet, no entanto, este tipo de adaptação no algoritmo GNP poderia ter outras consequências visto que não foi prevista no algoritmo original, nomeadamente um aumento de tráfego, principalmente em relação aos *landmarks*, que poderiam sofrer de uma sobrecarga de medições de latência, o que podia por sua vez causar medições de latência com valores superiores e assim distorções nas coordenadas virtuais.

Em termos de rede, é modelada a latência entre os diversos nós, dependendo do tipo de rede simulada. Neste caso, a utilização do gerador de topologias de rede Orbis, permite a obtenção de modelos aproximados aos que seriam de esperar na rede global. Esta modelação de latência pode ser complementada com a introdução de *Jitter*. Algo que não é modelado é o tempo de processamento das mensagens nos nós, no entanto, no contexto deste estudo tal pode ser considerado desprezável, pois o foco prende-se com a latência.

Tendo em conta o que foi até agora referido pode-se considerar que o simulador é limitado em alguns aspectos, no entanto, nem todos esses aspectos têm demasiada relevância para o contexto do estudo efectuado. A abstracção de alguns aspectos de mais baixo nível de rede, mas também de aspectos globais da Internet permite, no entanto, que em termos de escala, seja possível ter simulações com um número considerável de nós, na ordem das dezenas de milhar, sem ser necessário recorrer a máquinas com uma grande capacidade de processamento e/ou memória. Este aspecto é importante, na medida em que se pretende que o sistema LiveFeeds possa em teoria funcionar com dezenas ou centenas de milhar de nós, pelo que a escala de simulação é adequada.

Na tabela que se segue (Tabela 1) é apresentada uma síntese sobre a forma como os factores do modelo de simulação referidos acima influenciam o estudo realizado nesta dissertação.

Factores	Tipo de implicação
Abstracção dos protocolos TCP/UDP	Positivo
Não existe perda de pacotes	Desprezável (neste contexto)
Não existe tráfego de background	Negativo
Não existem filas de espera	Negativo
Não existem padrões de actividade de rede nem picos de tráfego	Negativo
Não é usado *jitter*	Desprezável
Topologia de rede Orbis	Positivo
Escalável devido à abstracção elevada	Positivo

Tabela 1 – Implicações do modelo de simulação.

4.8 O algoritmo LiveFeeds no contexto do simulador

No contexto do simulador, foi usada uma versão simplificada do sistema LiveFeeds que não possui gestão de filiação. Nesta versão todos os nós fazem inicialmente parte do sistema, não havendo gestão de entradas e saídas, pelo que os nós se encontram de forma permanente no sistema. A esta versão simplificada do sistema LiveFeeds foi dada a designação de "Tempest1". Na mesma, a classe "Simulation" é estendida pela classe principal do "Tempest1", a classe "Main" (Figura 17), o ponto de entrada da aplicação. Nesta classe estão definidas algumas constantes específicas do sistema LiveFeeds e de simulação, como o *fanout factor* e o número total de nós. Durante a fase inicial da execução do "Tempest1" todos os nós do sistema são criados e inseridos numa estrutura que os irá conter, sendo posteriormente iniciada a difusão de mensagens. Em intervalos regulares de tempo é escolhido um nó de forma aleatória para efectuar o envio de uma mensagem que será difundida por todos os nós segundo o algoritmo de difusão do sistema LiveFeeds descrito anteriormente.

```
public class Main extends Simulation implements Displayable {
   public static final int BROADCAST_MAX_FANOUT = 10;
   public static final int TOTAL_NODES = 10000;

   protected Main() { … }
   public Main init() { … }
   public static void main( String[] args ) throws Exception { … }
   …
}
```

Figura 17 – Resumo da classe Main do tempest1.

A implementação de um nó neste sistema é feita na classe "Node" (Figura 18) que estende a classe "AbstractNode" do simulador e implementa uma variante da interface "MessageHandler" que não possui perda de mensagens. O tipo de mensagem a ser difundida pelos nós do sistema é um objecto de uma classe que estende a classe abstracta "Message" que é designada por "BroadcastMessage" e que vai implementar o método de entrega de mensagens a nós que por sua vez vai invocar o método de tratamento de mensagem recebida no nó de destino, o que corresponde ao evento de recepção de mensagem. Este evento será tratado na classe "Node" onde o nó tratará de enviar a mensagem para os nós seguintes através do envio de mensagem UDP que é facultado pelo simulador. Cada nó deste sistema possui uma chave única no sistema que serve de identificador LiveFeeds. Ao receber uma mensagem, é extraído o intervalo de nós corrente que faz parte da mensagem, o intervalo recebido é então partido em sub intervalos de identificadores segundo o *fanout factor* sendo escolhido aleatoriamente um nó de cada intervalo ao qual será enviada a mensagem, continuando assim a difusão, até a mensagem chegar a todos os nós. Um intervalo de nós é representado pela classe "Range", esta classe permite a divisão em sub intervalos de nós que pertencerão à mesma classe. Esta classe possui ainda o método de escolha aleatória de um nó pertencente ao intervalo.

```
public class Node extends AbstractNode
    implements Comparable<Node>, AppMessageHandler {

    public long key ;

    public Node() { … }
    public void broadcast( Object o ) { … }
    public int compareTo(Node other) { … }
    public void start() { … }
    public void onReceive(EndPoint src, BroadcastMessage msg) { … }
    public void onReceive(EndPoint src, DeliverPayload m) { … }
    …
}
```

Figura 18 – Resumo da classe Node.

4.9 Os algoritmos estudados e a sua integração no simulador

De forma a poder integrar os diferentes algoritmos na plataforma de simulação usada foi necessário efectuar adaptações, estendendo a lógica do algoritmo simplificado do LiveFeeds ("Tempest1") que foi usado e adicionando funcionalidades que não existiam, nomeadamente a nível suporte de medições para obtenção de resultados. Nesta secção são

descritas essas construções e também o trabalho de adaptação de cada algoritmo em particular para operar neste contexto.

De modo a separar a implementação do algoritmo LiveFeeds ("Tempest1") da implementação que permite integrar os diferentes algoritmos que facultam a obtenção da noção de proximidade relativa entre os nós, foi primeiramente criada uma extensão da classe "Main" do "Tempest1" num novo pacote Java. Nesse pacote foram efectuados os desenvolvimentos específicos relacionados com a integração dos algoritmos de proximidade relativa.

Foi tomada a opção de estender a classe "Node" do "Tempest1" por uma classe abstracta que foi designada por "AbstractProximityNode" (Figura 19). Esta classe representa um tipo de nó que possui uma noção de proximidade relativa através de coordenadas virtuais cujo objecto é fornecer uma base comum às implementações específicas dos algoritmos em estudo. A classe "AbstractProximityNode" possui métodos para aplicar coordenadas ao nó e obter as mesmas, bem como um método para obter a distância entre nós em termos de coordenadas. Para facilitar a comparação de nós em termos de coordenadas a classe implementa a interface "Comparator" que faz parte da interface de programação do Java e que possui um método para comparar dois objectos da mesma classe, neste caso objectos que estendam a classe "AbstractProximityNode". A escolha de nós em função de proximidade vai então tirar partido desta lógica de comparação. A classe "Range" do "Tempest1", já referida anteriormente, que representa um intervalo de nós, possui um método que permite obter o nó mais próximo de entre os nós do intervalo segundo a comparação que seja efectuada no nó, sendo este o método usado durante a escolha dos nós do intervalo para os quais vai ser difundida a mensagem.

Para além das funcionalidades e métodos já referidos, a classe "AbstractProximityNode" possui ainda um membro que é um contador de mensagens enviadas, incrementado de cada vez que uma mensagem é enviada por um nó que estenda esta classe. A classe possui ainda um método que permite aceder ao valor deste contador. Este contador vai ser usado de forma a comparar a quantidade de mensagens enviadas pelos nós, para verificar uma questão que é levantada em 5.2 e que se prende com a carga a que cada nó estará sujeito devido ao uso de sistemas de coordenadas virtuais.

```
public abstract class AbstractProximityNode extends Node implements
Comparator<AbstractProximityNode> {

   protected double[] coordinates;
   protected int sentMessagesCount;
   public static final transient boolean emulateNode = true;

   public AbstractProximityNode() { … }

   public int getSentMessageCount() { return sentMessagesCount; }
   public double[] getCoordinates() { return coordinates; }
   public void setCoordinates(double[] coords) { … }
   public long key() { return key; }
   public double distanceTo(AbstractProximityNode other) { … }
   public void start() { … }
   public boolean isEmulateNode() { return emulateNode; }
   public void onReceive(EndPoint src, BroadcastMessage msg) { … }
   public void broadcast(Object o) { … }
   public void onReceive(EndPoint src, DeliverPayload m) {
   public int compare(AbstractProximityNode a,
      AbstractProximityNode b) {
      double dA = this.distanceTo(a);
      double dB = this.distanceTo(b);
      return (dA == dB)? (a.key < b.key ? -1 : 1): (dA < dB ? -1 : 1);
   }
   …
}
```

Figura 19 – Resumo da classe AbstractProximityNode.

Em seguida é feita uma descrição pormenorizada do que foi feito de modo a integrar os algoritmos do GNP e do Vivaldi.

4.9.1 Integração do algoritmo do GNP

Para poder integrar o algoritmo GNP no contexto da versão usada do sistema LiveFeeds foi feita uma pesquisa com o objectivo de encontrar em domínio público uma versão funcional do GNP que pudesse ser integrada neste sistema de uma forma facilitada. Foi encontrada uma versão do sistema GNP realizada pela Universidade da Virginia (Estados Unidos da América), construída em Java e que faz parte de um sistema existente designado de HyperCast [33], um sistema de investigação usado no desenvolvimento de protocolos e aplicações para o nível aplicacional em redes *overlay*. Esta versão consiste numa classe "Landmark" (Figura 20) cujo objectivo é modelar o sistema de *landmarks* que caracteriza o GNP. A classe "Landmark" possui um construtor que aceita como parâmetro a matriz de distâncias em termos de latência entre os nós que são usados como *landmarks* do algoritmo GNP. Durante a execução deste construtor são calculadas as coordenadas de cada *landmark* através da matriz de distâncias dada, o que é um processo moroso, poden-

do demorar alguns minutos no caso se serem criadas dezenas de *landmarks*. De modo a calcular as coordenadas de um determinado nó, segundo o algoritmo GNP, é usado um método da classe "Landmark" que recebe um vector de latências entre esse mesmo nó e cada um dos *landmarks* do sistema GNP.

```
/* Copyright (c) 1999-2005, The Rector and Board of Visitors of the
   University of Virginia.
…
public class Landmark {
   int numLandmarks;
   int dimensions;
   float[][] coordinates;
   float[][] lmDelays;
   boolean randomInit = true;

   public Landmark(float[][] delay, int dim) { … }

   public float[] fitTargetData(float[] tDelays,
      Boolean[] index )
   { … }
   …
}
```

Figura 20 – Resumo da classe Landmark.

A classe "Landmark" foi encapsulada de modo a poder ser usada no contexto em que foi inserida, assim, foi construída uma classe "GNPLandmarks" (Figura 21) que serve de interface entre a classe "Landmark" e o sistema no qual foi inserida. Essa classe contém um construtor que recebe como parâmetro o número de *landmarks* a usar para o GNP, sendo de seguida criados os nós que vão servir de *landmarks* para o algoritmo GNP. Estes nós não fazem parte do conjunto de nós do sistema "LiveFeeds", no entanto, são criados na mesma rede de modo a pertencerem ao mesmo espaço e assim fornecerem resultados válidos em termos de coordenadas. Depois de serem criados os nós que vão servir de *landmarks* é calculada a matriz de distâncias que vai ser passada ao construtor da classe "Landmark". O objecto da classe "Landmark" então criado estará a partir desse momento disponível na classe "GNPLandmarks" como membro da classe. Esse objecto será usado para calcular as coordenadas individuais de cada nó do sistema LiveFeeds, sendo para isso usado o método "GetCoordinates" que recebe um endereço ("NetAddress") do simulador e devolve as suas coordenadas segundo o GNP. Para calcular essas coordenadas é construído o vector de distâncias entre o endereço para o qual se querem calcular as coordenadas e cada um dos *landmarks* GNP, sendo esse vector passado como parâmetro ao método que efectua o cálculo das coordenadas na classe "Landmark".

```
public class GNPLandmarks {
   ArrayList<NetAddress> landmarks;
   float[][] delayMap;
   Landmark gnpAlgorithm;

   public GNPLandmarks(int numLandmarks) { … }
   public double[] getCoordinates(NetAddress address) { … }
}
```

Figura 21 – Resumo da classe GNPLandmarks.

Foi criada uma classe específica de nó para o algoritmo GNP a que se designou de "GNPNode" (Figura 22). Esta classe estende a classe abstracta "AbstractProximityNode" descrita anteriormente, tendo como características específicas os dois construtores: um construtor por defeito (sem argumentos) e um construtor que aceita como único argumento o objecto da classe "GNPLandmarks" que é usado para efectuar o cálculo das coordenadas para o nó. A razão de ser da existência destes dois construtores prende-se com o facto de o cálculo de coordenadas GNP para todos os nós do sistema poder ser um processo moroso. Devido a este facto, foi criada uma forma de guardar as coordenadas dos nós GNP calculadas para ficheiro. Assim, as coordenadas GNP apenas precisam de ser calculadas uma única vez através do processo moroso do algoritmo GNP, nas simulações seguintes, se o ficheiro que guarda as coordenadas dos nós já existir, será usado o construtor por defeito (sem argumentos) sendo atribuídas a posteriori ao nó GNP as coordenadas que constarem do ficheiro. Se o ficheiro que contém as coordenadas não existir é seguido o processo normal de cálculo de coordenadas para os nós usando o construtor que aceita como argumento o objecto da classe "GNPLandmarks". As coordenadas GNP são guardadas no ficheiro de coordenadas à medida que são calculadas.

```
public class GNPNode extends AbstractProximityNode {
   public GNPNode() { super(); }

   public GNPNode( GNPLandmarks gnp) {
      super();
      coordinates = gnp.getCoordinates(this.address);
   }
}
```

Figura 22 – Resumo da classe GNPNode.

4.9.2 Integração do algoritmo do Vivadi

Tal como para o caso do algoritmo GNP que acabou de ser descrito, no caso do algoritmo Vivaldi, foi feita uma pesquisa com o objectivo de encontrar em domínio público uma

versão funcional segundo os aspectos essenciais do Vivaldi que pudesse ser integrada no sistema em estudo de uma forma simplificada. Foram encontradas várias versões do algoritmo Vivaldi, no entanto, nem todas seriam integráveis de uma forma fácil no sistema em estudo. Foi efectuado um estudo no sentido de determinar qual dessas versões do Vivaldi melhor se adaptaria ao sistema em estudo. Foi encontrada uma versão que faz parte do código fonte de um sistema *peer-to-peer* (*Azureus*, aplicação cliente para redes *peer-to-peer* que usam o protocolo *BitTorrent*) mas que apesar de completa, se encontra demasiado ligada ao referido sistema *peer-to-peer*, isto implicaria que seria uma tarefa demasiado morosa e sujeita a problemas fazer a adaptação desta versão do algoritmo Vivaldi de modo a integrá-la no sistema em estudo. Foi também encontrada uma versão do algoritmo Vivaldi escrita em C++ [34], produzida na Universidade de Harvard (Estados Unidos da América). Esta versão é mais genérica e independente de qualquer sistema, isto é, consiste no algoritmo em si, sem estar ligada a nenhum sistema específico, ao contrário da versão anteriormente referida. Desta forma, considerou-se que esta última versão seria mais facilmente adaptável apesar de existir em primeira análise, uma dificuldade que seria a conversão de código fonte entre C++ e Java. Esta complicação foi no entanto ultrapassada sem grandes problemas mesmo apesar de ter sido um processo manual, no qual foi necessário ter alguns cuidados, nomeadamente de modo a manter a integridade o algoritmo original e sem incorrecções de código que pudessem gerar outros problemas.

Tal como para o algoritmo GNP, foi criada uma classe específica de nó para o algoritmo Vivaldi, a qual se resolveu dar a designação de "VivaldiNode" (Figura 23). Esta classe estende a classe abstracta "AbstractProximityNode" descrita anteriormente. Em relação à classe "AbstractProximityNode" foi personalizado o método que devolve a distância em termos de coordenadas entre dois nós de modo a reflectir os resultados da aplicação do algoritmo Vivaldi. O método que trata o evento de recepção de uma nova mensagem no nó foi também personalizado de modo efectuar duas acções que são específicas para este algoritmo, nomeadamente, fazer a actualização das coordenadas do nó, segundo o algoritmo do Vivaldi, de modo a reflectir o efeito da medição de latência entre o nó que enviou e o nó que recebeu a mensagem, mas também fazer com que o algoritmo Vivaldi possua um tempo de adaptação, tal como passarei a explicar em 5.3.3. Para além destas adaptações, a classe "VivaldiNode" inclui ainda a lógica do nó Vivaldi que proveio do código fonte original (convertido para a linguagem Java), o qual inclui os métodos necessários para efectuar a referida actualização das coordenadas do nó, tendo em conta a

latência medida ao nó origem aquando da recepção de mensagens. A classe "VivaldiNode" possui ainda alguns membros privados específicos do algoritmo Vivaldi, como as coordenadas actuais do nó, o erro estimado associado a essas coordenadas e as amostras de latências medidas no nó. Nesta versão adaptada ao sistema LiveFeeds, a cada mensagem recebida é efectuada uma medição de latência entre o nó que recebeu a mensagem e o nó do qual a mensagem foi propagada. Esta medição será usada como uma amostra de latência para o nó de origem. É possível guardar as últimas n amostras para cada nó. Cada amostra possui uma estampilha temporal. Uma amostra tem um peso no ajuste de coordenadas do algoritmo Vivaldi segundo a antiguidade da sua estampilha temporal. Neste caso, o peso de uma amostra vai ser inversamente proporcional à antiguidade da referida estampilha. O peso de guardar estas amostras em termos de memória e o seu processamento verificou-se demasiado elevado e incomportável em termos de simulação centralizada pelo que esta amostragem acabou por ser descartada na adaptação do algoritmo Vivaldi ao simulador. Esta adaptação e as suas implicações são discutidas na secção 5.3.3.

```
public class VivaldiNode extends AbstractProximityNode {
   public VivaldiNode() { … }

   public void onReceive(EndPoint src, BroadcastMessage msg) { … }
   public double distanceTo( AbstractProximityNode other) { … }

   private void update(VivaldiNode other) {
      Vivaldi.processSample(++stamp, this, other,
         (float)(endpoint.latency(other.endpoint) * 100));
   }
   // implementation of vivali node algorithm logic...
   …
}
```

Figura 23 – Resumo da classe VivaldiNode.

A lógica propriamente dita do algoritmo Vivaldi encontra-se na classe "Vivaldi" (Figura 24), que possui um método estático que faz o processamento de uma amostra, sendo assim responsável pela actualização das coordenadas para o nó. Este método é chamado a partir da classe "VivaldiNode" após a medição de latência que se constitui como uma amostra, sendo esta amostra passada ao método referido que fará o seu processamento. O método de processamento de uma amostra recebe ainda como argumentos, os nós entre os quais foi medida a latência de modo a poder actualizar não só as suas coordenadas mas também o erro estimado associado e as suas amostras de latências.

```
public class Vivaldi {
   public static int processedSamplesCounter;
   public static final Random VivaldiRandom = new Random(1);
   static int SAMPLE_EXPIRATION = 0;
   static double ERROR_FRACTION = 0.25;
   static double DAMPENING_FRACTION = 0.25;
   static double INITIAL_WEIGHTED_ERROR = 1.;
   public static boolean USING_SAMPLE_HISTORY = false;
   …
   public static void processSample(int o_stamp,
      VivaldiNode me, VivaldiNode you, float o_rawLatency) { … }
   …
}
```

Figura 24 – Resumo da classe Vivaldi.

4.9.3 Integração dos algoritmos “melhor” e “pior”

Para além dos tipos de nós GNP e Vivaldi foi ainda criada uma classe de nó que usa directamente as latências para cada nó de um intervalo de modo a determinar quais os próximos nós para os quais a mensagem será difundida. A esta classe de nós que, tal como o “GNPNode” e o “VivaldiNode”, estende a classe “AbstractProximityNode”, foi atribuída a designação de “KnowledgeNode” (Figura 25). Esta classe pode funcionar em dois modos, com escolha do nó mais próximo em termos de latência ou com escolha do nó mais distante em termos de latência. A forma de definir o modo em que a classe vai funcionar é através de uma variável. Assim, é possível tirar partido de uma facilidade dada pelo simulador, na qual é possível obter a latência a um nó sem ter de efectuar uma medições como seria de esperar em ambientes a funcionar sobre uma rede real. Este tipo de nó pode ser visto como o nó que possui conhecimento directo das latências em todos os momentos. A razão da utilização deste tipo de nó pode ser melhor entendida no capítulo que se segue e prende-se com a necessidade de ter uma forma de comparar as diferentes abordagens.

```
public class KnowledgeNode extends AbstractProximityNode {
   public enum ChoiceModes { Best, Worst }
   public static ChoiceModes choiceMode = ChoiceModes.Worst;

   public KnowledgeNode() {
      super() ;
      // uses network latency directly to measure
      // distance between nodes...
      this.coordinates = null;
   }

   public double distanceTo(AbstractProximityNode other) {
      return this.address.latency(other.address ) ;
   }

   public int compare(AbstractProximityNode a,
      AbstractProximityNode b) {
      if(choiceMode == ChoiceModes.Best)
         return super.compare(a, b);
      else {
         double dA = this.distanceTo(a);
         double dB = this.distanceTo(b);
         // Worst choice
         return (dA == dB) ?
            (a.key < b.key ? -1 : 1) : (dA > dB ? -1 : 1);
      }
   }
}
```

Figura 25 – Resumo da classe KnowledgeNode.

Neste capítulo foram apresentadas diversas abordagens possíveis para efectuar experimentação para sistemas distribuídos, como a experimentação com base na rede global, a emulação e finalmente a simulação. Esta última abordagem foi escolhida para efectuar a experimentação neste contexto. Existia já uma versão simplificada do algoritmo de *broadcast* do LiveFeeds integrada no simulador que foi usado. Foi apresentado o simulador e as suas características bem como a versão do algoritmo de *broadcast* do LiveFeeds e a sua integração no contexto do simulador. Foram integradas versões dos algoritmos GNP e Vivaldi no contexto do simulador, mais propriamente na versão do algoritmo LiveFeeds de modo ter uma matriz sintética de distâncias entre os diversos nós participantes, sendo que essa matriz que é dada pelas coordenadas virtuais fornecidas por cada um dos algoritmos. Foi ainda integrado um algoritmo que tira partido directo do conhecimento que é possível ter a cada momento sobre a latência entre nós participantes no simulador e assim obter uma forma de ter uma forma de avaliar e comparar entre si a qualidade das versões dos algoritmos GNP e Vivaldi integrados bem como das respectivas aproximações de matrizes de distâncias sintéticas. No capítulo que se segue, são apresentados de forma mais aprofundada os critérios de comparação e testes efectuados tendo por

base a plataforma apresentada neste capítulo bem como adaptações que tiveram de ser implementadas em cada algoritmo de modo a tornar os testes fazíveis no ambiente de simulação bem como a forma de obter resultados para os testes.

5. Critérios de comparação e planificação de testes

Neste capítulo são apresentados os critérios de comparação e planificação de testes tendo por base o simulador e o algoritmo de *broadcast* do LiveFeeds nele integrado, bem como as versões dos algoritmos GNP e Vivaldi que foram integrados neste ambiente. Desta forma tornou-se necessário considerar critérios de comparação válidos entre os diversos algoritmos em estudo que fornecem uma aproximação a uma matriz de distâncias entre os nós participantes.

Ao efectuar a integração destes algoritmos no ambiente anteriormente descrito tiveram de ser implementadas algumas modificações de modo a tornar os testes fazíveis em termos de simulação sendo ainda necessário implementar uma solução para se obter e compilar os resultados dos testes. Todas estas considerações são desenvolvidas ao longo deste capítulo.

5.1 Considerações sobre o algoritmo óptimo para o LiveFeeds através do uso de sistemas de proximidade relativa

Podemos considerar como dados do problema em causa as condições em que opera o algoritmo de *broadcast* do sistema LiveFeeds, mas também o ambiente de simulação usado. É necessário ter em conta esses dados de modo a poder efectuar uma avaliação aceitável e consistente neste contexto. Os mesmos podem ser considerados como restrições do problema e são descritos em seguida.

O sistema LiveFeeds assenta sobre uma de rede “overlay”, na qual cada nó pode comunicar directamente com qualquer outro nó, ou seja, tal como numa rede “full mesh”.

Os nós participantes são identificados através de identificadores escolhidos aleatoriamente. Na versão simplificada do sistema LiveFeeds em estudo, não existe gestão de filiação, pelo que todos os nós fazem inicialmente parte do sistema e cada um possui um identificador único. Os identificadores são números inteiros que não possuem qualquer informa-

ção sobre a localização ou proximidade relativa dos nós visto que a sua atribuição é feita sem ter por base qualquer informação acerca da topologia de rede.

Um dos pontos de partida que faz parte da definição do algoritmo de *broadcast* do sistema LiveFeeds é a forma como as mensagens são propagadas a todos os nós. Neste sistema, cada nó propaga a mensagem a enviar, para um número fixo de nós, o chamado *fanout factor*, a seguir designado por B.

Como já foi referido anteriormente, para propagar uma mensagem o nó divide o intervalo de identificadores em consideração em cada momento em B sub intervalos, enviando uma mensagem para um nó em cada intervalo. Cada mensagem é portanto difundida por uma árvore cujo grau é B.

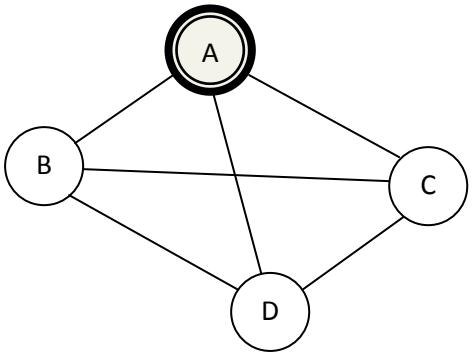
Por forma a mensurar e comparar de forma aceitável a possível melhoria que algoritmos de proximidade relativa ou de coordenadas virtuais podem oferecer no contexto do sistema e tendo em conta o que acima foi apresentado, pretende-se saber, em primeiro lugar qual o critério de comparação entre os diferentes algoritmos e de seguida é necessário determinar qual o melhor algoritmo para esse critério.

O critério de comparação deve ser o tempo médio que decorre desde o envio de uma mensagem até que a mesma chega a cada nó. O algoritmo que minimize esse tempo, por comparação com todas as outras aproximações possíveis, será o algoritmo óptimo.

Este critério de comparação é o que é usado normalmente para comparar sistemas cuja propagação de mensagens é feita através de uma árvore, tal como no caso em estudo, sendo o que faz mais sentido.

Numa primeira aproximação, como existe em cada momento uma divisão entre os nós do intervalo de escolha, uma abordagem que seria candidata a minimizar o tempo médio que cada mensagem demora a chegar a cada nó seria escolher em cada momento o nó mais próximo em cada sub intervalo de nós. Este algoritmo dito “guloso” apesar de possivelmente fornecer uma boa aproximação ao algoritmo óptimo, não é o algoritmo óptimo. Facilmente se encontram contra exemplos que o confirmam. Por exemplo:

Considerando o caso em que A é o nó emissor, e para simplificar, que o *fanout factor* é 1, para a configuração presente na Figura 26:



Considerando os dados:

$\overline{AB} = 1$

$\overline{AC} = 1 + 2\alpha$

$\overline{AD} = 1 + \alpha$

$\overline{BC} = \overline{BD} = 2$

$\overline{CD} = \alpha$

$\overline{DC} = \overline{CD}$

$\overline{CB} = \overline{BC}$

$\alpha \ll \overline{AB}$

Figura 26 – Ligações entre os nós (Contra exemplo)

Algoritmo *guloso*:

$$\overline{AB} + \overline{BC} + \overline{CD}$$

Tempo médio para cada nó desde o emissor:

$$\frac{1 + (1 + 2) + (1 + 2 + \alpha)}{3} = \frac{7 + \alpha}{3}$$

Contra exemplo:

$$\overline{AD} + \overline{DC} + \overline{CB}$$

Tempo médio para cada nó desde o emissor:

$$\frac{(1 + \alpha) + \left((1 + \alpha) + \alpha\right) + \left(\left((1 + \alpha) + \alpha\right) + 2\right)}{3} =$$

$$= \frac{5 + 5 \times \alpha}{3}$$

Para qualquer α no intervalo $]0, 1/2[$ o algoritmo "guloso" seria pior em média que o contra exemplo.

Tendo em conta o exemplo anterior, prova-se que o algoritmo "guloso" não é o algoritmo óptimo que minimiza o tempo médio que cada mensagem demora a chegar a cada nó.

É fácil inferir que o verdadeiro algoritmo óptimo, a existir, não será de cálculo trivial, dada a complexidade do problema, no qual seria necessário efectuar uma comparação baseada em toda a combinatória possível de resultados.

Não sendo fácil obter os resultados correspondentes ao algoritmo óptimo, uma forma simplificada de mensurar a qualidade e comparar entre si os diferentes algoritmos de proximidade, neste contexto, seria estabelecer uma escala entre aproximações do pior algoritmo possível, e do melhor algoritmo possível. Tendo estes dois extremos seria possível encaixar entre um "algoritmo óptimo aproximado" e o "algoritmo péssimo aproximado" todas as abordagens intermédias, que serão melhores ou piores consoante estejam mais próximas dessas aproximações do melhor possível ou do pior possível respectivamente. Assim, será possível estabelecer uma comparação clara entre as diferentes abordagens, neste caso, os algoritmos de proximidade relativa ou baseados em coordenadas virtuais.

Estabelecendo como hipótese que não é possível obter o "algoritmo óptimo" nem o "algoritmo péssimo", determine-se a validade dos critérios óptimo e péssimo aproximados, nos quais são escolhidos os nós mais próximos e mais distantes, respectivamente, em cada intervalo, a cada momento.

Existindo uma escala entre um algoritmo aproximadamente melhor e um aproximadamente pior, é possível inferir que o algoritmo de escolha aleatória estaria situado entre o melhor e o pior, mais precisamente no ponto intermédio dessa mesma escala, isto quando o número de ensaios tende para infinito. Desta forma, a escolha do nó mais próximo em cada um desses intervalos poderá ser considerada uma aproximação válida ao algoritmo óptimo em termos de comparação para o critério escolhido se, ao estabelecer uma escala entre a escolha do nó mais próximo em cada intervalo e a escolha do nó mais distante em cada intervalo, a escolha do nó aleatório, se situar aproximadamente a meio dessa escala.

Alguns ensaios preliminares efectuados deram a indicação de que o critério aleatório se situaria tendencialmente no ponto intermédio entre o algoritmo que escolhe o nó mais próximo em termos de latência em cada um dos intervalos em cada momento e o algoritmo que escolhe o nó mais distante em termos de latência em cada um dos intervalos em cada momento, nos termos definidos anteriormente. Resultados definitivos encontram-se no capítulo seguinte.

Assim, a escolha do nó mais próximo em cada um dos intervalos em cada momento pode ser considerada como um algoritmo aproximado ao óptimo, que designarei de algoritmo óptimo aproximado. Da mesma forma poder-se-á dizer que o algoritmo aproximado ao pior possível seria escolher o nó mais distante em cada momento para cada intervalo, que designarei de algoritmo péssimo aproximado.

5.2 Verificação das implicações do uso de coordenadas virtuais

O uso de algoritmos que fornecem uma aproximação de matriz de distâncias e a consequente escolha de nós em função dos resultados fornecidos por esses mesmos algoritmos, além de fornecer vantagens, poderá implicar outras consequências que podem não ser óbvias.

Uma das implicações possíveis do uso de critérios de escolha não aleatórios, poderá ser uma sobrecarga para alguns nós. Como já foi referido anteriormente, a propagação de uma mensagem no sistema LiveFeeds processa-se como numa árvore cujo grau é igual ao chamado *fanout factor*, pelo que os nós serão por vezes nós folha e outras vezes nós intermédios nessa árvore. Os nós folha, por não terem de propagar a mensagem para mais nós tendo assim menor carga de trabalho.

Através da escolha aleatória de nós, não baseada em critérios de latência, no limite, é previsível que cada nó seja folha um número de vezes que está relacionado com o *fanout factor* usado, isto é, com o grau da árvore de difusão, neste caso, quanto maior for o grau, maior a porção de nós folha de uma árvore equilibrada.

Quando existe uma escolha de nós baseada em critérios de latência, como os que são fornecidos pelos algoritmos GNP e Vivaldi, poderá ocorrer uma situação em que alguns nós, por estarem perto em termos de latência, relativamente a muitos outros nós sejam muitas vezes escolhidos como os próximos nós a encaminhar mensagens e assim sejam poucas vezes nós folha. Por consequência, estes nós estarão sujeitos a uma carga superior. Verificando-se simultaneamente o inverso, quando alguns nós raramente são escolhidos, tornando-se repetidamente nós folha.

Assim, também é desejável avaliar em que medida ocorre a situação descrita, quando são escolhidos os nós usando critérios baseados em latência, em relação à escolha aleatória e quais os efeitos resultantes para esses mesmos nós, nomeadamente nos casos em que exis-

tam nós cujo tráfego e carga são aumentados devido ao facto de serem constantemente escolhidos como nós intermédios e assim serem obrigados a encaminhar mensagens.

5.3 Planificação e testes

Depois de encontrado um critério adequado de comparação entre os diversos algoritmos e de modo a poder testar e obter resultados válidos e relevantes no contexto desta dissertação e segundo esse mesmo critério, foi necessário efectuar uma planificação de modo a prever qual o conjunto de testes a efectuar.

Para obter um conjunto de testes significativo a partir do qual seja possível extrair resultados válidos e respectivas conclusões é necessário fazer variar os parâmetros de simulação mais significativos neste contexto segundo valores aceitáveis e efectuar simulações segundo as combinações possíveis dessas variações para cada um dos algoritmos em comparação, isto é, entre o algoritmo original, que efectua a escolha dos nós de forma aleatória, e os algoritmos que fornecem uma aproximação de matriz de latências através de coordenadas virtuais e ainda o algoritmo que usa directamente as latências entre os diversos nós.

De entre o conjunto de variáveis que é possível fazer variar faz sentido fazer variar o número de nós participantes do sistema LiveFeeds de modo a aferir se o comportamento dos diversos algoritmos é influenciado por esta variável. Torna-se também necessário fazer variar o *fanout factor*, que é uma variável directamente relacionada com o algoritmo LiveFeeds e não com a simulação em si como o número de nós.

Para além destes parâmetros é necessário ter em conta quais os modelos de rede a usar. Durante a fase inicial do desenvolvimento da parte prática desta dissertação foi usada a chamada rede Euclidiana, que já foi descrita anteriormente. Numa fase mais avançada começou a ser usada a rede gerada pelo gerador de topologias de rede Orbis, também já descrito anteriormente. Este último tipo de rede possui um parâmetro que importa fazer variar neste contexto, nomeadamente o número de nós que se encontram no núcleo da rede Orbis.

A variação dos valores concretos destes parâmetros faz-se entre intervalos considerados razoáveis segundo o parâmetro em causa e a sua função, visto que não é possível testar

muitas combinações uma vez que isso significaria efectuar um número bastante elevado de simulações, o que não é possível em tempo útil.

As combinações usadas sobre as variações de parâmetros encontram-se na tabela seguinte (Tabela 2) e têm em conta a variação de número de nós para quando se usa uma rede gerada a partir do gerador de topologias de rede Orbis, o número de nós participantes no sistema e ainda o *fanout factor*.

Nº de nós Orbis (núcleo da rede)	Nº de nós LiveFeeds	Fanout Factor
500	500	3
		10
	2.000	3
		10
	10.000	3
		10
2.000	500	3
		10
	2.000	3
		10
	10.000	3
		10

Tabela 2 – Combinações das variações de parâmetros.

O número de combinações para estes parâmetros corresponde às simulações base que se considerou ser necessário efectuar. É possível questionar se estas variações serão suficientes para ter um conjunto de resultados representativo, por exemplo, efectuando simulações não só para estas combinações mas também um número mínimo de configurações de rede diferentes, usando os diferentes algoritmos em estudo que para cada uma dessas simulações de forma a obter os respectivos. No entanto, como se pode perceber, o número de simulações que é necessário efectuar de modo a obter resultados para todas essas combinações é bastante elevado, sendo que o tempo total de simulação seria impraticável com o tempo disponível para efectuar esta dissertação e possivelmente a variação de configurações de rede não faria uma diferença significativa nos resultados finais. Foram, no entanto, efectuadas as simulações de acordo com a tabela acima, sendo os respectivos resultados apresentados e analisados no capítulo seguinte.

Para cada simulação é necessário ainda enviar um número significativo de mensagens para depois obter os resultados que são baseados numa função de média. Esse número de

mensagens enviadas pelos nós é, no caso do algoritmo de escolha aleatória cinco vezes superior ao número de nós LiveFeeds da simulação em causa. Para o caso do algoritmo de escolha aleatória o número de mensagens trocadas foi assim escolhido devido à grande diversidade de escolhas possíveis. Para o caso do algoritmo GNP e do algoritmo que usa directamente o conhecimento das latências basta enviar uma mensagem por nó, isto é, cada nó envia uma mensagem. Neste caso, como a latência é estática e não existe evolução no algoritmo GNP não existiria mais valia em enviar mais do que uma mensagem por nó. No caso do algoritmo Vivaldi, como este é um algoritmo que tem uma evolução ao longo do tempo e um tempo de convergência, trata-se uma situação mais complexa que por isso é tratada na secção 5.3.3.

Para além do que já foi referido, tornou-se ainda necessário efectuar a verificação da questão levantada na secção anterior, sobre a forma como a escolha de nós através dos critérios anteriormente referidos poderá afectar positiva ou negativamente a carga para determinados nós. Assim, para cada simulação foi medido o número de mensagens enviadas por cada nó. Deste modo é possível fazer uma comparação entre os diversos nós para determinar se existe uma diferença significativa em alguns nós e especialmente em comparação com a versão original do algoritmo que efectua a propagação de mensagens de uma forma aleatória.

5.3.1 Forma de efectuar medições

De modo a obter resultados que possam ser avaliados e comparados no contexto do simulador, a forma encontrada de efectuar medições foi usar o objecto que é transportado em cada mensagem que é trocada entre cada nó do simulador. Na versão do algoritmo LiveFeeds adaptada ao simulador (“Tempest1”) o objecto transportado em cada mensagem nada contém. Esse objecto foi substituído por uma classe dedicada às medições e respectivos cálculos. Essa classe foi designada por “TrackablePayload” (Figura 27) e foi construída com o intuito de manter o controle sobre cada mensagem e do estado global das medições e assim obter as medições necessárias.

A classe “TrackablePayload” possui duas vertentes principais, uma é a vertente associada a cada mensagem trocada. A segunda vertente é a que agrega os dados e cálculos globais de simulação e para a qual contribui a informação medida em cada mensagem.

De cada vez que um nó emissor procede ao envio de uma mensagem este constrói um objecto da classe "TrackablePayload" que constituirá o conteúdo da mensagem. Ao construir o referido objecto é usado o construtor que recebe como argumento o próprio nó emissor, que vai ser guardado num membro privado da classe "TrackablePayload" assim como o tempo de simulação no qual a mensagem foi emitida. O nó emissor e o tempo de simulação em que a mensagem foi emitida vão servir de referência nos cálculos. Sempre que uma mensagem é recebida por um nó, este invoca um método de actualização presente na classe "TrackablePayload". Este método vai actualizar os dados obtidos em relação ao nó origem, usando a diferença entre o tempo de simulação actual e o tempo em que a mensagem foi emitida e assim manter sempre a noção de quanto tempo demorou uma mensagem proveniente de um dado emissor a chegar a determinado nó. Toda esta informação é guardada em tabelas de dispersão de modo ser facilmente acessível. No final da simulação é feita a média dos tempos, primeiro por cada nó emissor e em seguida para o conjunto das médias de tempos por nó emissor.

As características descritas permitem obter no final de uma simulação, o tempo médio que decorre desde o envio de uma mensagem até que a mesma chega a cada nó, que é precisamente o critério de comparação aceite como válido entre os diversos algoritmos em análise.

Uma característica da classe "TrackablePayload" é permitir obter resultados parciais, isto é, tendo apenas em conta uma fase da simulação específica. Esta característica foi integrada na classe devido ao tempo de adaptação do algoritmo Vivaldi (discutido na secção seguinte) permitindo assim obter resultados relativos apenas às fases posteriores de simulação em que apenas é usado o algoritmo Vivaldi.

A classe "TrackablePayload" possui ainda um método para agregar o número de mensagens enviadas por todos os nós, acedendo ao contador de mensagens enviadas presente na classe "AbstractProximityNode", exportando esses resultados sobre a forma de um gráfico de barras, no qual é possível visualizar facilmente a variação desse número de mensagens por nó. Para facilitar e dar um maior rigor à comparação entre os resultados para os diversos algoritmos é ainda calculado o desvio padrão de cada amostragem.

```
public class TrackablePayload {
   private AbstractNode lastNode;
   private AbstractNode sourceNode;
   private double startTime;
   …
   public void update(AbstractNode currentNode) { … }

   public int getCount() { … }
   public static void addNewDistance(double distance) { … }
   public static double getAverageDistance() { … }
   public AbstractNode getLastNode() { … }
   public static int getNCycles() { … }
   public static void startPartial(int nCycles) { … }
   public static void stopPartial(int nCycles) { … }
   public boolean isCompleted() { … }
   public static void generateChart() { … }
   …
}
```

Figura 27 – Resumo da classe TrackablePayload.

5.3.2 Adaptações específicas para o Algoritmo GNP

Para o caso do algoritmo GNP não foram necessárias adaptações para além das que seriam necessárias para a integração no ambiente de simulação e que já foram descritas no capítulo anterior. No entanto foram feitas e tentadas algumas formas de minimizar o tempo de processamento inicial. Este algoritmo possui uma fase inicial em que são computadas as coordenadas dos nós especiais, os chamados *landmarks*, após esta fase são computadas as coordenadas de cada nó participante individualmente. Estas duas fases são caracterizadas pelo processamento necessário de modo a resolver o problema de optimização e assim obter as coordenadas dos nós. Este processamento tem uma duração que pode ser normalmente da ordem dos minutos dependendo do número de nós e *landmarks* no sistema. Após o processamento inicial necessário para obter as coordenadas dos nós cada nó fica com as coordenadas associadas que são constantes ao longo da simulação, pelo que não existe mais processamento associado ao algoritmo GNP durante a simulação. Uma forma encontrada para tornar este processamento mais rápido foi usar processamento paralelo através de “threads”. O uso de desse processamento paralelo fez com que o tempo de processamento diminuísse consideravelmente, principalmente devido ao facto de durante o cálculo das coordenadas de alguns nós, o processamento ser especialmente demorado durante o problema de optimização. No entanto, o uso de *threads* foi abandonado por não ser garantido que estas interferissem de alguma forma com as *threads* próprias do simulador, nomeadamente com o relógio interno de simulação. Outra forma mais indirecta mas eficaz para diminuir o tempo de processamento inicial associa-

do ao algoritmo GNP foi guardar as coordenadas dos nós em ficheiro após o cálculo das mesmas. Desta forma, para uma simulação com a mesma raiz aleatória, isto é, com a mesma matriz de latências entre nós, as coordenadas virtuais GNP são carregadas de ficheiro em vez de serem de novo calculadas. O tempo de carregamento de coordenadas a partir de ficheiro é desprezável quando comparado com o tempo de cálculo destas. Assim foi possível repetir simulações sem ser necessário calcular sempre as coordenadas virtuais dos nós.

De modo a poder efectuar uma comparação válida entre os diferentes algoritmos neste contexto, é necessário efectuar a comparação através das medições efectuadas sobre os mesmos conjuntos de nós para cada algoritmo, isto é, para a mesma matriz de latências entre os nós. Para poder repetir simulações e assim efectuar comparações no mesmo contexto existe a raiz aleatória. Essa raiz faz parte do ambiente de simulação, assim como outros parâmetros, como por exemplo, o número de nós participantes. Cada vez que se usa ou modifica um dos parâmetros de ambiente de simulação torna-se necessário recalcular as coordenadas virtuais segundo o GNP. Estas são então guardadas para ficheiro e assim evita-se a repetição do referido cálculo nas simulações seguintes para o mesmo ambiente.

Das classes de nós criados para este fim, o nó GNP é o único para o qual é necessário efectuar o cálculo de coordenadas virtuais na fase inicial antes da simulação propriamente dita. Desta forma, o nó GNP é o único que tem informação associada guardada mesmo após uma simulação para ser usada em subsequentes simulações. Após o carregamento, no caso de o ficheiro com as coordenadas já existir ou do cálculo das mesmas em caso contrário, o nó da classe “GNPNode” é construído através da atribuição das respectivas coordenadas.

5.3.3 Adaptações específicas e dificuldades para o algoritmo Vivaldi

Devido à complexidade do algoritmo Vivaldi e ao facto de este estar integrado num ambiente de simulação centralizado e por isso limitado em alguns aspectos, foi necessário efectuar algumas modificações e optimizações no código e em alguns aspectos de funcionamento do algoritmo Vivaldi. Existiram algumas dificuldades associadas a este processo de adaptação mas também algumas dúvidas sobre se estas modificações teriam alguma

implicação no normal funcionamento do algoritmo Vivaldi. Estas questões são descritas e discutidas ao longo desta secção.

Eliminação do histórico de amostras e suas implicações

Na versão do algoritmo Vivaldi adaptada ao sistema LiveFeeds, a cada mensagem recebida é efectuada uma medição de latência entre o nó que recebeu a mensagem e o nó a partir do qual a mensagem foi propagada. Na versão original do algoritmo Vivaldi, esta medição é usada como uma amostra de latência para o nó de origem, sendo possível guardar as últimas n amostras para cada nó. A cada amostra é associada uma estampilha temporal. Segundo a antiguidade da estampilha temporal assim será o seu peso no ajuste de coordenadas do algoritmo Vivaldi. Neste caso, o peso de uma amostra vai ser inversamente proporcional à antiguidade da respectiva estampilha.

O peso de guardar estas amostras em termos de memória e processamento aquando do reajuste de coordenadas virtuais revelou-se demasiado elevado e incomportável em termos de simulação centralizada. À medida que se efectuavam experimentações sobre o algoritmo Vivaldi neste contexto, verificou-se que a quantidade de memória ocupada pelo histórico de amostragem era demasiada para uma computador vulgar, mesmo para um número baixo de amostras guardadas. O processamento associado ao tratamento da amostragem também se verificou ser demasiado elevado. Devido a estas situações a amostragem acabou por ser descartada na adaptação do algoritmo Vivaldi ao simulador. A eliminação do histórico de amostras poderia ter implicações negativas no desempenho do algoritmo Vivaldi se o modelo de simulação fosse mais dinâmico, no entanto, como este é um modelo estático e as latências são constantes não existe qualquer vantagem em guardar e usar o histórico de amostras, pelo que este pode ser considerado dispensável.

Tempo de adaptação e factores de instabilidade do algoritmo Vivaldi

No contexto de simulação usado, ao iniciar uma simulação usando o algoritmo Vivaldi, as coordenadas iniciais de cada nó são todas iguais à origem do referencial. À medida que se processam mensagens, as coordenadas de cada nó vão sendo ajustadas individualmente em função das latências medidas. Como as coordenadas da generalidade dos nós do algoritmo Vivaldi partem da origem do referencial existe uma fase inicial em que estas não são estáveis. Devido a essa instabilidade inicial, que é característica do algoritmo Vivaldi, a escolha de nós tendo em conta as coordenadas dadas pelo mesmo vai aumentando gra-

dualmente, por fases, à medida que a simulação vai evoluindo, partindo de uma situação em que os nós são escolhidos aleatoriamente até que são finalmente escolhidos apenas tendo em conta as coordenadas dadas pelo algoritmo. Na fase inicial os nós são escolhidos aleatoriamente, passando em seguida a uma fase em que são escolhidos segundo as coordenadas dadas pelo algoritmo Vivaldi em metade das ocasiões, passando ainda por uma fase em que essa escolha ocorre em setenta por cento das ocasiões até que, atingindo uma fase considerada mais estável, a escolha ocorre apenas tendo em conta as coordenadas virtuais dadas pelo algoritmo Vivaldi. Sendo que a duração ou comprimento de uma simulação é determinada pelo número de mensagens envidadas pelos nós e sendo esse número um dos parâmetros de simulação, apresenta-se na Tabela 3 a distribuição de probabilidades para um nó ser escolhido de acordo com o algoritmo Vivaldi – $P(Vivaldi)$, tendo em conta a fase em que se encontra a simulação. Distribuição de probabilidades da utilização do algoritmo Vivaldi ao longo de uma simulação.

Fase de simulação (Início – 0 %; Fim – 100 %)	$P(Vivaldi)$
0 – 10 %	0
10 – 20 %	0,5
20 – 33,3(3) %	0,7
33,3(3) – 100 %	1

Tabela 3 – Distribuição de probabilidades da utilização do algoritmo Vivaldi ao longo de uma simulação.

Desta forma espera-se que a convergência de coordenadas para um estado mais estável seja mais rápida e menos influenciada pelo próprio algoritmo Vivaldi, que ao serem efectuadas as escolhas pode acontecer que alguns nós sejam menos escolhidos e isso ter uma influência negativa na estabilização de coordenadas, principalmente numa fase inicial.

As características específicas do algoritmo Vivaldi fazem com que tenha uma complexidade em termos de simulação superior ao algoritmo GNP ou à versão de escolha aleatória. Esta complexidade acrescida verifica-se sobretudo em termos de processamento e ocupação de memória física e fica-se a dever principalmente ao dinamismo do algoritmo Vivaldi, que faz com que cada nó esteja constantemente em reajustamento das suas coordenadas, o que aumenta consideravelmente o processamento, e tenha um histórico de

amostras por nó, o que faz com que a ocupação de memória física possa atingir níveis bastante elevados ou até mesmo incomportáveis dependendo do número de nós usado em simulação.

Vivaldi+h

O algoritmo Vivaldi possui uma evolução que pode ser designada por "Vivaldi mais altura", ou de uma forma mais reduzida "Vivaldi+h". Esta evolução foi descrita no capítulo 2, na secção dedicada ao algoritmo Vivaldi, e consiste em ter em conta uma componente extra, a altura (h). O valor desta componente extra modela o peso do chamado "local loop", mais especificamente as condições em que o nó em causa se liga à Internet, enquanto que as restantes componentes modelam o núcleo da rede. Segundo os autores deste algoritmo e desta variante, esta introduz uma melhoria na precisão do algoritmo.

Esta evolução do algoritmo Vivaldi encontra-se presente no código que foi adaptado ao contexto de simulação e pode ser ligada ou desligada através de uma simples variável presente na classe que implementa a lógica do algoritmo Vivaldi e que tem o mesmo nome deste algoritmo. Devido à importância desta evolução do algoritmo Vivaldi e a facilidade com que pode ser testada, esta passa então a estar em paralelo com o algoritmo original do Vivaldi e com os restantes algoritmos em estudo para ser testada e avaliada neste contexto.

Optimizações efectuadas e limites de simulação

A complexidade do algoritmo Vivaldi e o facto de este ser um algoritmo em constante adaptação e evolução de forma paralela à evolução das condições de rede fazem que este seja especialmente pesado num ambiente de simulação centralizado, a nível de memória, de processamento e de tempo de execução. Uma simulação usando o algoritmo Vivaldi e um número de nós considerável consome uma grande quantidade de memória e processamento. Para além disso uma simulação nestas condições demora bastante tempo até ser concluída, na ordem de horas de simulação.

Numa primeira fase tornou-se muito complicado efectuar simulações de teste usando o algoritmo Vivaldi, devido às questões já referidas mas também porque o código obtido e o próprio ambiente de simulação requeriam alguns ajustes. Na fase inicial foi apenas possível testar a integração do algoritmo Vivaldi no ambiente de simulação usando um

número diminuto de nós pois a máquina virtual do Java não possuía memória suficiente para lidar com maiores quantidades de nós no sistema. Posteriormente foi aumentada a atribuição de memória (*heap size*) à máquina virtual do Java, o que minimizou um pouco o impacto dessa situação. Mesmo assim, por muito que se aumentasse a capacidade de memória atribuída à máquina virtual do Java, para quantidades de nós mais significativas esta medida não foi suficiente devido ao limite físico de memória das máquinas usadas em testes. A partir de uma certa fase de simulação a memória atribuída não era suficiente e o sistema entrava num estado de *trashing*, no qual o "garbage collector" passava a ocupar toda a capacidade de processamento do computador numa tentativa de libertar memória. Perante este cenário foram efectuadas algumas optimizações no código que foi obtido do algoritmo Vivaldi, o qual não estaria preparado para ser integrado num ambiente de simulação centralizado com todas as suas condicionantes. Estas optimizações tiveram um efeito positivo, no entanto, este não foi suficiente. Apenas quando foi desligado o histórico de amostras do Vivaldi foi possível efectuar simulações para uma quantidade de nós significativa usando o algoritmo Vivaldi, pelo que essa acção teve uma enorme importância para que o trabalho pudesse prosseguir.

Mesmo após todas as optimizações e modificações referidas, no sentido de tornar o algoritmo Vivaldi passível de ser usado no ambiente de simulação centralizado que foi usado na parte prática nesta dissertação, a simulação usando este algoritmo e até mesmo os restantes continua a ser limitada pela quantidade de memória física disponível no computador. Não foi estabelecido um limite concreto, mas é de prever que para uma quantidade de nós acima dos dez mil o tempo de simulação para o algoritmo Vivaldi seria demasiado num computador actual visto que uma simulação usando o algoritmo Vivaldi com dez mil nós LiveFeeds participantes durou nunca menos de seis horas, o que é bastante, tendo em conta que é necessário efectuar mais do que uma simulação por cada combinação das variações de parâmetros anteriormente referidas.

6. Resultados e sua discussão

Neste capítulo é feita a apresentação dos resultados obtidos a partir do que foi descrito anteriormente. É ainda feita a análise destes resultados sendo finalmente discutido o seu significado.

Como foi referido no capítulo anterior, o critério de comparação entre os diversos algoritmos presentes neste estudo foi o tempo médio que decorre desde o envio de uma mensagem até que a mesma chega a cada nó. De modo a ter uma base de comparação razoável entre os diversos algoritmos, foram estabelecidos limites, superior e inferior, que são dados pela escolha directa através de comparação de latências em cada momento, o que é conseguido através da classe "KnowledgeNode", que como já foi referido, representa um nó do simulador que efectua as escolhas seguindo esse mesmo critério, funcionando em dois modos: escolha do melhor e escolha do pior em termos de latência de entre os nós cuja escolha é possível em cada momento.

Também foi medido de que forma a escolha de nós de acordo com critérios não aleatórios afecta a carga na generalidade dos nós. Isto é, se existem nós que passam a efectuar demasiado trabalho, por oposição a outros que vêm a sua carga diminuída. Como a difusão de mensagens se processa numa árvore cujo grau é igual ao *fanout factor*, os nós intermédios dessa árvore vão emitir no máximo *fanout factor* mensagens, enquanto que os nós que são folha não emitem mensagens. Se a escolha de nós for aleatória, no limite, é de prever que todos os nós emitirão um número igual de mensagens, sendo que o número de nós folha na árvore de difusão é uma função do grau da árvore, sendo tanto maior quanto maior o grau. De modo a medir e poder comparar a carga nos diversos nós são contadas as mensagens enviadas por cada nó durante uma simulação.

Como também já foi referido anteriormente, foram efectuados numerosos ensaios de acordo com variações de diversos parâmetros. Em seguida apresentam-se esses mesmos resultados, começando pelos resultados relativos ao tempo médio de chegada das mensagens aos nós, apresentando posteriormente os resultados relativos ao número de mensagens enviadas por cada nó de modo a comparar a carga nestes.

6.1 Apresentação de resultados

Em seguida são apresentados os resultados relativos ao tempo médio que decorre desde o envio de uma mensagem até que a mesma chega a cada nó. Os resultados são apresentados numa tabela cuja primeira coluna identifica o algoritmo, a segunda coluna apresenta o referido tempo médio em termos absolutos para o algoritmo respectivo e a terceira e última coluna apresenta esse tempo médio em termos de percentagem, em que o algoritmo melhor (aqui identificado por "Knowedge – Best") corresponde a 100% e o algoritmo pior (aqui identificado por "Knowedge – Worst") corresponde a 0 %, de acordo com a escala já descrita no capítulo anterior. Para cada tabela são identificados os parâmetros dos ensaios.

6.1.1 Rede Euclidiana

Nesta subsecção apresentam-se os resultados relativos ao tempo médio que decorre desde o envio de uma mensagem até que a mesma chega a cada nó para uma rede euclidiana a duas dimensões.

Os resultados apresentados na Tabela 4 mostram o algoritmo aleatório bastante deslocado (cerca de 59%) em relação à posição intermédia entre o algoritmo "pior" e o algoritmo "melhor" (50%), mostram ainda um desempenho do algoritmo GNP de aproximadamente 100%, isto é, com uma aproximação elevada ao desempenho do algoritmo "melhor". As duas variantes do algoritmo Vivaldi encontram-se com um desempenho aproximado entre elas, sendo em ambos os casos um desempenho que apesar de não ser tão bom como o do algoritmo GNP, é ainda assim bastante aproximado ao algoritmo "melhor".

	Tempo médio (emissão-recepção)	%
Knowlege - Best	0,099921423	**100**
Knowlege - Worst	0,493608748	**0**
Aleatório	0,260712176	**59,15775205**
GNP	0,09994363	**99,99435916**
Vivaldi (0-3750)	0,145658908	**88,38228154**
Vivaldi (750-3750)	0,127001014	**93,12154869**
Vivaldi (1250-3750)	0,122489592	**94,26748897**
Vivaldi+h (0-3750)	0,145787271	**88,34967612**
Vivaldi+h (750-3750)	0,126758694	**93,18310003**
Vivaldi+h (1250-3750)	0,122103187	**94,36563919**

Tabela 4 – Resultados para rede Euclidiana, 500 nós LiveFeeds e *fanout factor* = 3.

Os resultados apresentados na Tabela 5 mostram o algoritmo aleatório novamente bastante deslocado em relação à posição intermédia entre o algoritmo "pior" e o algoritmo "melhor". O desempenho do algoritmo GNP é de novo aproximadamente 100%. As duas variantes do algoritmo Vivaldi continuam com um desempenho aproximado entre elas, mas neste caso um pouco mais baixo do que o apresentado na tabela anterior.

	Tempo médio (emissão-recepção)	%
Knowlege - Best	0,075135164	**100**
Knowlege - Worst	0,242099041	**0**
Aleatório	0,145608366	**57,79134772**
GNP	0,075148601	**99,99195193**
Vivaldi (0-3750)	0,100074928	**85,06277879**
Vivaldi (750-3750)	0,092997743	**89,30153066**
Vivaldi (1250-3750)	0,092477165	**89,61332175**
Vivaldi+h (0-3750)	0,099076015	**85,66105944**
Vivaldi+h (750-3750)	0,092086246	**89,84745528**
Vivaldi+h (1250-3750)	0,091350335	**90,28821612**

Tabela 5 – Resultados para rede Euclidiana, 500 nós LiveFeeds e *fanout factor* = 10.

Os resultados apresentados na Tabela 6 mostram-se aproximados aos resultados apresentados na Tabela 4, sendo que a única variável que muda entre estas duas tabelas é o número de nós LiveFeeds, que passa de 500 na Tabela 4, para 2000 na Tabela 6.

	Tempo médio (emissão-recepção)	%
Knowlege - Best	0,102837585	**100**
Knowlege - Worst	0,651367846	**0**
Aleatório	0,323540357	**59,76470435**
GNP	0,102859025	**99,99609153**
Vivaldi (0-15000)	0,165543979	**88,56828914**
Vivaldi (3000-15000)	0,138732022	**93,45625221**
Vivaldi (5000-15000)	0,132456591	**94,60029684**
Vivaldi+h (0-15000)	0,164633771	**88,73422491**
Vivaldi+h (3000-15000)	0,137955183	**93,59787415**
Vivaldi+h (5000-15000)	0,131848121	**94,71122418**

Tabela 6 – Resultados para rede Euclidiana, 2.000 nós LiveFeeds e *fanout factor* = 3.

Os resultados apresentados na Tabela 7 mostram-se aproximados aos resultados apresentados na Tabela 6, sendo que a única variável que muda entre estas duas tabelas é o *fanout factor*, que passa de 3 na Tabela 6, para 10 na Tabela 7.

	Tempo médio (emissão-recepção)	%
Knowlege - Best	0,078533439	**100**
Knowlege - Worst	0,322734324	**0**
Aleatório	0,176514662	**59,87679453**
GNP	0,078538834	**99,99779098**
Vivaldi (0-15000)	0,108720403	**87,63847098**
Vivaldi (3000-15000)	0,097094202	**92,39938763**
Vivaldi (5000-15000)	0,094672387	**93,3911181**
Vivaldi+h (0-15000)	0,108352297	**87,78920969**
Vivaldi+h (3000-15000)	0,097109378	**92,39317311**
Vivaldi+h (5000-15000)	0,094443864	**93,48469815**

Tabela 7 – Resultados para rede Euclidiana, 2.000 nós LiveFeeds e *fanout factor* = 10.

Os resultados apresentados na Tabela 8 mostram-se aproximados aos resultados anteriores, no entanto, acentua-se ainda mais a deslocação do algoritmo aleatório em relação à posição intermédia entre o algoritmo "pior" e o algoritmo "melhor".

	Tempo médio (emissão-recepção)	%
Knowlege - Best	0,10448957	**100**
Knowlege - Worst	0,855050819	**0**
Aleatório	0,392632559	**61,60966353**
GNP	0,104498597	**99,99879718**
Vivaldi (0-75000)	0,184660231	**89,31857174**
Vivaldi (15000-75000)	0,148897462	**94,08337525**
Vivaldi (25000-75000)	0,140265568	**95,23343379**
Vivaldi+h (0-75000)	0,184628159	**89,32284476**
Vivaldi+h (15000-75000)	0,149059874	**94,06173647**
Vivaldi+h (25000-75000)	0,140231196	**95,23801333**

Tabela 8 – Resultados para rede Euclidiana, 10.000 nós LiveFeeds e *fanout factor* = 3.

Os resultados apresentados na Tabela 9 mostram-se aproximados aos resultados anteriores, acentua-se ainda mais a deslocação do algoritmo aleatório em relação à posição intermédia entre o algoritmo "pior" e o algoritmo "melhor", situando-se acima dos 62%. É ainda de salientar que o desempenho do algoritmo GNP ultrapassou ligeiramente o desempenho do algoritmo "melhor", o que vem confirmar o que foi referido e exemplificado no capítulo anterior, em que é dado um contra exemplo simples em como é possível obter caminhos mais curtos do que os que são dados pelo algoritmo "melhor", então designado por algoritmo "guloso".

	Tempo médio (emissão-recepção)	%
Knowlege - Best	0,076383623	**100**
Knowlege - Worst	0,409294066	**0**
Aleatório	0,202325123	**62,16955567**
GNP	0,076376472	**100,002148**
Vivaldi (0-75000)	0,118835193	**87,24835139**
Vivaldi (15000-75000)	0,10440492	**91,58293258**
Vivaldi (25000-75000)	0,101948997	**92,32064523**
Vivaldi+h (0-75000)	0,119882877	**86,93364691**
Vivaldi+h (15000-75000)	0,105535839	**91,24322591**
Vivaldi+h (25000-75000)	0,103314614	**91,91043992**

Tabela 9 – Resultados para rede Euclidiana, 10.000 nós LiveFeeds e *fanout factor* = 10.

As principais observações a reter dos ensaios realizados para a rede euclidiana a duas dimensões prendem-se com o deslocamento bastante acentuado do desempenho do algoritmo aleatório em relação ao ponto médio de 50% entre o algoritmo "melhor" e o "pior", estando geralmente próximo dos 60%. O desempenho na ordem de 100% do algoritmo GNP e o não tão bom mas ainda assim bastante elevado desempenho de ambas as variantes do algoritmo Vivaldi, que no caso desta rede euclidiana são aproximados entre si.

6.1.2 Rede Orbis – 500 nós

Nesta subsecção apresentam-se os resultados relativos ao tempo médio que decorre desde o envio de uma mensagem até que a mesma chega a cada nó para uma rede gerada através do gerador de topologias Orbis com 500 nós internos.

Os resultados apresentados na Tabela 10 mostram o algoritmo aleatório aproximadamente a meio entre o algoritmo "pior" e o algoritmo "melhor", mostram ainda um desempenho do algoritmo GNP (cerca de 76%) muito abaixo do que se verifica na rede euclidiana, sendo que o algoritmo Vivaldi mostra-se mais próximo do algoritmo "melhor", especialmente a variante Vivaldi+h, que no melhor dos casos, ultrapassa os 92%.

	Tempo médio (emissão-recepção)	%
Knowlege - Best	0,822343743	**100**
Knowlege - Worst	1,703849453	**0**
Aleatório	1,258553509	**50,5153783**
GNP	1,029715794	**76,47524581**
Vivaldi (0-3750)	1,045945481	**74,63411349**
Vivaldi (750-3750)	1,004519486	**79,33357199**
Vivaldi (1250-3750)	0,993567538	**80,57598573**
Vivaldi+h (0-3750)	0,951972066	**85,29467002**
Vivaldi+h (750-3750)	0,899054696	**91,29773614**
Vivaldi+h (1250-3750)	0,885167636	**92,87311555**

Tabela 10 – Resultados para rede Orbis (500 nós), 500 nós LiveFeeds e *fanout factor* = 3.

Os resultados apresentados na Tabela 11, com um *fanout factor* de 10, mostram mais uma vez o algoritmo aleatório aproximadamente a meio entre o algoritmo "pior" e o algoritmo "melhor", mostram ainda um desempenho do algoritmo GNP semelhante aos dados anteriores, sendo que o desempenho de ambas as variantes do algoritmo Vivaldi piora ligeiramente em relação aos dados anteriores, sendo ligeiramente inferior ao desempenho do algoritmo GNP para a variante Vivaldi mas ainda assim bastante é superior ao desempenho do algoritmo GNP para a variante Vivaldi+h.

	Tempo médio (emissão-recepção)	%
Knowlege - Best	0,511923972	**100**
Knowlege - Worst	0,888775678	**0**
Aleatório	0,705380813	**48,66499534**
GNP	0,601049862	**76,34987767**
Vivaldi (0-3750)	0,630654407	**68,49412305**
Vivaldi (750-3750)	0,616460154	**72,26065834**
Vivaldi (1250-3750)	0,611627826	**73,54294744**
Vivaldi+h (0-3750)	0,580785795	**81,72707685**
Vivaldi+h (750-3750)	0,558512764	**87,6373673**
Vivaldi+h (1250-3750)	0,55400251	**88,83419197**

Tabela 11 – Resultados para rede Orbis (500 nós), 500 nós LiveFeeds e *fanout factor* = 10.

Os resultados apresentados na Tabela 12, com um *fanout factor* de 3 e desta vez para 2.000 nós LiveFeeds, mostram-se algo semelhantes aos resultados apresentados na Tabela 10 para o mesmo *fanout factor* mas com apenas 500 nós. O algoritmo Vivaldi+h apresenta aqui a sua melhor aproximação ao algoritmo "melhor", ultrapassando os 94%.

	Tempo médio (emissão-recepção)	%
Knowlege - Best	0,897873717	**100**
Knowlege - Worst	2,177409399	**0**
Aleatório	1,574535063	**47,11664894**
GNP	1,202536456	**76,18958627**
Vivaldi (0-15000)	1,226093347	**74,34853638**
Vivaldi (3000-15000)	1,162496347	**79,31885497**
Vivaldi (5000-15000)	1,144900691	**80,69401444**
Vivaldi+h (0-15000)	1,080087646	**85,75937104**
Vivaldi+h (3000-15000)	0,993491594	**92,52714256**
Vivaldi+h (5000-15000)	0,969705677	**94,3860917**

Tabela 12 – Resultados para rede Orbis (500 nós), 2.000 nós LiveFeeds e *fanout factor* = 3.

Os resultados apresentados na Tabela 13, com um *fanout factor* de 10, mostram mais uma vez o algoritmo aleatório aproximadamente a meio entre o algoritmo "pior" e o algoritmo "melhor", mostram ainda um desempenho do algoritmo GNP semelhante aos dados apresentados na Tabela 12, sendo que o desempenho de ambas as variantes do algoritmo Vivaldi piora ligeiramente em relação aos dados apresentados na Tabela 12, mas ainda assim é superior ao desempenho do algoritmo GNP.

	Tempo médio (emissão-recepção)	%
Knowlege - Best	0,569100875	**100**
Knowlege - Worst	1,143790666	**0**
Aleatório	0,861715478	**49,08303457**
GNP	0,705056821	**76,34272473**
Vivaldi (0-15000)	0,72288281	**73,2408793**
Vivaldi (3000-15000)	0,695566805	**77,99405322**
Vivaldi (5000-15000)	0,688918041	**79,15098416**
Vivaldi+h (0-15000)	0,664882573	**83,33332195**
Vivaldi+h (3000-15000)	0,629827445	**89,43315668**
Vivaldi+h (5000-15000)	0,622081601	**90,7809872**

Tabela 13 – Resultados para rede Orbis (500 nós), 2.000 nós LiveFeeds e *fanout factor* = 10.

Os resultados apresentados na Tabela 14, com 10.000 nós LiveFeeds marcam uma diferença em relação ao algoritmo GNP, visto que este piora ligeiramente o seu desempenho relativamente aos resultados anteriores, onde o desempenho deste algoritmo foi aproximadamente constante. Em relação às duas variantes do algoritmo Vivaldi, o desempenho pode ser considerado semelhante ao que foi visto nos resultados anteriores, sendo que a variante Vivaldi+h mantém-se como a que apresenta uma maior aproximação ao algoritmo "melhor", ultrapassando mais uma vez os 92%.

	Tempo médio (emissão-recepção)	%
Knowlege - Best	0,966936664	**100**
Knowlege - Worst	2,779355704	**0**
Aleatório	1,909076479	**48,01755035**
GNP	1,448404598	**73,4350653**
Vivaldi (0-75000)	1,467521267	**72,38030541**
Vivaldi (15000-75000)	1,384809697	**76,94390624**
Vivaldi (25000-75000)	1,361733922	**78,21710933**
Vivaldi+h (0-75000)	1,256320717	**84,0332701**
Vivaldi+h (15000-75000)	1,13992983	**90,4551231**
Vivaldi+h (25000-75000)	1,106396231	**92,30533533**

Tabela 14 – Resultados para rede Orbis (500 nós), 10.000 nós LiveFeeds e *fanout factor* = 3.

Os resultados apresentados na Tabela 15, que o algoritmo GNP mantém o seu desempenho relativamente aos resultados da Tabela 14, no entanto, o desempenho do algoritmo Vivaldi, nas suas duas variantes, baixa cerca de 4 pontos percentuais em relação aos resultados presentes na Tabela 14.

	Tempo médio (emissão-recepção)	%
Knowlege - Best	0,578689532	**100**
Knowlege - Worst	1,395378784	**0**
Aleatório	0,98361505	**50,41865467**
GNP	0,797677377	**73,18590337**
Vivaldi (0-75000)	0,827680407	**69,51216461**
Vivaldi (15000-75000)	0,796139982	**73,37415068**
Vivaldi (25000-75000)	0,790345842	**74,08361748**
Vivaldi+h (0-75000)	0,735957537	**80,74322585**
Vivaldi+h (15000-75000)	0,690083472	**86,36030297**
Vivaldi+h (25000-75000)	0,681437825	**87,41892431**

Tabela 15 – Resultados para rede Orbis (500 nós), 10.000 nós LiveFeeds e *fanout factor* = 10.

No caso dos ensaios para a rede Orbis com 500 nós internos verificou-se que, ao contrário do que aconteceu para a rede euclidiana, o desempenho do algoritmo aleatório ficou situado aproximadamente nos 50%, como era esperado. Outra verificação a reter é o desempenho quase sempre constante do algoritmo GNP, excepto quando o número de nós LiveFeeds foi o máximo (10.000), em que o seu desempenho decresceu cerca de três pontos percentuais, de aproximadamente 76% para aproximadamente 73%. Verificou-se ainda que o desempenho da variante simples (sem altura) do algoritmo Vivaldi foi na maioria dos casos ligeiramente melhor que o desempenho do algoritmo GNP, se não for contado o período de aprendizagem do Vivaldi. A variante Vivaldi+h foi a que teve um

desempenho mais aproximado ao do algoritmo "melhor", no melhor dos casos ultrapassando os 94%.

6.1.1 Rede Orbis – 2.000 nós

Nesta subsecção apresentam-se os resultados relativos ao tempo médio que decorre desde o envio de uma mensagem até que a mesma chega a cada nó para uma rede gerada através do gerador de topologias Orbis com 2.000 nós internos.

Os resultados apresentados na Tabela 16 mostram um deslocamento acentuado do desempenho do algoritmo aleatório para os 58%. Em relação ao algoritmo GNP, o seu desempenho apresenta-se bastante aproximado ao que se verificou para o mesmo número de nós LiveFeeds, na rede Orbis com 500 nós internos. As duas variantes do algoritmo Vivaldi continuam a ter um desempenho mais aproximado ao do algoritmo "melhor", sendo que a variante Vivaldi+h apresenta o seu melhor desempenho até agora registado: quase 96%.

	Tempo médio (emissão-recepção)	%
Knowlege - Best	0,908184999	**100**
Knowlege - Worst	2,033325598	**0**
Aleatório	1,372287002	**58,75164373**
GNP	1,167089813	**76,98911458**
Vivaldi (0-3750)	1,15261177	**78,2758909**
Vivaldi (750-3750)	1,109686084	**82,09103065**
Vivaldi (1250-3750)	1,097124672	**83,20746106**
Vivaldi+h (0-3750)	1,032019927	**88,99382641**
Vivaldi+h (750-3750)	0,971984547	**94,32963773**
Vivaldi+h (1250-3750)	0,953394491	**95,98188067**

Tabela 16 – Resultados para rede Orbis (2.000 nós), 500 nós LiveFeeds e *fanout factor* = 3.

Os resultados apresentados na Tabela 17 mostram um deslocamento acentuado do desempenho do algoritmo aleatório para os 56%. Em relação ao algoritmo GNP, o seu desempenho apresenta-se bastante aproximado aos resultados apresentados na tabela anterior. As duas variantes do algoritmo Vivaldi continuam a ter um desempenho mais aproximado ao do algoritmo "melhor", no entanto, menor que o desempenho obtido com um *fanout factor* = 3.

	Tempo médio (emissão-recepção)	%
Knowlege - Best	0,561542554	**100**
Knowlege - Worst	1,044759027	**0**
Aleatório	0,769600855	**56,94304456**
GNP	0,675657695	**76,38426089**
Vivaldi (0-3750)	0,691398107	**73,12683643**
Vivaldi (750-3750)	0,676961031	**76,11454003**
Vivaldi (1250-3750)	0,673121538	**76,90910997**
Vivaldi+h (0-3750)	0,633150965	**85,18088371**
Vivaldi+h (750-3750)	0,608648638	**90,2515565**
Vivaldi+h (1250-3750)	0,603128059	**91,39402168**

Tabela 17 – Resultados para rede Orbis (2.000 nós), 500 nós LiveFeeds e *fanout factor* = 10.

Os resultados apresentados na Tabela 18 apresentam um ligeiro aumento no desempenho do algoritmo GNP, sendo que o deslocamento do algoritmo aleatório continua a manter-se em relação ao que se vinha a verificar. A variante Vivaldi+h continua a ser a que tem um desempenho mais próximo do algoritmo "melhor", a ultrapassar os 91%, enquanto a variante Vivaldi apenas ultrapassa ligeiramente o algoritmo GNP em termos percentuais.

	Tempo médio (emissão-recepção)	%
Knowlege - Best	1,007462838	**100**
Knowlege - Worst	2,620542146	**0**
Aleatório	1,706067081	**56,69126499**
GNP	1,370854957	**77,47214802**
Vivaldi (0-15000)	1,360141271	**78,13632403**
Vivaldi (3000-15000)	1,29647787	**82,08302403**
Vivaldi (5000-15000)	1,278651056	**83,18816586**
Vivaldi+h (0-15000)	1,196589393	**88,27543361**
Vivaldi+h (3000-15000)	1,10621615	**93,87796304**
Vivaldi+h (5000-15000)	1,079855652	**95,5121355**

Tabela 18 – Resultados para rede Orbis (2.000 nós), 2.000 nós LiveFeeds e *fanout factor* = 3.

Os resultados apresentados na Tabela 19 apresentam uma manutenção no desempenho do algoritmo GNP em relação aos resultados apresentados na Tabela 18, sendo que o deslocamento do algoritmo aleatório continua a manter-se elevado. A variante Vivaldi+h continua a ser a que tem um desempenho mais próximo do algoritmo "melhor", a ultrapassar os 92%, seguida da variante Vivaldi, que ultrapassa os 80%.

	Tempo médio (emissão-recepção)	%
Knowlege - Best	0,632382597	**100**
Knowlege - Worst	1,348936334	**0**
Aleatório	0,933160424	**58,02438656**
GNP	0,795518611	**77,2332477**
Vivaldi (0-15000)	0,802696124	**76,23157654**
Vivaldi (3000-15000)	0,777391902	**79,76295457**
Vivaldi (5000-15000)	0,771817977	**80,54083417**
Vivaldi+h (0-15000)	0,731350804	**86,18830637**
Vivaldi+h (3000-15000)	0,694716039	**91,30093964**
Vivaldi+h (5000-15000)	0,686714835	**92,4175627**

Tabela 19 – Resultados para rede Orbis (2.000 nós), 2.000 nós LiveFeeds e *fanout factor* = 10.

Os resultados apresentados na Tabela 20 apresentam uma manutenção no desempenho do algoritmo GNP em relação aos resultados apresentados anteriormente, sendo que o deslocamento do algoritmo aleatório continua a manter-se elevado. A variante Vivaldi+h continua a ser a que tem um desempenho mais próximo do algoritmo "melhor", a ultrapassar os 95%, seguida da variante Vivaldi, que se aproxima dos 82%.

	Tempo médio (emissão-recepção)	%
Knowlege - Best	1,110788559	**100**
Knowlege - Worst	3,421408434	**0**
Aleatório	2,074081025	**58,31021464**
GNP	1,623087125	**77,82852249**
Vivaldi (0-75000)	1,635218123	**77,30351195**
Vivaldi (15000-75000)	1,551142245	**80,94218392**
Vivaldi (25000-75000)	1,52739245	**81,97003776**
Vivaldi+h (0-75000)	1,384523214	**88,15319394**
Vivaldi+h (15000-75000)	1,258603065	**93,60282028**
Vivaldi+h (25000-75000)	1,220584576	**95,24820077**

Tabela 20 – Resultados para rede Orbis (10.000 nós), 2.000 nós LiveFeeds e *fanout factor* = 3.

Os resultados apresentados na Tabela 21 apresentam uma manutenção no desempenho do algoritmo GNP em relação aos resultados apresentados anteriormente, sendo que o deslocamento do algoritmo aleatório continua a manter-se elevado. A variante Vivaldi+h continua a ser a que tem um desempenho mais próximo do algoritmo "melhor", mas inferior ao que se verificou na tabela anterior. A variante Vivaldi apenas ultrapassa ligeiramente o desempenho do algoritmo GNP.

	Tempo médio (emissão-recepção)	%
Knowlege - Best	0,659918986	**100**
Knowlege - Worst	1,689468814	**0**
Aleatório	1,068671578	**60,29793013**
GNP	0,890101266	**77,64243425**
Vivaldi (0-75000)	0,918729575	**74,86177143**
Vivaldi (15000-75000)	0,888070185	**77,83971275**
Vivaldi (25000-75000)	0,881870182	**78,44191802**
Vivaldi+h (0-75000)	0,812181881	**85,2107309**
Vivaldi+h (15000-75000)	0,763984837	**89,89210152**
Vivaldi+h (25000-75000)	0,753930007	**90,86872547**

Tabela 21 – Resultados para rede Orbis (10.000 nós), 2.000 nós LiveFeeds e *fanout factor* = 10.

Em relação aos ensaios para a rede Orbis com 2.000 nós internos verificou-se que, ao contrário do que aconteceu para a rede Orbis com 500 nós internos e à semelhança do que aconteceu com a rede euclidiana, o desempenho do algoritmo aleatório ficou situado muito acima dos 50%, neste caso aproximadamente entre os 56 e os 60%. Outra verificação a reter é o desempenho quase sempre constante do algoritmo GNP, aproximadamente entre os 76% e os 78%. Verificou-se ainda que o desempenho da variante simples (sem altura) do algoritmo Vivaldi sempre ligeiramente melhor que o desempenho do algoritmo GNP, se não for contado o período de aprendizagem do Vivaldi. A variante Vivaldi+h foi a que teve um desempenho mais aproximado ao do algoritmo "melhor", chegando mesmo a ultrapassar os 95%.

De uma forma geral, os resultados apontam para uma superioridade em termos de desempenho relativamente ao tempo médio que decorre desde que é emitida até à recepção em cada nó para o algoritmo Vivaldi, nomeadamente na sua variante Vivaldi+h. O algoritmo GNP apenas se mostrou superior na rede euclidiana.

6.2 Resultados relativos à carga gerada nos nós

Em seguida são apresentados os resultados relativos à forma como a escolha de nós de acordo com critérios não aleatórios afecta a carga na generalidade dos nós. Para isso foram produzidos gráficos de barras com uma representação da distribuição do número de mensagens enviadas por cada nó, sendo ainda apresentado o desvio padrão de cada amostragem. De modo a facilitar a comparação dos gráficos relativos a simulações com diferentes durações e número global de mensagens enviadas os valores apresentados são divididos pelo número total de mensagens criadas na simulação. Assim, no eixo horizontal

encontram-se os nós e no eixo vertical o número de mensagens envidadas por cada nó, dividido do número total de mensagens criadas na simulação respectiva. De modo a facilitar a leitura e compreensão dos gráficos, apenas são apresentados os gráficos relativos às simulações com 500 nós LiveFeeds, para cada um dos algoritmos em estudo. Os resultados obtidos para 2.000 e 10.000 nós LiveFeeds são relativamente semelhantes, pelo que não existe necessidade real de apresentar os gráficos, sendo feita ainda assim uma análise global dos resultados.

6.2.1 Rede Euclidiana

Nesta subsecção apresentam-se os resultados relativos à forma como a escolha de nós de acordo com critérios não aleatórios afecta a carga na generalidade dos nós para uma rede euclidiana a duas dimensões. São apenas apresentados os resultados para 500 nós LiveFeeds e *fanout factor* = 3.

No gráfico da Figura 28 é possível verificar que o número médio de mensagens enviadas sobre o total de mensagens criadas é geralmente próximo de 1, com um desvio padrão baixo, que reflecte claramente o carácter aleatório da amostragem, verificando-se assim que a carga se distribui de forma equivalente pela generalidade dos nós.

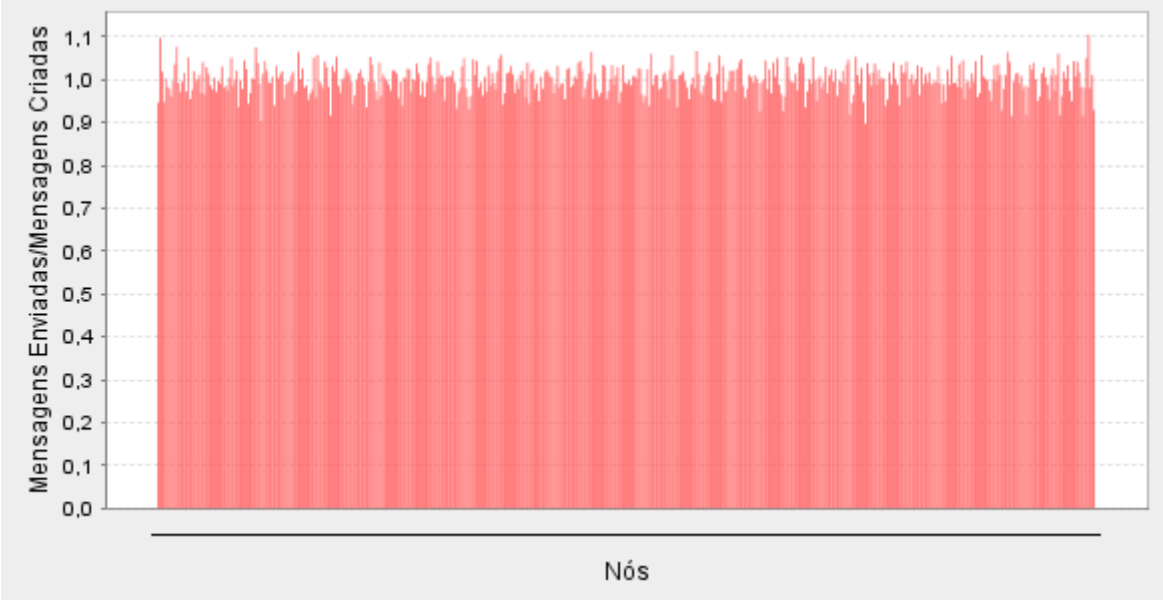


Figura 28 - Número de mensagens enviadas por nó: Algoritmo aleatório, 500 nós LiveFeeds, *fanout factor* = 3, desvio padrão da amostra = 0,032633322846425

No gráfico da Figura 29 (para o algoritmo "melhor") é possível verificar que a distribuição de carga pelos diferentes nós já não é de forma alguma tão equivalente como no grá-

fico anterior, neste caso existem nós que distribuem quase três vezes mais carga que a média e nós que por oposição distribuem menos de um quarto da média, daí o desvio padrão ser mais de dez vezes superior ao verificado para o caso do algoritmo aleatório.

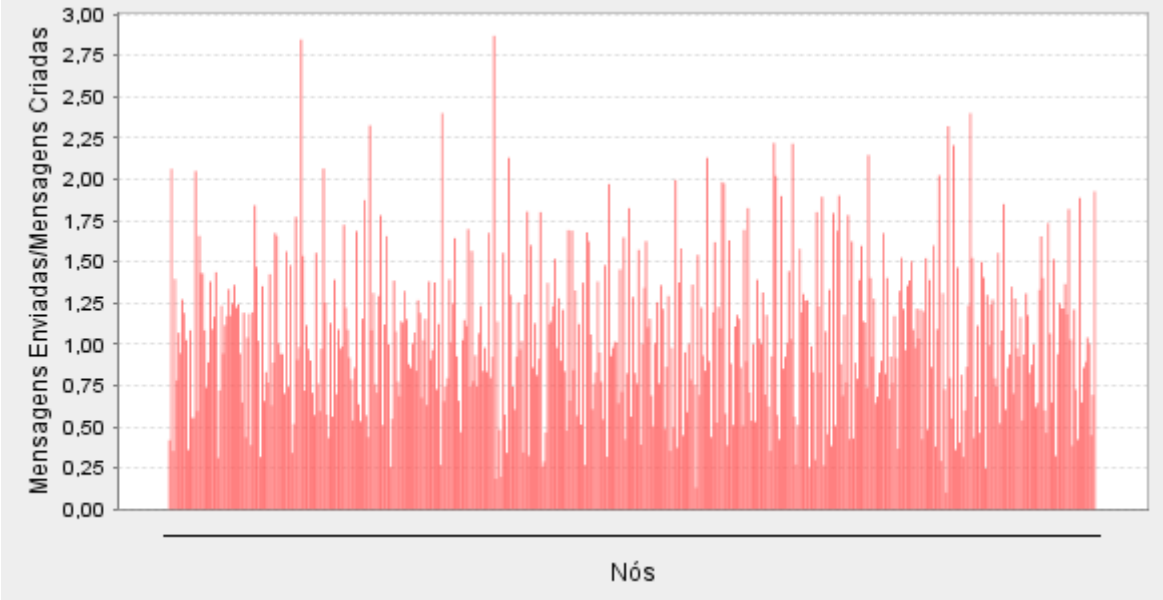


Figura 29 - Número de mensagens enviadas por nó: algoritmo "melhor", 500 nós LiveFeeds, *fanout factor* = 3, desvio padrão da amostra = 0,46978379536123

No gráfico da Figura 30 (para o algoritmo "pior") é possível verificar que a distribuição de carga pelos diferentes nós extremamente desigual, neste caso existem nós que distribuem três vezes mais carga que a média e nós que por oposição nunca distribuem mensagens, daí o desvio padrão ser quase trinta vezes superior ao verificado para o caso do algoritmo aleatório.

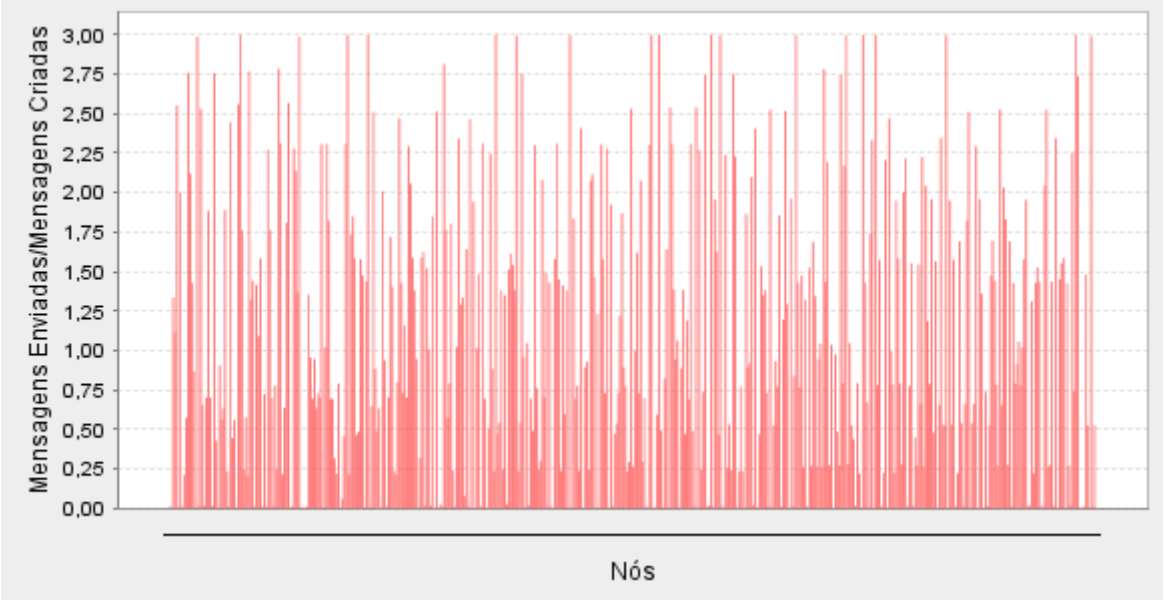


Figura 30 - Número de mensagens enviadas por nó: algoritmo "pior", 500 nós LiveFeeds, *fanout factor* = 3, desvio padrão da amostra = 0,90493189824649

No gráfico da Figura 31 (para o algoritmo GNP) é possível verificar que a distribuição de carga pelos diferentes nós é equivalente ao que se verifica no gráfico da Figura 29, o que se fica a dever ao facto de o algoritmo GNP ter um desempenho quase equivalente ao algoritmo "melhor". Neste caso existem nós que distribuem quase três vezes mais carga que a média e nós que por oposição distribuem menos de um quarto da média, daí o desvio padrão ser mais de dez vezes superior ao verificado para o caso do algoritmo aleatório.

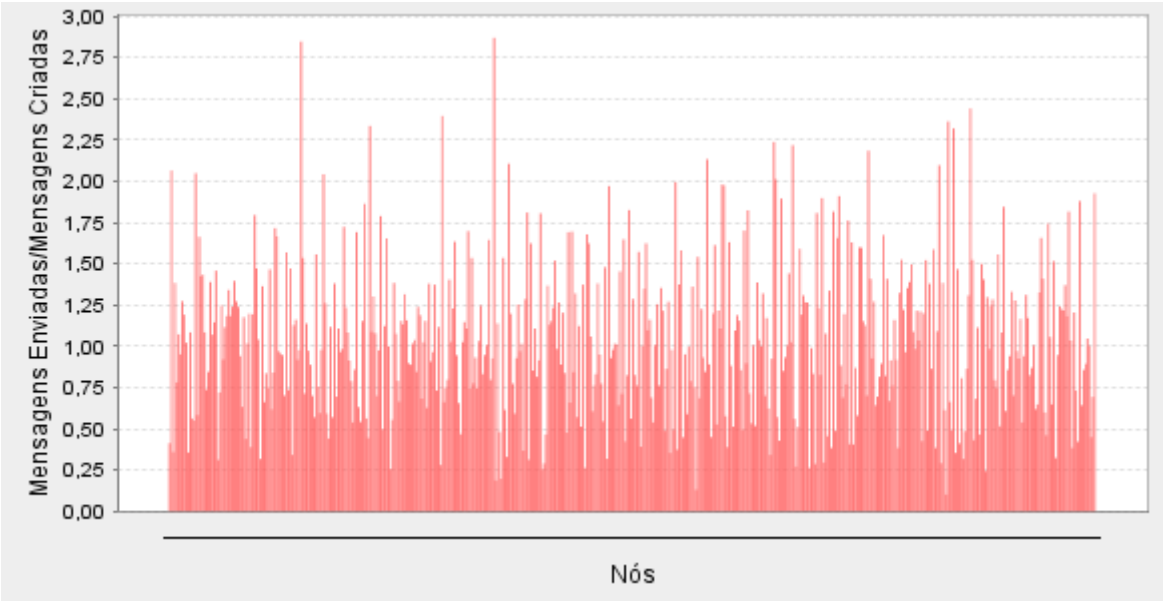


Figura 31 - Número de mensagens enviadas por nó: algoritmo GNP, 500 nós LiveFeeds, *fanout factor* = 3, desvio padrão da amostra = 0,4707719008777

No gráfico da Figura 32 (para o algoritmo Vivaldi) é possível verificar que a distribuição de carga pelos diferentes nós é quase equivalente às distribuições verificadas para os algoritmos GNP e "melhor", o que é visível no seu desvio padrão, que é aproximado.

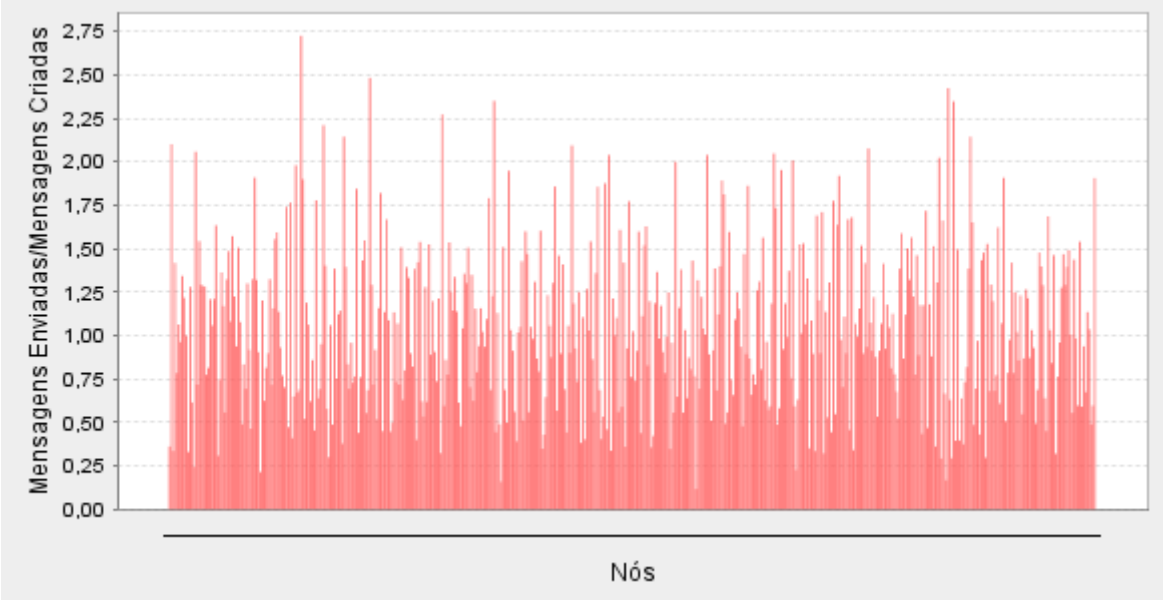


Figura 32 - Número de mensagens enviadas por nó: algoritmo Vivaldi, 500 nós LiveFeeds, *fanout factor* = 3, desvio padrão da amostra = 0,46830841159287

No gráfico da Figura 33 (para o algoritmo Vivaldi+h) é possível verificar que a distribuição de carga pelos diferentes nós é quase equivalente às distribuições verificadas para os algoritmos GNP, "melhor" e para a variante simples do algoritmo Vivaldi, o que é visível no seu desvio padrão, que é aproximado.

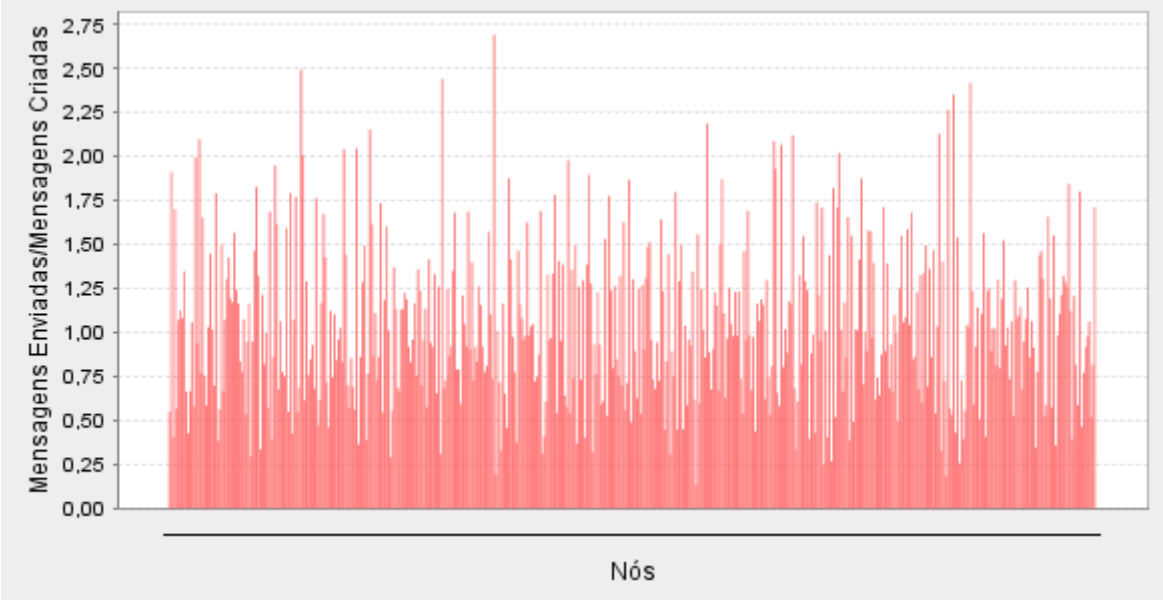


Figura 33 - Número de mensagens enviadas por nó: algoritmo Vivaldi+h, 500 nós LiveFeeds, *fanout factor* = 3, desvio padrão da amostra = 0,45335041341126

6.2.2 Rede Orbis – 500 nós

Nesta subsecção apresentam-se os resultados relativos à forma como a escolha de nós de acordo com critérios não aleatórios afecta a carga na generalidade dos nós para uma rede gerada pelo gerador de topologias de rede Orbis, com 500 nós internos. São apenas apresentados os resultados para 500 nós LiveFeeds e *fanout factor* = 10.

No gráfico da Figura 34 é possível verificar que o número médio de mensagens enviadas sobre o total de mensagens criadas é geralmente próximo de 1, com um desvio padrão baixo, que reflecte claramente o carácter aleatório da amostragem, verificando-se assim que a carga se distribui de forma equivalente pela generalidade dos nós. Apesar de o desvio padrão ser baixo, verifica-se que ainda assim é superior ao que se verifica para *fanout factor* = 3.

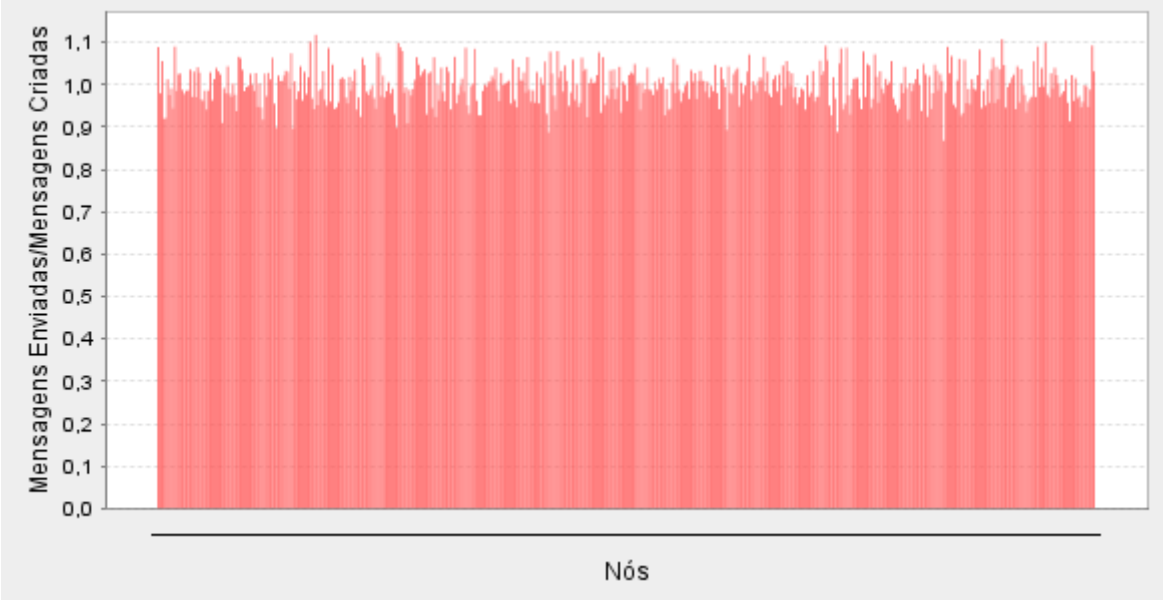


Figura 34 - Número de mensagens enviadas por nó: algoritmo aleatório, 500 nós LiveFeeds, *fanout factor* = 10, desvio padrão da amostra = 0,04280684057485

No gráfico da Figura 35 (para o algoritmo "melhor") é possível verificar que a distribuição de carga pelos diferentes nós é bastante desigual, neste caso existem nós que distribuem mais de quatro vezes mais carga que a média e nós que por oposição distribuem uma carga quase nula, daí o desvio padrão ser mais de vinte e três vezes superior ao verificado para o caso do algoritmo aleatório.

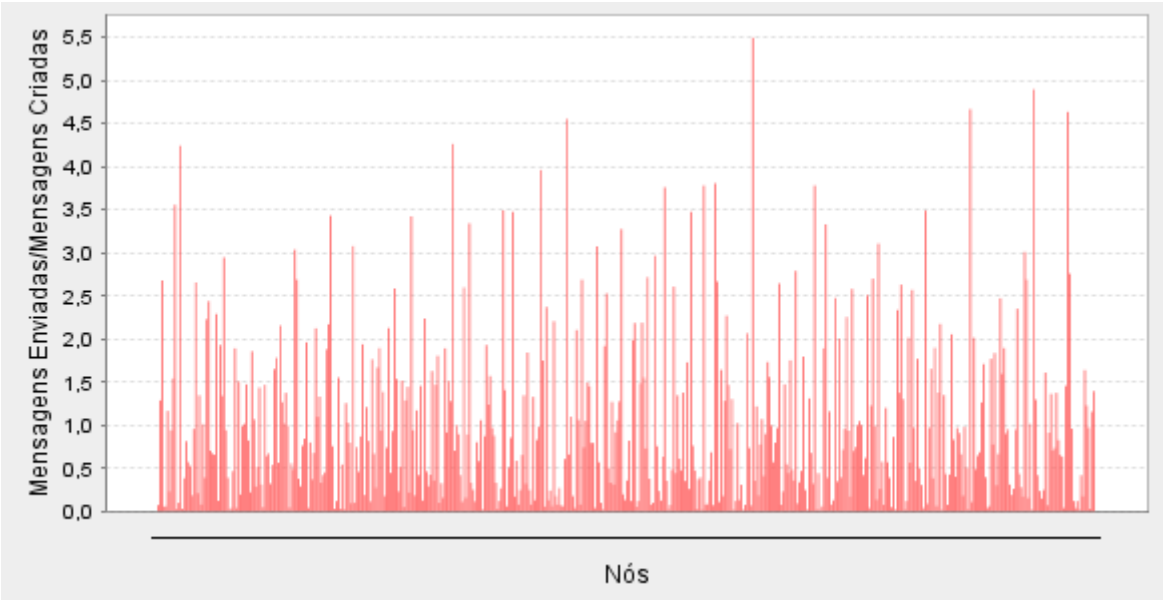


Figura 35 - Número de mensagens enviadas por nó: algoritmo "melhor", 500 nós LiveFeeds, *fanout factor* = 10, desvio padrão da amostra = 0,9875513606410

No gráfico da Figura 36 (para o algoritmo "pior") é possível verificar que a distribuição de carga pelos diferentes nós extremamente desigual, ainda mais que no caso do algoritmo "melhor", neste caso existem nós que distribuem oito vezes mais carga que a média e nós que por oposição numerosos nós que nunca distribuem mensagens, daí o desvio padrão ser quase quarenta vezes superior ao verificado para o caso do algoritmo aleatório.

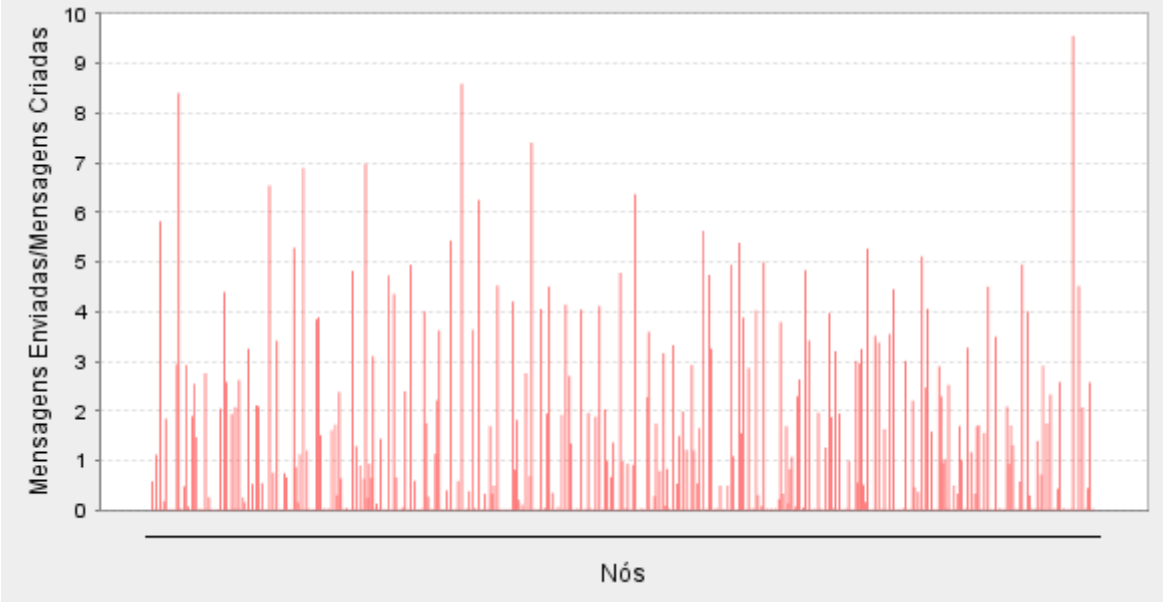


Figura 36 - Número de mensagens enviadas por nó: algoritmo "pior", 500 nós LiveFeeds, *fanout factor* = 10, desvio padrão da amostra = 1,64896293574355

No gráfico da Figura 37 (para o algoritmo GNP) é possível verificar que a distribuição de carga pelos diferentes nós é bastante desigual, no entanto não é uma diferença tão acentuada como no caso do algoritmo "melhor", sendo que o desvio padrão para este caso é cerca de metade do desvio padrão para o caso do algoritmo "melhor" sendo que nenhum nó a uma carga superior a três vezes a média, existindo no entanto nós que estão sujeitos a menos de um quarto da carga média.

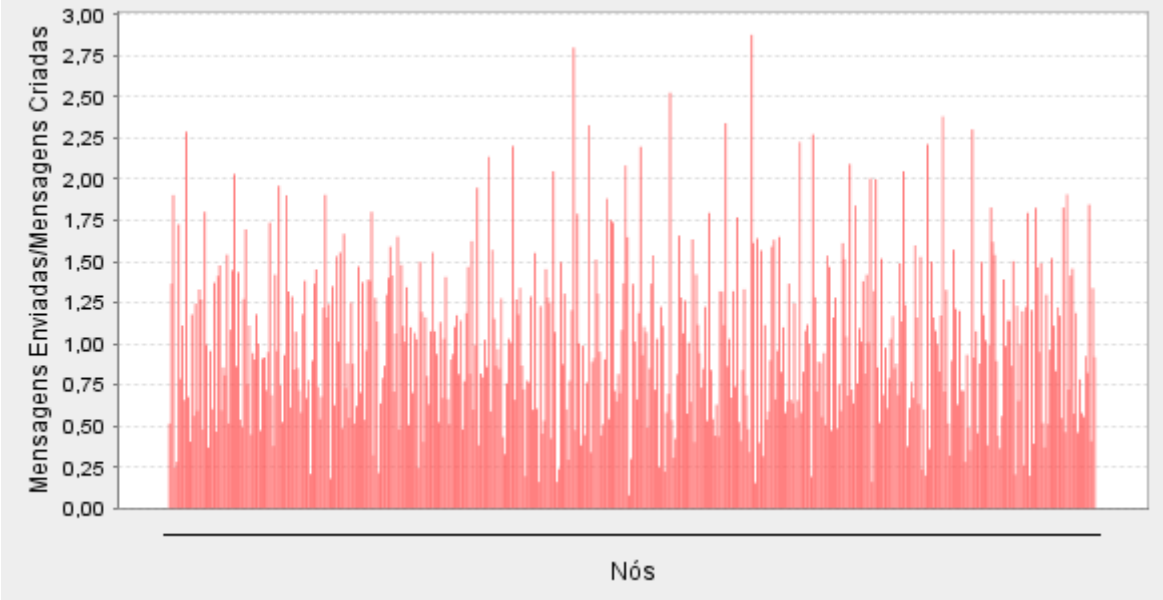


Figura 37 - Número de mensagens enviadas por nó: algoritmo GNP, 500 nós LiveFeeds, *fanout factor* = 10, desvio padrão da amostra = 0,49314819859349

No gráfico da Figura 38 (para o algoritmo Vivaldi) é possível verificar que a distribuição de carga pelos diferentes nós, apesar de ter uma desigualdade acentuada, esta é menos acentuada que no caso do algoritmo GNP, o que é reflectido por um menor desvio padrão.

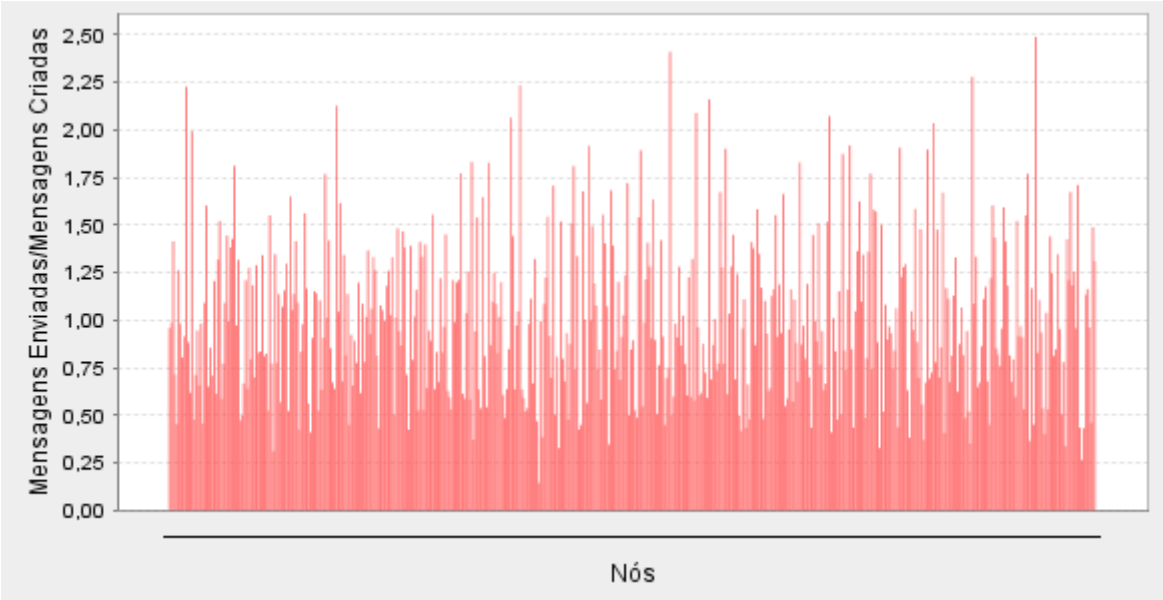


Figura 38 - Número de mensagens enviadas por nó: algoritmo Vivaldi, 500 nós LiveFeeds, *fanout factor* = 10, desvio padrão da amostra = 0,41834954737569

No gráfico da Figura 39 (para o algoritmo Vivaldi+h) é possível verificar que a distribuição de carga pelos diferentes quase tão desigual quanto para o caso do algoritmo "melhor", sendo que o desvio padrão para este caso é bastante aproximado ao do algorit-

mo “melhor”, ainda assim, não são muitos os casos de nós que emitam mais de quatro vezes a média de mensagens, não existindo neste caso nó que emitam virtualmente nenhuma mensagem, o que acontece no caso do algoritmo “melhor”.

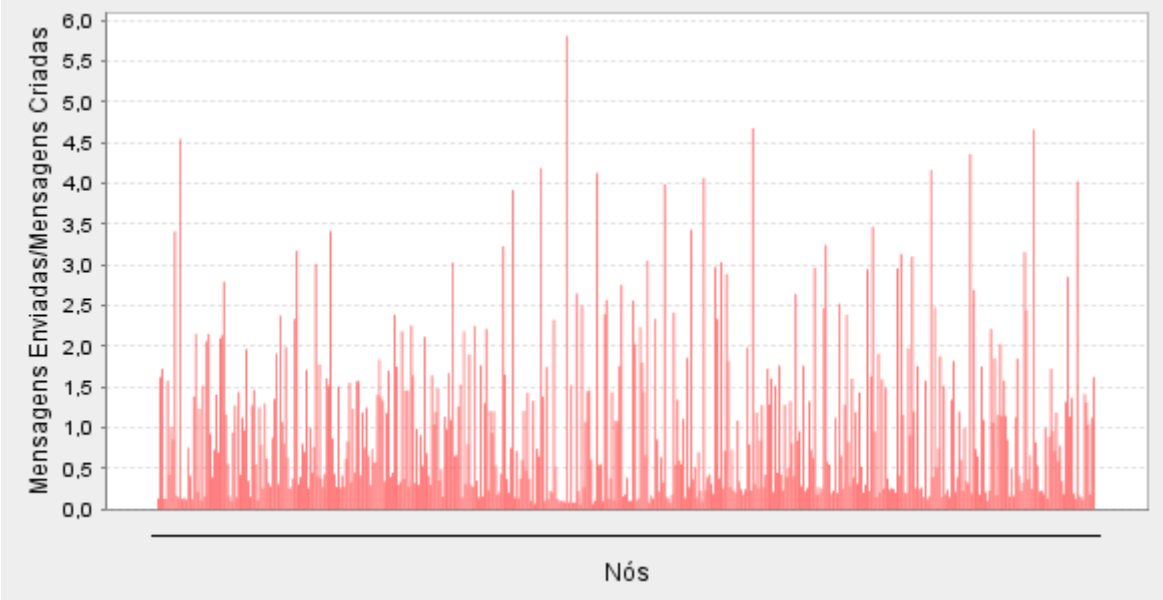


Figura 39 - Número de mensagens enviadas por nó: algoritmo Vivaldi+h, 500 nós LiveFeeds, *fanout factor* = 10, desvio padrão da amostra = 0,96680541617051

Como síntese desta secção é possível dizer que se verifica nos resultados apresentados e nos resultados relativos aos restantes ensaios que a carga fica sempre bem distribuída quando se trata do algoritmo aleatório, como seria de esperar. Este facto torna-se ainda mais evidente quando o número de ensaios é superior. É ainda de salientar que quanto mais se aproxima o desempenho de um dado algoritmo em relação ao algoritmo “melhor”, maior a desigualdade na distribuição de carga entre os nós, com excepção do algoritmo “pior”, que consegue ser extremamente desigual na distribuição de carga, segundo o que se pode visualizar nos gráficos, para o caso do algoritmo “pior” existe uma maior desigualdade do que no caso do algoritmo “melhor” o que significa que existem nós que são mais vezes escolhidos como piores e outros nós que nunca são escolhidos como piores, mais vezes. Neste caso a comparação poderia ser efectuada com os restantes algoritmos (GNP e Vivaldi), se estes também estivessem a funcionar de modo a escolher os piores em cada momento segundo as coordenadas virtuais estimadas. Também se verifica que quanto maior o *fanout factor* maior a desigualdade, quando são comparados algoritmos equivalentes, isto deve-se ao facto de potencialmente ser possível enviar um número superior de mensagens de cada vez, e o número de escolhas a efectuar durante uma simulação ser menor, daí decorre que no limite, um nó que seja sempre nó intermédio e

cujo *fanout factor* = 3, tenha uma carga 3 vezes superior à média, enquanto que para um *fanout factor* = 10, este tenha uma carga 10 vezes superior à média.

Um resumo dos resultados obtidos relativos à carga nos diferentes nós encontra-se na tabela seguinte.

Algoritmo	Tipo de rede	N.º de nós	*Fanout factor*	Desvio padrão
Aleatório	Euclidiana	500	3	0,032633322846425
"Melhor"	Euclidiana	500	3	0,46978379536123
"Pior"	Euclidiana	500	3	0,90493189824649
GNP	Euclidiana	500	3	0,4707719008777
Vivaldi	Euclidiana	500	3	0,46830841159287
Vivaldi+h	Euclidiana	500	3	0,45335041341126
Aleatório	Orbis (500 nós)	500	10	0,04280684057485
"Melhor"	Orbis (500 nós)	500	10	0,9875513606410
"Pior"	Orbis (500 nós)	500	10	1,64896293574355
GNP	Orbis (500 nós)	500	10	0,49314819859349
Vivaldi	Orbis (500 nós)	500	10	0,41834954737569
Vivaldi+h	Orbis (500 nós)	500	10	0,96680541617051

Tabela 22 - Resumo dos resultados relativos à carga nos diferentes nós.

6.3 Análise

O comportamento do algoritmo GNP na rede Euclidiana em duas dimensões é muito próximo do algoritmo "melhor" (quase 100%), sendo simultaneamente muito melhor do que as duas variações do algoritmo Vivaldi, para todas as configurações testadas. No entanto, esse desempenho cai quando é usada a rede gerada a partir do gerador de topologias Orbis. Neste caso, de uma forma geral o algoritmo Vivaldi passa a ter um desempenho igual ou superior ao desempenho do algoritmo GNP, sendo que a variante do algoritmo Vivaldi que usa altura tem geralmente um desempenho superior a 90%, muito superior ao desempenho do algoritmo GNP.

De uma forma geral os algoritmos comportam-se melhor em termos relativos quando o *fanout factor* é menor, nomeadamente o algoritmo Vivaldi. No caso do algoritmo GNP o seu desempenho é bastante estático, mantendo-se em média em cerca de 75% para a rede gerada pelo gerador de topologias Orbis e em cerca de 100% para a rede euclidiana. Em

termos absolutos acontece precisamente o contrário devido ao facto de com um *fanout factor* superior ser possível propagar as mensagens muito mais rapidamente na rede, no entanto, um *fanout factor* superior tem um efeito negativo na carga sobre os nós, aumentando a desigualdade entre estes. É necessário ainda salientar que a desigualdade de carga entre os nós é tanto maior quanto o seu maior for o seu desempenho. Em relação ao algoritmo aleatório, como seria de esperar, este é o que apresenta uma menor diferença de carga entre os diferentes nós, sendo tal mais evidente quando o número de ensaios é superior, sendo que o desvio padrão para ensaios para o algoritmo aleatório com 10.000 nós e *fanout factor* = 3, nos quais são emitidas 50.000 mensagens, chega a ser da ordem das milésimas. O facto de na maioria dos ensaios, nomeadamente nos ensaios da rede euclidiana e da rede Orbis com 2.000 nós internos, o desempenho do algoritmo aleatório se ter afastado dos 50% leva a crer que nestes casos, os algoritmos ditos "melhor" e "pior" possivelmente não se encontrarão equidistantes do algoritmo absolutamente melhor e do algoritmo absolutamente pior. No entanto, os resultados relativos aos algoritmos em estudo aparentam estar consistentes com os ensaios em que o algoritmo aleatório se aproximou realmente dos 50%, tal como na hipótese descrita no capítulo anterior.

Os ensaios foram efectuados ao longo de um número significativo de horas de simulação, sendo que os resultados que aqui se apresentam são uma fracção de todas as horas que foram dispendidas, nomeadamente em despistagem de erros, o que obrigou inclusive a repetições de muitas simulações.

O tempo total de simulação significativo foi de 156 horas, 53 minutos e 34 segundos, sendo que este tempo corresponde apenas a tempo de processamento no simulador e exclui o tempo de recolha e organização de resultados, bem como o tempo de simulação em testes durante o desenvolvimento.

7. Conclusões e trabalho futuro

Após a apresentação e análise dos resultados relativos à parte prática desta dissertação resta reflectir sobre o significado desses mesmo resultados, tendo em consideração todas as limitações presentes no ambiente de simulação e daí obter as conclusões que se é possível tirar e a perspectivas de trabalho a desenvolver no futuro.

Em primeiro lugar, é possível concluir que o uso de sistemas que tiram partido das latências entre os diversos nós para obter uma aproximação de uma matriz de distâncias virtuais tem verdadeiramente um efeito positivo na redução do tempo médio que decorre desde que uma mensagem é enviada por um nó emissor, até a mesma chegar ao nó receptor. Esta conclusão é válida no contexto deste estudo e com todas as suas condicionantes.

Verificou-se que o algoritmo GNP tem um melhor desempenho na rede euclidiana que ambas as variantes do algoritmo Vivaldi, que apresentam um desempenho equilibrado entre si neste tipo de rede, visto que a variante Vivaldi+h faz uso da altura por forma a modelar a distância entre a periferia e o núcleo de rede, o que não existe na rede euclidiana a duas dimensões, pelo que, como seria de esperar, a utilização da noção de altura numa rede deste género não tem qualquer efeito. Neste tipo de rede, o desempenho do algoritmo GNP é virtualmente igual ao desempenho do algoritmo "melhor", no entanto, na rede gerada pelo gerador de topologias de rede Orbis, o algoritmo Vivaldi, especialmente a sua evolução que possui a noção de altura (Vivaldi+h), tem um melhor desempenho que o algoritmo GNP, aproximando-se de forma convincente do algoritmo "melhor". O facto de o algoritmo GNP ter um desempenho tão diverso em ambos os tipos de rede poderá ter a ver com a forma como o algoritmo GNP opera no contexto de ambos os tipos de rede. Numa rede gerada pelo gerador de topologias de rede Orbis, os nós encontram-se mais dispersos pela periferia da rede, na rede euclidiana não existe noção de periferia ou núcleo da rede. Os *landmarks* do GNP são colocados da mesma forma que os restantes nós na rede Orbis, isto é, na periferia da rede. No caso da rede gerada pelo gerador Orbis, existe um maior afastamento entre a generalidade dos nós, em relação ao que acontece com os nós da rede euclidiana, assentando no facto de que nesta não existe um núcleo de

rede, sendo que os nós, incluindo os *landmarks* GNP são colocados ao mesmo nível no plano euclidiano. Uma propriedade que se verificou em anteriores estudos relativos a algoritmos que usam *landmarks*, como o algoritmo GNP, é que este tipo de algoritmos comporta-se melhor no caso de existir um número maior de *landmarks* e caso estes estejam colocados próximo do maior número de outros nós, possível, o que melhora a sua precisão. No caso da rede Orbis, como existe um maior afastamento entre os nós e necessariamente entre os *landmarks* GNP, é possível que, tendo um número superior de *landmarks* o desempenho do algoritmo GNP também fosse melhor nos ensaios usando a rede Orbis. No entanto, um número maior de *landmarks* corresponde a uma maior carga para os mesmos e para a rede em geral, pelo que não seria desejável aumentar o número de *landmarks* apenas em função de uma possível melhoria do desempenho do algoritmo GNP, mas também por motivos infraestruturais.

Na rede gerada pelo gerador de topologias Orbis, o algoritmo Vivaldi, na sua versão original, isto é, sem noção de altura, comporta-se de modo que se pode considerar ligeiramente superior ao comportamento do GNP, mas apenas após o seu período de aprendizagem estar completo. Já a evolução do algoritmo Vivaldi que usa a noção de altura, tem um comportamento que é o que mais se aproxima do algoritmo "melhor", distanciando-se claramente do algoritmo GNP, mas também da variante original do algoritmo Vivaldi. Isto permite concluir que no caso das redes Orbis, que possuem toda uma estrutura que separa a periferia e o núcleo de rede, o facto de ser usada uma noção de altura que visa medir e incluir nas coordenadas virtuais essa mesma separação permite ao algoritmo Vivaldi+h destacar-se dos demais pela positiva, neste tipo de rede. Mais uma vez, a estrutura característica da rede Orbis pode contribuir também para explicar o facto de o algoritmo GNP ter um desempenho perfeito em redes euclidianas e um desempenho não tão bom em redes Orbis, sendo de notar que o algoritmo GNP não possui qualquer noção de altura.

Verificou-se que por serem efectuadas escolhas de nós baseadas nas latências e por sua vez nos algoritmos que estimam coordenadas virtuais baseadas na latência entre os nós, são criados desequilíbrios de carga entre os diversos nós presentes no sistema, sendo que esse desequilíbrio de carga é tanto maior quanto melhor o desempenho do algoritmo em relação ao algoritmo aleatório, que distribui a carga de forma equilibrada. Este desequilíbrio pode, no limite, fazer com que alguns nós sejam sempre nós intermédios da árvore de difusão de mensagens, tendo uma carga elevada, enquanto que outros nós nunca o são,

tendo uma carga nula. Os nós que por serem demasiadamente escolhidos ficam sujeitos a uma carga elevada podem constituir-se como pontos críticos no sistema. Se a variação de carga se reflectir na latência que é usada para calcular as coordenadas virtuais, um algoritmo dinâmico, como o Vivaldi, poderia reajustar as coordenadas dos nós e assim evitar um maior desequilíbrio. O simulador usado não permite fazer reflectir variações de carga na latência, pelo que não foi possível verificar esse efeito que seria interessante de verificar numa futura evolução do simulador ou noutro ambiente de teste. Poderiam também ser implementados mecanismos que permitissem diminuir a variação de carga, ao não escolher sempre o melhor mas de entre *n* melhores, ou em alternativa efectuar essa escolha, por vezes, de forma aleatória.

Não é possível perante as condicionantes do estudo, nomeadamente do ambiente simplificado de simulação e do tempo disponível, obter conclusões definitivas, mas perante os dados obtidos, de entre os algoritmos em estudo, o algoritmo que possivelmente melhor atingirá o objectivo de minimizar o tempo médio que decorre desde que uma mensagem é enviada por um nó emissor até chegar ao nó receptor é o algoritmo Vivaldi+h. Apesar de na rede euclidiana o algoritmo GNP ter um desempenho superior, a superioridade do algoritmo Vivaldi+h é por demais evidente na rede gerada pelo gerador Orbis, sendo considerado que este tipo de rede se aproxima mais de uma rede real, que a rede euclidiana.

Em termos de complexidade, o algoritmo GNP possui uma maior complexidade quando comparado com o algoritmo Vivaldi. O facto de um nó, para tomar conhecimento das suas coordenadas, ter de efectuar medições a todos os *landmarks* presentes no sistema, faz com que existam dificuldades práticas, quer a nível de rede, quer a nível dos nós na generalidade, mas também para os próprios *landmarks*. A nível de rede, a quantidade de pedidos que são direccionados aos *landmarks* teria certamente um peso significativo. Em relação aos nós participantes, estes podem não estar bem ligados, e por isso podem ter alguma dificuldade em efectuar as medições e obter as respostas, para poder efectuar o cálculo de coordenadas. Num sistema em larga escala, os *landmarks* estariam permanentemente sobre uma enorme carga, devido às grandes quantidades de pedidos que receberiam, assim, teriam forçosamente de ter uma grande capacidade de resposta, o que para além do simples facto de estes já serem só por si, uma infra-estrutura complementar, tornaria essa infra-estrutura algo difícil de conceber e manter de um modo prático. O algoritmo Vivaldi, não possui este tipo de problemas, começando pelo facto de não possuir qualquer infra-estrutura especial, mas também, tendo em conta que as medições efectua-

das pelos nós, uns aos outros, estão inerentes à própria dinâmica de um algoritmo *peer-to-peer*.

Outro factor que pode ser tido em conta na escolha de um dos algoritmos é a instabilidade de rede, também designada por "churn". Este factor não é simulado pelo simulador, no entanto, é possível prever que o algoritmo Vivaldi, mais dinâmico e em permanente actualização, se adaptaria melhor a instabilidade na rede do que o algoritmo GNP, pelo menos após o seu período de aprendizagem. Este último é um algoritmo estático, sendo que apenas uma variante do GNP no qual os *landmarks* actualizassem as suas coordenadas periodicamente, e em que os nós participantes efectuassem medições periódicas aos *landmarks*, recalculando as suas próprias coordenadas, se poderia adaptar a condições de *churn* na rede. No entanto, uma variante do algoritmo GNP com essa natureza mais dinâmica teria os problemas descritos anteriormente mas de uma forma aumentada, o que possivelmente tornaria impossível o seu uso.

Para além do *churn* existe ainda um outro factor que poderia provocar alguma instabilidade nas coordenadas do algoritmo Vivaldi. Esse factor prende-se com a entrada e saída de nós do sistema, problema que não é contemplado nesta dissertação, pois não existe gestão de filiação na versão do sistema LiveFeeds usada. A entrada de um novo nó no sistema cujas coordenadas virtuais estarão, por exemplo, na origem do referencial, poderiam provocar alguma instabilidade e incorrecção nas coordenadas dos nós que comunicassem com este. No entanto, este factor pode ser minimizado no sentido em que os nós Vivaldi, além de possuírem as coordenadas, possuem também um erro associado que é tido em conta no reajustamento de coordenadas dos nós. Um novo nó a entrar no sistema, teria necessariamente que ter um erro significativo associado, o que teoricamente minimizaria as implicações das suas coordenadas virtuais nas coordenadas virtuais dos nós com os quais este comunicasse.

Existe no entanto uma fragilidade no algoritmo Vivaldi que poderia pôr em causa a sua estabilização. Essa fragilidade prende-se com o seu período de aprendizagem, no qual, a generalidade dos nós ainda não possui coordenadas estáveis. Neste caso, o *churn* poderá impedir ou pelo menos tornar mais morosa a estabilização de coordenadas no algoritmo Vivali, o que agravado do facto de num ambiente real existirem entradas e saídas de nós poderá ter efeitos negativos no desempenho deste algoritmo. Não foi possível através deste estudo averiguar qual a gravidade destes factores para o desempenho deste algoritmo e

também do algoritmo GNP, mas tal estudo teria todo o interesse numa possível continuação futura deste trabalho.

O facto de a parte prática deste estudo ter sido efectuada num simulador de rede, que não permite ter em conta alguns aspectos de baixo nível de rede e de toda a dinâmica que se verifica na Internet, faz com que não seja possível afirmar que as observações verificadas se possam também verificar de forma semelhante numa situação em que o sistema LiveFeeds estivesse disseminado na Internet, no entanto, são dadas indicações no sentido de que os algoritmos que fornecem uma função de distância na rede, baseada em latências, poderão ter uma contribuição positiva para minimizar o tempo médio que decorre desde que uma mensagem é enviada por um nó emissor até chegar ao nó receptor, no contexto de um sistema como o LiveFeeds.

Fora do contexto específico do sistema LiveFeeds, isto é, no contexto de outro algoritmo ou sistema de difusão de conteúdos, baseado em *peer-to-peer*, seria simples inferir que as conclusões retiradas desta dissertação se aplicariam de uma forma equivalente, principalmente a nível relativo, visto que em termos absolutos, os ganhos que se obteriam noutro tipo de sistema poderiam ser diversos, devido principalmente às características específicas do algoritmo de difusão em árvore, aplicando uma divisão por identificadores, o que influência, neste caso, o leque de escolhas possíveis dos algoritmos de que estimam matrizes de distâncias, em forma de coordenadas virtuais.

Como outro possível factor negativo conta-se o desequilíbrio de carga verificado entre os nós, no entanto, é possível que pequenas alterações na forma como são escolhidos os nós e a utilização de algoritmos mais dinâmicos como o Vivaldi, poderem minimizar essa situação ao efectuar uma melhor adaptação. No entanto, o desequilíbrio de carga, não deixaria de ser um efeito secundário induzido pela utilização deste tipo de algoritmos.

Para além do que já foi referido como possíveis experiências a efectuar no futuro, neste caso, numa tentativa de diminuição do desequilíbrio de carga entre os diferentes nós, poderá ser efectuado um estudo mais alargado, nomeadamente, usando outros algoritmos alternativos e recorrendo a um leque mais alargado de ensaios. Esses ensaios poderiam ser efectuados sobre uma evolução do simulador, que já tivesse em conta aspectos de mais baixo nível de rede, e sobretudo aspectos de instabilidade de rede que fossem o mais próximo possível do que se passa na Internet, incluindo alguns dos padrões de utilização de rede conhecidos, e que fazem com que haja, por exemplo, picos de utilização de rede

durante certos períodos do dia. Como modelo de rede, poderão também ser usados, para além de redes geradas por geradores de topologias, como o Orbis, ou da rede euclidiana simples, conjuntos de dados reais que estejam disponíveis em domínio público, isto é, obtidos a partir de ferramentas que permitem obter matizes de latências entre os nós, o que poderia dar aos resultados um carácter mais próximo do que seria espectável na realidade. Seria ainda desejável libertar o algoritmo Vivaldi do seu tempo de aprendizagem, e assim torná-lo mais estável perante situações de instabilidade na rede ou no caso de existirem muitas entradas e saídas. O modo de fazer isso poderá passar por obter coordenadas preliminares, consideradas mais próximas das coordenadas pós estabilização. Para isso poderão ser usados métodos complementares que ajudem o Vivaldi nessa tarefa, como por exemplo, um algoritmo como o GNP, através do qual se obtivessem as coordenadas iniciais dos nós, em vez de estas partirem simplesmente da origem do referencial. Essa aproximação poderia também ser efectuada por meio de aproximações através de correspondência com coordenadas estimadas no espaço físico, através de métodos de geolocalização IP. De uma forma simples o que se propõe, e que intuitivamente salta à vista após reflectir sobre os resultados obtidos, é que, possivelmente um algoritmo que tirasse partido de várias características positivas de vários algoritmos poderia constituir-se como uma solução mais consistente para a problemática aqui discutida, funcionando os diferentes algoritmos em complementaridade.

De um modo mais genérico, apesar de a utilização dos tipos de sistemas aqui referidos poder trazer benefícios, é necessário ter sempre em conta que existem outros factores a considerar além da sua precisão ou desempenho, nomeadamente em ambientes de teste mais próximos da realidade e principalmente na Internet, alguns dos quais já referidos anteriormente, mas também, e nomeadamente, a sua aplicabilidade, ou seja, se é possível aplicar cada um dos métodos a casos concretos e em tempo real, o seu impacto nas infra-estruturas existentes, designadamente a sobrecarga que estes causam para a rede e para outros sistemas que dela usufruem, a segurança, fiabilidade e vulnerabilidades que estes possuem e que é comum os autores relegarem para segundo plano. A fiabilidade, bem como a precisão dos métodos estudados pode ser gravemente afectada por factores que são comuns na Internet. Os métodos que se baseiam em medições de latência que fazem uso do protocolo ICMP, nomeadamente através dos conhecidos Ping e Traceroute, sofrem frequentemente bloqueios, através de *firewalls* por parte de alguns ISP ou administradores de domínios e/ou redes locais, como forma de prevenção contra ataques hostis ou sim-

plesmente como forma de redução de banda passante consumida. O uso de *proxies* é também um impedimento, nomeadamente para sistemas que não se baseiem em medições, por exemplo, os que se baseiam em informação contida em bases de dados, devido ao facto de o verdadeiro nó por trás dos pedidos ser desconhecido, podendo o seu domínio ser completamente diverso do domínio do *proxy* através do qual faz ligação. Apesar de estas fragilidades serem algo a ter em conta aquando do uso e implementação deste tipo de sistemas, estas escapam um pouco ao âmbito deste documento, devendo ser alvo de estudo em contextos futuros.

8. Bibliografia

[1] R. Bindal et al., “Improving Traffic Locality in BitTorrent via Biased Neighbor Selection”, in ICDCS ’06: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems, page 45, Lisbon, Portugal, Jul. 2006.

[2] “Reverse DNS Lookup”, URL: http://en.wikipedia.org/wiki/Reverse_DNS_lookup, Nov. 2007.

[3] “WHOIS”, URL: http://en.wikipedia.org/wiki/WHOIS, Jan. 2008.

[4] C. Davis et al., “RFC 1876 A Means for Expressing Location Information in the Domain Name System”, URL: http://tools.ietf.org/html/rfc1876, Jan 1996.

[5] V. Padmanabhan, L. Subramanian, “An investigation of geographic mapping techniques for Internet hosts”, in SIGCOMM, San Diego, CA, USA, Aug. 2001.

[6] J. A. Muir and P. C. van Oorschot, “Internet geolocation and evasion”, Technical Report TR-06-05, Carleton University – School of Computer Science, Apr. 2006.

[7] M. Freedman, M. Vutukuru, N. Feamster, and H. Balakrishnan, “Geographic locality of IP prefixes”, in Proc. ACM/SIGCOMM IMC, Berkeley, CA, USA, Oct. 2005.

[8] B. Gueye, S. Uhlig, S. Fdida, “Investigating the imprecision of IP Block-Based Geolocation”, in PAM - Passive and Active Measurement Conference, Louvain-la-neuve, Belgium, Apr. 2007.

[9] B. Gueye, A. Ziviani, M. Crovella, and S. Fdida, “Constraint-based geolocation of internet hosts”, in Proceedings of ACM Internet Measurement Conference (IMC ’04), Taormina, Sicily, Italy.

[10] E. Katz-Bassett, J. John, A. Krishnamurthy, D. Weatherall, T. Anderson, and Y. Chawathe, “Towards IP Geolocation Using Delay and Topology Measurements”, in Proceedings of ACM Internet Measurement Conference (IMC ’06), Rio de Janeiro, Brazil, Oct. 2006.

[11] E. Ng and H. Zhang, “Predicting Internet network distance with coordinates-based approaches”, in Proceedings of IEEE INFOCOM 2002, New York, NY, USA, Jun. 2002.

[12] E. Ng and H. Zhang, “A Network Positioning System for the Internet”, in Proceedings of USENIX, Boston, MA, USA, Jun. 2004.

[13] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti, “Lighthouses for Scalable Distributed Location”, In Intl. Workshop on Peer-To-Peer Systems, Berkeley, CA, USA, Feb. 2003.

[14] F. Dabek, R. Cox, F. Kaahoek, and R. Morris, “Vivaldi: A Decentralized Network Coordinate System”, in SIGCOMM ’04, Portland, Oregon, USA, Aug. 2004.

[15] Y. Shavitt and T. Tankel, “Big-bang simulation for embedding network distances in Euclidean space”, in Proc. of IEEE Infocom, San Francisco, CA, USA, Apr. 2003.

[16] P. Francis, S. Jamin, V. Paxson, L. Zhang, D. Gryniewicz, and Y. Jin, “An Architecture for a Global Internet Host Distance Estimation Service”, in Proceedings of IEEE Infocom, New York, NY, USA, Mar. 1999.

[17] P. Francis, S. Jamin, C. Jin, Y. Jin, V. Paxson, D. Raz, Y. Shavitt, and L. Zhang, “Idmaps: A global internet host distance estimation service”, in IEEE/ACM Transactions on Networking, Oct. 2001.

[18] K. P. Gummadi, S. Saroiu, and S. D. Gribble, “King: Estimating latency between arbitrary Internet end hosts”, in Proc. of SIGCOMM IMW 2002, Marseille, France, Nov. 2002.

[19] B. Wong, A. Slivkins, and E. G. Sirer, “Meridian: A lightweight network location service without virtual coordinates”, in SIGCOMM, Philadelphia, Pennsylvania, USA, Aug. 2005.

[20] P. Sharma, Z. Xu, S. Banerjee, and S.-J. Lee, “Estimating Network Proximity and Latency”, in ACM SIGCOMM Computer Communications Review, Pisa, Italy, Jul. 2006.

[21] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak and M. Bowman, “PlanetLab: an overlay testbed for broad-coverage services” in ACM SIGCOMM Computer Communication Review, Karlsruhe, Germany, Jul. 2003.

[22] “The PlanetLab Consortium”, URL: http://www.planet-lab.org, Jan. 2008.

[23] L. Rizzo, “Dummynet: a simple approach to the evaluation of network protocols” in ACM SIGCOMM Computer Communication Review, Cannes, France, Jan. 1997.

[24] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker, “Scalability and accuracy in a large-scale network emulator” in Proceedings of OSDI, Boston, MA, USA, Dec. 2002.

[25] “The Network Simulator - ns-2”, URL: http://www.isi.edu/nsnam/ns, Jan. 2009.

[26] X. Chang, “Network simulations with OPNET” in Proceedings of Simulation Conference, Phoenix, AZ, USA, Dec. 1999.

[27] “OPNET Solutions: Network R&D”, URL: http://www.opnet.com/solutions/network_rd, Jan. 2009.

[28] E. Zegura, K. Calvert and S. Bhattacharjee, “How to model an internetwork” in INFOCOM, San Francisco, CA, USA, Mar. 1996.

[29] A. Medina, A. Lakhina, I. Matta and J. Byers, “BRITE: An Approach to Universal Topology Generation” in Proc. of the International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS), Cincinnati, OH, USA, Aug. 2001.

[30] P. Mahadevan, C. Hubble, B. Huffaker, D. Krioukov and A. Vahdat, "Orbis: rescaling degree correlations to generate annotated Internet topologies" in Proc. of ACM SIGCOMM., Kyoto, Japan, Aug. 2007.

[31] Sun Microsystems, Inc, “Java Technology”, URL: http://java.sun.com, Jan. 2009.

[32] S. Floyd and V. Paxson, “Difficulties in Simulating the Internet” in IEEE/ACM Transactions on Networking (TON), Piscataway, NJ, USA, Aug. 2001.

[33] “HyperCast”, URL: http://www.comm.utoronto.ca/hypercast, Jan. 2009.

[34] “The C++ Resources Network”, URL: http://www.cplusplus.com, Jan. 2009.