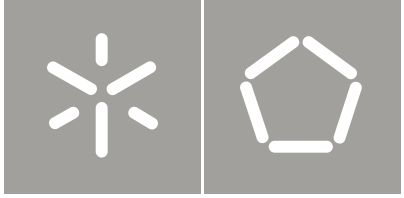


Universidade do Minho
Escola de Engenharia

Paulo César de Oliveira Jesus

Agregação e Contagem em Redes P2P

Tese de Mestrado
Mestrado em Sistemas Móveis

Trabalho efectuado sob a orientação do
Professor Doutor Carlos Miguel Ferraz Baquero Moreno
Professor Doutor Paulo Sérgio Soares de Almeida

Julho de 2007

DECLARAÇÃO

Nome

_____

Endereço electrónico: _____ Telefone: _____ / _____

Número do Bilhete de Identidade: _____

Título dissertação □/tese □

_____

_____

_____

Orientador(es):

_____

_____ Ano de conclusão: _____

Designação do Mestrado ou do Ramo de Conhecimento do Doutoramento:

_____

É AUTORIZADA A REPRODUÇÃO INTEGRAL DESTA TESE/TRABALHO APENAS PARA EFEITOS DE INVESTIGAÇÃO, MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE;

Universidade do Minho, ____/____/_____

Assinatura: _____

Agradecimentos

Aproveito a oportunidade para agradecer a todas as pessoas que me ajudaram e apoiaram durante o processo de desenvolvimento desta dissertação. A todas elas dedico a concretização deste estudo.

Agradeço em especial:

- Aos meus orientadores, Professor Doutor Carlos Miguel Ferraz Baquero Moreno e Professor Doutor Paulo Sérgio Soares de Almeida, pela sua disponibilidade, pelas suas importantes orientações, críticas e sugestões fornecidas ao longo da realização deste estudo;
- À Telma pelo seu apoio, motivação e ajuda nos momentos mais difíceis;
- Aos meus colegas de Mestrado, pelo seu companheirismo, espírito de equipa e alegria.

Agregação e Contagem em Redes P2P

Resumo

A agregação tem um papel importante na implementação de sistemas distribuídos, nomeadamente em rede P2P (*peer-to-peer*), permitindo a obtenção de propriedades globais (tais como o tamanho da rede, ou a temperatura média de uma rede de sensores). Estes valores globais podem por sua vez, ser utilizados na execução de operações basilares para o funcionamento do sistema. Embora aparentemente simples, a agregação tem-se revelado um problema difícil e interessante, quando se procura soluções em ambientes distribuídos, onde não existe um elemento central único com uma visão global do sistema.

Este estudo descreve diversas soluções para o problema da agregação, sendo proposta e avaliada uma nova solução para este problema. Esta dissertação oferece duas contribuições científicas relevantes acerca deste tema. A primeira contribuição consiste na apresentação de uma taxonomia das principais técnicas e mecanismos de agregação existentes, sendo esta dividida de acordo com dois aspectos principais: comunicação (referindo os protocolos e estruturas de comunicação usados) e computação (citando os fundamentos e princípios de computação nos quais os algoritmos se baseiam).

A segunda e mais importante contribuição deste estudo consiste na apresentação de uma nova solução para o problema da agregação e contagem (determinação do tamanho da rede). O novo algoritmo proposto — *Flow Updating* — evidencia um melhor desempenho global que os algoritmos anteriores da mesma categoria — *averaging*, assumindo-se como a melhor solução em termos de velocidade de execução e custos de comunicação. Mais importante, é a introdução de uma abordagem que lhe permite uma efectiva tolerância a faltas, sendo esta uma característica que não foi encontrada nos algoritmos de *averaging* anteriores.

Aggregation and Counting in P2P Networks

Abstract

Aggregation plays an important role in the implementation of distributed systems, namely in P2P (peer-to-peer) networks, providing the summary of global properties (like the network size, or the average temperature read by a sensor network). These global values can be used to perform key operations during system execution. Although apparently simple, aggregation has reveled to be a hard, and rich, problem when seeking solutions in distributed environments, with no single element holding a global vision of the system.

This study describes several solutions for the aggregation problem, and, as well, presents and evaluates a new solution for this problem. This dissertation includes two relevant scientific contributions about this subject. The first contribution consists on a survey of the essential aggregation's techniques and existing mechanisms, organized according two main aspects: communication (referring protocols and structures used in data communication) and computing (indicating the computing concepts and models used by the algorithms).

The second and main contribution of this study describes a new solution to the aggregation and counting (network size estimation) problem. The new algorithm — *Flow Updating* — exhibits a better global performance than previous algorithms in the same category (averaging), proving to be the best solution in terms of execution speed and communication cost. Most important, it introduces an approach that can tolerate message losses, a property that was not found on existing averaging algorithms.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O advento da Internet, o seu crescimento e expansão a escala mundial, potenciaram o aparecimento de um mercado de novas oportunidades e desafios das tecnologias da informação. Actualmente, a Internet permite uma fácil colaboração, comunicação e partilha de recursos em massa entre os quatro quantos do mundo. Desta feita, veio permitir uma aproximação das pessoas, instituições e outras entidades até aqui distantes fisicamente, promovendo a partilha de cultura e conhecimento. Portais cooperativos e comerciais, aplicações de comunicação VoIP (*Voice over Internet Protocol*) e sistemas P2P (*peer-to-peer*) de partilha de recursos, constituem alguns exemplos práticos que comprovam a importância que a Internet assumiu nos dias de hoje a todos os níveis (educativo, comercial, cultural ou simples entretenimento). Esta globalização constitui um núcleo de novas oportunidades, fomentando o aparecimento de novas ideias e aplicações, tentando tirar partido da sua abrangência e capacidade de distribuição de recursos.

Recentemente tem-se verificado um crescimento dos sistemas móveis e o aparecimento de novos paradigmas — computação ubíqua, tentando difundir no nosso meio quotidiano elementos virtuais com os quais podemos interagir, com o intuito de facilitar a execução das mais diversas tarefas do dia-a-dia, podendo inclusive proporcionar-nos novas experiências e vivências da realidade. Este tipo de sistemas começam a marcar uma nova era "*Everything, Everywhere Computing*" [66], onde de forma ubíqua e *Ad-hoc* se torna pos-

sível aceder à "rede" (Internet ou telecomunicações para uso de serviços de dados e voz) ou a qualquer outro sistema de informação existente no nosso raio de alcance, de acordo com a na nossa proximidade geográfica e credenciais de acesso. Esta filosofia e as suas características genéricas fomentam o aparecimento e crescimento de novos sistemas distribuídos, sustentados por novos e avançados recursos tecnológicos (redes de sensores e pequenos dispositivos de computação móvel).

Esta conjuntura, essencialmente orientada para a distribuição de unidades de computação e interacção P2P (ponto-a-ponto), embora bastante aliciante, levanta novos problemas e desafios do ponto de vista tecnológico e científico. Nesta área, o modelo de computação P2P, ao contrário do modelo cliente-servidor, apresenta melhores propriedade em termos de descentralização de processamento, distribuição de recursos, escalabilidade, adaptabilidade, dinamismo e tolerância a faltas. No entanto, tendo em conta este modelo de funcionamento, onde cada nó (*peer*) tem a mesma oportunidade de participar no sistema, torna-se problemático determinar qualquer propriedade global do sistema, uma vez que não existe nenhum elemento central com uma visão global do mesmo. Neste cenário torna-se necessário recorrer a mecanismos de agregação. É neste contexto que surge este estudo, abordando de forma abrangente um problema específico dos sistemas distribuídos: Agregação e contagem em redes P2P.

Neste estudo são descritas diversas soluções para o problema da agregação, sendo proposta e avaliada uma nova solução para este problema. Esta dissertação apresenta duas contribuições científicas relevantes acerca deste tema. A primeira contribuição deste trabalho consiste na primeira descrição de uma taxonomia das principais técnicas e mecanismos de agregação existentes, considerando várias dimensões de classificação (computação e comunicação). A segunda contribuição deste estudo consiste na apresentação de uma nova solução para o problema da agregação e contagem (determinação do tamanho da rede). O novo algoritmo proposto — *Flow Updating* — revela um melhor desempenho que todos os seus concorrentes (alguns dos algoritmos mas relevantes da mesma categoria), sendo mais rápido (melhor tempo de execução), mais "leve" (menor custo de comunicação) e mais

robusto (tolerante a ocorrência de perdas de mensagens). Para comparar o novo algoritmo com outros, foi implementado um simulador de rede que garante as mesmas condições de execução de todos os algoritmos avaliados e assegura uma justa comparação dos mesmos. No simulador de rede foi concretizada uma implementação específica dos algoritmos avaliados, para permitir estimar o tamanho de uma rede (um caso particular de agregação). Assim sendo, os resultados das simulações são apresentados no contexto da estimativa do tamanho de uma rede (contagem).

Esta dissertação está organizada do seguinte modo: No Capítulo 2 é efectuado um enquadramento geral deste estudo, sendo exposta a motivação para a realização deste trabalho e descrito um vasto conjunto de outros trabalhos relacionados com este tópico, sendo inclusive apresentada uma classificação taxonómica dos principais mecanismo e técnicas utilizados. No Capítulo 3 é efectuada uma descrição do novo algoritmo proposto como solução para o problema da agregação — *Flow Updating*. No Capítulo 4 é definido o modelo de simulação usado na avaliação da solução proposta, sendo apresentada uma sucinta descrição das principais características e funcionalidades do simulador de rede desenvolvido. Nesse capítulo, também é efectuada uma descrição da implementação dos algoritmo concorrentes (*Push-Sum Protocol* [38], *Push-Pull Gossiping* [31] e *DRG* [12]) com os quais o *Flow Updating* é comparado. No Capitulo 5 são analisados e discutidos os resultados da simulação do algoritmo proposto e dos demais algoritmos com os quais é comparado. Os resultados obtidos, comprovam o melhor desempenho do *Flow Updating*, sendo apontadas algumas fragilidades em alguns dos seus concorrentes. Finalmente, no Capítulo 6 são apresentadas algumas conclusões do presente estudo e, sugeridas algumas linha de investigação e pontos a desenvolver num trabalho futuro.

Capítulo 2

Enquadramento

Este estudo aborda um problema mediático da área dos Sistemas Distribuídos, com aplicação nos Sistemas Móveis e, que tem sido alvo de uma activa participação ao nível da investigação científica internacional — Agregação em Sistemas *Peer-to-Peer*, para o qual é proposto uma nova solução.

Em termos conceptuais, a computação *Peer-to-Peer* (P2P) [53] apresenta-se como a alternativa ao modelo cliente-servidor de um sistema distribuído. O P2P é amplamente usado já há muitos anos, sendo a Internet (do ponto de vista estrutural e funcional) o exemplo mais evidente da sua utilização e potencial. Tendo em conta as suas características, o modelo de computação P2P tem-se revelado como o mais apropriado para dar resposta aos recentes e emergentes paradigmas da computação móvel e ubíqua. O P2P ostenta as capacidades necessárias para responder aos requisitos de descentralização, adaptabilidade, dinamismo e tolerância a faltas, próprias das redes sem fios (*Ad-hoc*) e redes de sensores.

As principais características do P2P são a total distribuição de recursos e descentralização do processamento, permitindo a criação de sistemas mais flexíveis/dinâmicos, escaláveis e tolerantes. No entanto, a natureza descentralizada e *Ad-hoc* impõem determinados cuidados na concepção de aplicações para este tipo de arquitectura, na medida em que fornece a cada nó (*peer*) a mesma oportunidade de participar no sistema, não existindo nenhum elemento central com uma visão global do sistema. A independência de

um elemento central fomenta uma maior autonomia de cada nó e uma maior robustez do sistema, assim como facilita a evolução do sistema em maior escala. Com a descentralização do sistema, a relação entre a comunicação e a computação (local) entre os vários nós assume grande importância na sua capacidade de escalar (pouca comunicação e mais computação ⇒ mais escalável; muita comunicação e pouca computação ⇒ menos escalável). O facto de não depender de nenhuma infra-estrutura específica previamente estabelecida, podendo assumir qualquer tipo de forma e tamanho, confere ainda ao P2P flexibilidade e suporte para a implementação de sistemas dinâmicos. Do ponto de vista financeiro, o P2P possibilita a distribuição de custos na criação de sistemas de grande dimensão, tendo em conta uma possível cooperação das diversas entidades que partilham o sistema. Ao nível do desempenho, este tipo de sistemas é condicionado pela combinação entre a capacidade local de cada nó e a capacidade global de agregação do sistema (deixando de ser influenciada essencialmente por um único elemento central).

A natureza descentralizada do P2P conduz intuitivamente à necessidade de agregação de recursos e informação. A agregação é definida como a capacidade de agrupar, resumir e difundir informação, apresentando-se como um dos principais pilares na implementação de aplicações distribuídas nesta classe de sistemas [65]. A agregação tem um papel importante, podendo ser de grande utilidade na concretização de determinadas operações, tais como: a eleição de um líder (elegendo o elemento com o identificador de maior valor — máximo, ou simplesmente contando votos e números de processos); o balanceamento da carga do sistema (contando e calculando os valores médios, máximos e mínimos da carga do sistema); a monitorização do sistema, possibilitando a auto adaptação e recuperação de situações de erro.

Na secção seguinte são descritos mais alguns exemplos que retratam a importância da agregação em redes ponto-a-ponto, constituindo um factor de motivação para a realização deste estudo.

2.1 Motivação

A agregação é uma componente importante em redes ponto-a-ponto permitindo a obtenção de diversas estatísticas de rede e informações de admin-

istração, como por exemplo: os recursos disponíveis na rede, o tempo médio de sessão de cada nó, a carga média, máxima e mínima da rede. Se consideraremos a utilização de redes de sensores, a utilização de técnicas de agregação é essencial na monitorização e controlo de um determinado ambiente, permitindo o calculo de varias estatísticas, tais como: a temperatura média, a humidade média, a concentração de terminado elemento nocivo (e.g., monóxido de carbono), nível de ruído, etc. Dentre todas as estatísticas, a estimativa do tamanho da rede assume-se como uma das mais importantes, possibilitando o calculo de outras mais complexas. Em diversas situações, a informação do tamanho da rede também é utilizada como parâmetro de entrada para a execução de acções basilares ao funcionamento de determinados algoritmos e aplicações.

São vários os algoritmos que beneficiam do conhecimento do tamanho da rede para o seu correcto funcionamento, nomeadamente:

- Na criação autónoma de identificadores em ambientes móveis [33], recorrendo a uma simples geração de números aleatórios, a informação da dimensão do sistema é fundamental para a determinação do tamanho dos identificadores. Considerando a criação de um identificador de tamanho fixo, é possível calcular o número máximo de identificadores únicos que podem ser gerados de forma autónoma (pelo referido método), garantindo uma probabilidade predefinida para a ocorrência de uma colisão de identificadores (geração de um identificador já existente).

- Em [16][1] é apresentado um serviço probabilístico de comunicação em grupo baseado num processo de *random walk* para se adaptar aos requisitos de tolerância a faltas e dinamismo das redes *Ad-hoc*, assim como para determinar os elementos pertencentes a cada grupo. Este algoritmo usa a estimativa do tamanho da rede para calcular os limites superiores de algumas variáveis (contadores e tempo máximo de execução de um *random walk*). A utilização de uma estimativa mais exacta n ao invés do limite superior N permite uma reacção mais rápida às alterações da rede, ou seja, à adição e à remoção de novos elementos.

[1] Primeira versão apresentada em [15]

- Na construção determinística ou probabilística de topologias de encaminhamento para DHTs (*Distributed Hash Tables*), é essencial recorrer à estimativa do tamanho da rede para possibilitar a sua adaptação em redes ponto-a-ponto (tipicamente de natureza dinâmica) [48]. Outros algoritmos de construção e manutenção de topologias de encaminhamento recorrem ao conhecimento do tamanho da rede, para melhorar e/ou ajustar a sua performance, nomeadamente: Chord [61], Pastry [59], Tapestry [69], Viceroy [45], Symphony [47] e CAN [58].

- Alguns protocolos de disseminação epidémica de informação [22, 19] recorrem à estimativa do tamanho da rede para determinar o número de destinos considerados em cada nó.

Existem diversos métodos e técnicas que abordam o problema da estimativa do tamanho de uma rede e a realização de contagens (ver Secção 2.2). Essas técnicas são essencialmente baseadas em processos de agregação, não existindo actualmente nenhum algoritmo que consiga conciliar a obtenção de uma estimativa precisa (praticamente exacta) com a capacidade de adaptação ao dinamismo do sistema e à tolerância a faltas. O estabelecimento e expansão dos sistemas móveis (redes *Ad-hoc* e redes de sensores) e a necessidade de responder aos requisitos apresentados por estes, talvez sejam alguns dos factores que justifique a diligente actividade científica em específico nesta área. Ultimamente, têm surgido diversas contribuições apresentando novas técnicas, que comparam e avaliam alguns algoritmos existentes [46, 42, 57, 8, 12]. A necessidade de fornecer uma melhor resposta a este problema, tendo em atenção os requisitos de precisão, tolerância e adaptabilidade, constitui a principal motivação deste estudo acerca da "Agregação e Contagem em redes P2P".

2.2 Trabalho Relacionado

Diversos trabalhos têm sido desenvolvidos e publicados com o objectivo de apresentar a "melhor" (mais genérica, mais rápida, menos exigentes do ponto

de vista computacional ou com menor quantidade de troca de mensagens) solução para o problema da agregação e/ou contagem, considerando a sua aplicação em diversos cenários. Na secção subjacente é exposto um panorama geral dos estudos efectuados especificamente nesta área, sendo efectuada uma breve descrição dos algoritmos existentes por ordem cronológica (data da primeira publicação).

2.2.1 Evolução

(Madden et al., 2002) Em [44] é apresentado um método de agregação para redes *Ad-hoc* de sensores (sem fios) baseado nos mesmo princípios de agregação de informação usados em bases de dados, com suporte para a utilização de uma linguagem semelhante ao SQL (*Structured Query Language*). O método proposto permite agrupar os dados e implementar funções básicas de agregação em bases de dados, tais como: contagem, mínimo, máximo, soma e média. Para a sua execução, este algoritmo requer a criação e manutenção contínua de uma estrutura de encaminhamento de mensagens — "árvore", de forma a lidar com o dinamismo e mobilidade da rede. Nesta estrutura de encaminhamento existe um nó "raiz", o qual inicia o processo de agregação e aufere o resultado, sendo a agregação parcialmente calculada ao longo de cada nó da árvore até à raiz (cada nó agrega os dados dos seus "filhos" e reencaminha o resultado para o seu "pai"). O processo de agregação é constituído por duas fases: uma de propagação (em que o pedido de agregação é propagado pela estrutura de encaminhamento da rede, desde a raiz até as "folhas") e outra de agregação propriamente dita (em que os valores são agregados dos filhos para os pais, até chegarem à raiz). Este processo obriga à espera de um tempo mínimo até a obtenção do valor agregado correcto, de modo a garantir a conclusão das duas fases de execução e a participação de todos os nós no processo de agregação. O tempo de resposta até a recepção de um valor correcto depende do número de níveis da árvore de encaminhamento, sendo mais demorado para redes com uma estrutura mais "profunda". Para minimizar este efeito, este método usa uma técnica de *pipelined aggregate*, em que são utilizados intervalos temporais mais curtos (relativa-

mente ao tempo total necessário), nos quais são produzidos repetidamente resultados parciais da agregação. Durante cada intervalo, os nós que tenham recebido o pedido de agregação transmitem um resultado parcial, calculado através da aplicação da função de agregação ao valor lido localmente e aos resultados recebidos dos seus filhos durante do intervalo anterior. Ao longo da execução, ao fim de cada intervalo de tempo, o valor agregado resulta da participação de um número cada vez maior de nós, aumentando a fiabilidade do resultado e a aproximação ao resultado correcto que deverá ser alcançado após a execução de um número mínimo de repetições. Este método apresenta ainda algumas optimizações no sentido de diminuir o número de mensagens enviadas: tirando partido do *broadcast* de mensagens e concedendo poder de decisão local a cada nó no envio de mensagens — *hypothesis testing* (um nó pode decidir transmitir apenas se o valor agregado da sua sub árvore contribuir e afectar o valor agregado). Apesar disso, o correcto funcionamento deste algoritmo está totalmente dependente da estrutura de encaminhamento na qual assenta e da não ocorrência de rupturas na mesma.

(Dolev et al., 2002) Na determinação do tamanho da rede, em [16] [2] é usado um conjunto de *random walks* executados por agentes — *scouters*. Ao longo da execução do algoritmo, cada agente guarda o conjunto de todos os elementos por onde passa e um contador para cada um desses elementos. Quando um agente chega a um determinado nó, os contadores de todos os elementos são incrementados, excepto o do nó actual ao qual é atribuído o valor zero. Os elementos com um valor de contador muito elevado correspondem aos que não voltaram a ser visitados pelo *scouter*, suspeitando-se do seu desaparecimento na rede. Quando os contadores atingem um determinado valor os correspondentes elementos são removidos do conjunto guardado pelo agente — os elementos são ordenados por ordem crescente do valor do contador, sendo removidos se a diferença de valor relativamente ao anterior for muito elevada (superior a um valor predefinido). A contagem dos elementos guardados pelo agente fornece a estimativa do tamanho da rede.

[2] Primeira versão apresentada em [15]

(Jelasity and PreuB, 2002) A obtenção de informação global em ambientes P2P completamente distribuídos é efectuada em [32], através da difusão contínua da informação parcial do sistema existente localmente em cada nó, recorrendo à execução de um protocolo epidémico. Cada nó mantém uma base de dados "incompleta" de tamanho c com informação dos seus vizinhos (conjunto de c nós pertencentes a rede de tamanho n, em que normalmente $n \gg c$), sendo esta última actualizada periodicamente através da troca e fusão de informação com um vizinho escolhido aleatoriamente (ficando os dois actualizados). Explorando o normal funcionamento deste protocolo é possível obter uma estimativa do tamanho da rede, recorrendo ao calculo do número médio $\bar{d}$ de elementos diferentes detectados ao longo de k trocas de informação (dado pela expressão 2.1). A ideia principal desta técnica baseia-se na contagem da quantidade de novos elementos d detectados durante o processo de troca de informação de bases de dados entre nós (respectivamente as bases de dados D e D', sendo $d =\mid D' \setminus D \mid$). De acordo com este método, a estimativa do tamanho da rede é aproximada pela expressão 2.2.

$$\bar{d} = \frac{\sum_{i=1}^{k} d_i}{k} \tag{2.1}$$

$$n \approx \frac{\mid D' \mid\mid D \mid}{\mid D' \mid -\bar{d}} \tag{2.2}$$

(Kutylowski et al., 2002) Os autores apresentam um algoritmo probabilístico para efectuar uma aproximação ao tamanho da rede [35], num cenário de aplicação especifico — redes de acesso rádio *single-hop* com colisão de mensagens. Neste algoritmo assumem que o relógio local de cada elemento da rede está sincronizado com um relógio global. Em cada passo, cada um dos nós transmite uma mensagem com probabilidade p. A transmissão ocorre com sucesso, apenas se um único elemento da rede decidir enviar a mensagem. A probabilidade máxima desta ocorrência é atingida para $p = 1/N$, tendo um valor aproximado de $1/e$ (t/e se o processo for repetido t vezes). O algoritmo tira partido da referida característica, repetindo em cada etapa (com probabilidade fixa) o processo um determinado numero de vezes e contando

o número de transmissões com sucesso. Várias etapas são sucessivamente executadas, usando diferentes valores de probabilidade (decrescentes) e número de repetições (crescentes), até o número de transmissões com sucesso estar próximo do valor esperado (sendo calculado o valor aproximado do tamanho da rede). Este método foi recentemente melhorado, de forma a torná-lo imune a ataques de um adversário e tolerar faltas de transmissão [37]. Para tal, os autores recorreram à utilização de uma "janela" temporal (cada passo do algoritmo é executado em janelas temporais diferentes durante apenas um dos seus intervalos de tempo, sendo o intervalo determinado por uma função criptográfica pseudo aleatória) e de uma técnica de intercalação (combina a execução independente e em paralelo do mesmo algoritmo por diversos grupos, em intervalos de tempo distintos e pseudo aleatórios dentro da mesma janela temporal).

(Bawa et al., 2003) Em [7] é apresentado um conjunto de métodos de agregação ("SingleTree", "Propagate2All" e "MultipleTrees") baseados na criação de uma topologia de comunicação em árvore e um conjunto de métodos ("RandomizedReport", "BdayParadox" e "RandomIncWalk") específicos para estimar o tamanho de uma rede.

SingleTree O algoritmo "SingleTree" opera em duas fases distintas. Numa primeira fase, o nó inicial dissemina pela rede (*broadcast*) o pedido de agregação, sendo construída um árvore de comunicação — cada nó reconhece como "pai" o primeiro nó do qual recebe o referido pedido de agregação. Na segunda fase, o valor agregado é propagado e calculado de baixo para cima (das "folhas" da árvore até a "raiz") — cada pai espera pela resposta de todos os seus filhos, calcula o valor agregado e propaga-o para o seu próprio pai.

Propagate2All O algoritmo "Propagate2All" é uma variante do "SingleTree" que tenta eliminar o grave efeito da ocorrência de falhas de comunicação, numa topologia onde para cada nó existe um único caminho até a raiz, levando ao extremo a criação de rotas alternativas. No "Propagate2All"

após a recepção do pedido de agregação, os nós começam logo a enviar o seu valor para todos os seus vizinhos. Quando um nó recebe o valor de um vizinho, ele calcula o novo valor agregado. Se o novo valor for diferente do valor anterior, ele é enviado para todos os vizinhos. Cada nó activo executa o processo de agregação durante um determinado intervalo de tempo — $2\widehat{D}\Delta$ ($\widehat{D}$: estimativa do diâmetro da rede; Δ: tempo máximo de envio de uma mensagem), sendo produzido o resultado da agregação no nó inicial após esse período temporal. Neste algoritmo o valor de um nó pode chegar a outro diversas vezes por caminhos diferentes, originando um valor sobrestimado para funções de agregação não idempotentes. Para solucionar este problema, o "Propagate2All" usa um algoritmo de contagem probabilística [20] para estimar o número de origens distintas dos valores recebidos e calcular um valor de agregação mais aproximado.

MultipleTrees O "MultipleTrees" é uma solução intermédia entre o "SingleTree" e o "Propagate2All". Durante a fase de broadcast do pedido de agregação, o "MultipleTrees" cria um número finito de árvores de comunicação independentes (duas ou três, em vez de apenas uma como no "SingleTree"). O processo de agregação é efectuado de baixo para cima usando o mesmo esquema de agregação de dados do "Propagate2All".

RandomizedReport O "RandomizedReport" é um dos algoritmos específicos para estimar o tamanho da rede. Neste algoritmo um nó inicial envia um pedido para toda a rede (*broadcast*), cada nó que recebe o pedido responde ao nó inicial sujeito a uma probabilidade predefinida p. O nó inicial conta o número de respostas r (ao fim de um determinado período de tempo) e calcula a estimativa do tamanho da rede considerando o valor de probabilidade usado ($\widehat{N} = r/p$).

BdayParadox O algoritmo "BdayParadox" baseia-se no *birthday problem* e nos *random walks* para produzir uma estimativa do tamanho da rede. A ideia deste algoritmo é seleccionar aleatoriamente um conjunto de nós, através de um processo de *random walks*, até voltar a escolher um dos nós

anteriormente seleccionados. A contagem do total de nós amostrados x até à ocorrência da primeira repetição é usada para estimar o tamanho da rede ($\widehat{N} = x^2/2$).

RandomIncWalk O "RandomIncWalk" apresenta uma estratégia de *random walk* diferente da anterior. Neste algoritmo, o *random walk* é realizado progressivamente por nós da rede com um identificador cada vez maior até o processo não poder avançar mais. Em cada nó é escolhido aleatoriamente um elemento do conjunto de vizinhos com um identificador maior que o actual, para que lhe seja passado o testemunho usado no processo de *random walk*. A distância percorrida pelo testemunho é guardada e o processo repetido diversas vezes. A media das distâncias percorridas pelos processos de *random increasing walk* é usada para estimar o tamanho da rede.

(Manku, 2003) De forma a permitir a adaptação de topologias de encaminhamento para DHTs em redes de natureza dinâmica — redes P2P, em [48] foi desenvolvido um método para estimar o tamanho da rede em todos os nós. Esta técnica baseia-se na medição da densidade dos identificadores mais próximos, apresentando algumas semelhanças com a técnica de contagem probabilística [20][3]. A correcta aplicação desta técnica implica a existência de uma distribuição uniforme dos identificadores únicos associados aos nós da rede.

(Kutylowski et al., 2003) Kutylowski et al. [41] apresentam um algoritmo para o cálculo da média (arredondado para um valor inteiro) num cenário muito específico. É considerada a aplicação do algoritmo numa rede *Ad-hoc* de acesso rádio (sem fios) com múltiplos canais de comunicação (de acordo com a norma IEEE 802.11) e um modelo de comunicação em que todos os nós comunicam directamente com os restantes (*single-hop*) num meio partilhado (transmissão de dados por *broadcast*). O modelo de comunicação considera a ocorrência de falhas de comunicação, caso mais do que um nó

[3]Como os próprios autores referem

da rede tente transmitir em simultâneo através do mesmo canal de comunicação. O protocolo consiste na repetição ao longo do tempo de uma etapa constituída por $3N/R$ passos (N: tamanho da rede; R: número de canais de comunicação), em que três passos consecutivos representam uma ronda. Durante uma etapa, os nós da rede transmitem e recebem dados através de canais e rondas diferentes (ambos determinados através da escolha de valores aleatórios). Cada nó executa um conjunto de operações distintas em dois canais e rondas diferentes determinados pela escolha de dois valores aleatórios ($t, t' \in [1, \cdots, N]$). No primeiro canal ($t \ mod \ R$) e respectiva ronda ($\lceil t/R \rceil$), cada nó da rede envia o seu valor actual num dos dois primeiros passos da ronda (determinado pela escolha de um bit aleatório). No terceiro passo da ronda, os nós ficam à escuta de mensagens que contenham o valor transmitido e outro valor associado, sendo posteriormente efectuado o cálculo da média e o resultado obtido adoptado como novo valor. No segundo canal escolhido ($t' \ mod \ R$) e respectiva ronda ($\lceil t'/R \rceil$), cada nó escuta as mensagens recebidas durante os dois primeiros passos, concatenando e enviando as mensagens recebidas no terceiro passo. Este algoritmo implementa uma função de agregação específica — média, assumindo o conhecimento do tamanho da rede (geralmente também calculado por algoritmos de agregação mais genéricos).

(Kempe et al., 2003) Uma propagação epidémica uniforme de pequenas mensagens é proposta em [38] para implementar um protocolo de agregação em sistema distribuídos de grande escala. A ideia base do algoritmo (*Push-Sum Protocol* — para calcular somas e médias) consiste na distribuição par a par do valor a agregar pelos nós vizinhos na rede. Para isso, cada nó mantém e propaga o valor de duas variáveis: uma variável s para guardar a soma dos valores agregados e uma variável w para armazenar um peso associado (o valor inicial das referidas variáveis depende da função de agregação que se pretende utilizar — soma, média ou contagem). Periodicamente, cada nó escolhe aleatoriamente um vizinho. De seguida, o nó envia metade do valor das suas variáveis para o vizinho escolhido e outra metade para ele próprio. Os novos valores recebidos são somados aos valores anteriores. Em

cada iteração, o valor agregado pode ser estimado pelo quociente entre as duas variáveis s/w. A precisão da estimativa obtida com este algoritmo depende da conservação da "massa" do sistema ao longo do tempo, ou seja, da manutenção do valor global do sistema resultante do somatório de todos os valores locais (em cada nó). Tendo em consideração a importância da conservação da "massa" para obter um resultado agregado correcto, os autores assumem que os nós têm capacidade de detectar faltas na entrega das mensagens (reenviando para eles próprios as mensagens que não chegaram ao destino, conservando assim a "massa" do sistema). Na Secção 4.3 é exposta informação mais detalhada acerca da utilização deste método para estimar o tamanho de uma rede.

(Horiwitz and Malkhi, 2003) K. Horiwitz e D. Malkhi [28] apresentam uma técnica para estimar o tamanho da rede executando o algoritmo apenas na entrada ou saída de elementos da rede. Para concretizar o processo de estimativa é mantido um anel virtual na rede, estando cada nó da rede ligado a um único "sucessor". Os autores assumem que cada nó da rede detém um estimador preciso. Quando um novo nó entra na rede é lhe atribuído aleatoriamente um "sucessor", do qual consulta de imediatamente o estimador, incrementando por um o valor da estimativa, assumindo os dois nós o resultado da nova estimativa. No caso da saída de um elemento da rede, é executado o processo inverso. Este método permite obter uma estimativa "grosseira" e dispersa ao longo da rede (variando entre $n/2$ e n^2, sendo n o tamanho da rede), utilizando no entanto poucos recursos de comunicação durante o processo.

(Jelasity et al., 2004) Em [29] é proposto um método de agregação (para determinar médias e máximos) descentralizado e escalável, assente na utilização do protocolo *newscast*. O protocolo consiste na disseminação da informação de uma *cache* de itens (de tamanho predefinido), mantida por cada nó da rede. Periodicamente, cada nó escolhe aleatoriamente outro nó, considerando a informação dos endereços de rede existente nos diferentes itens da *cache*. Os dois nós trocam a informação das suas *caches* e fundem a infor-

mação recebida com a informação local (descartando os itens mais antigos e garantindo a existência na *cache* de pelo menos um item proveniente de cada nó). A aplicação da função adequada (máximo ou média), considerando os valores existentes na *cache* (trocados durante a execução do protocolo *newscast*), permite determinar uma estimativa da agregação em todos os nós da rede. Este método prevê um funcionamento contínuo, em sucessivos ciclos de execução, de forma a permitir adaptar a sua utilização em ambientes dinâmicos.

(Considine et al., 2004) Um método de agregação para redes de sensores é apresentado em [13], considerando a ocorrência de faltas e a utilização de funções de agregação sensíveis a recepção de valores duplicados (somas, contagens e médias). Este método é baseado na técnica de contagem probabilística de Flajolet e Martin [20] usada para estimar o número de elementos distintos numa base de dados [21], sendo proposta uma generalização desta técnica para a sua utilização em outras funções de agregação não idempotentes — soma. De forma a suportar a ocorrência de falhas de comunicação (e a falha de alguns nós da rede) é considerada a utilização de técnicas de encaminhamento *multi-path* (vários caminhos possíveis para chegar a um determinado destino). O algoritmo é executado em duas fases, sendo o processo de agregação iniciado por um único nó e o resultado final determinado no mesmo. Na primeira fase, um nó — *root* (raiz) — efectua o pedido de agregação, sendo este último propagado pela rede. Durante a fase de propagação, todos os nós determinam a sua distância (nível) relativamente ao nó inicial e guardam os níveis dos seus vizinhos — resultando na criação de uma topologia de encaminhamento hierárquica e *multi-path* (semelhante a criação de múltiplas árvores de encaminhamento). Na segunda fase, são calculados os valores agregados parciais (tendo em conta a generalização da técnica de contagem referenciada anteriormente) e propagados por *broadcast* em várias rondas pelos diversos níveis (criados na primeira fase) da rede, desde os nós mais afastados (nível mais elevado) até ao nó inicial. Em cada ronda, os dados recebidos pelos nós do próximo nível são agregados com os dados parciais existentes localmente, até ao nó inicial onde é calculado o valor agregado

final.

(Jelasity and Montresor, 2004) Jelasity e Montresor [30] propuseram a utilização de um protocolo de agregação anti-entropia, para estimar de forma proactiva e contínua os valores agregados (valores extremos, medias e contagens) em todos os nós de uma rede. A execução deste protocolo recorre a uma propagação epidémica, em que ocorrem trocas síncronas de valores agregados entre vizinhos (par-a-par). Cada nó escolhe periodicamente um vizinho para lhe enviar o seu valor actual. Após o envio do seu valor agregado, o nó activo aguarda a recepção do valor do seu vizinho e aplica a função de agregação tendo em conta o valor enviado (actual) e recebido, a fim de obter uma nova estimativa da agregação. Sempre que um nó recebe o valor agregado de um vizinho, ele responde com o seu valor actual e, calcula uma nova estimativa usando como parâmetro o valor recebido e o enviado (actual). Para permitir a contínua adaptação dos valores agregados num ambiente dinâmico, os autores sugerem a aplicação de um mecanismo de reinicialização. De acordo com este mecanismo, o protocolo é reiniciado periodicamente (podendo coexistir várias execuções em paralelo) sempre que é atingido um número predefinido de ciclos de execução (necessários para atingir o grau de convergência desejado). Apesar da análise apresentada acerca da adaptação do algoritmo ao dinamismo da rede, não é abordado o problema da ocorrência de faltas.

Push-Pull Gossiping Em [31] é apresentado um estudo mais aprofundado da ideia anteriormente descrita — "Push-Pull Gossiping", sendo exposta uma solução mais matura em termos práticos (separação da execução em duas *threads* distintas; uso de um *timeout* para detectar possíveis faltas, ignorando a troca de mensagens nessa situação; sugestão de várias variantes de implementação para diversas funções de agregação; consideração da execução de múltiplas instâncias para aumentar a robustez do protocolo). Na Secção 4.3 é exposta informação mais detalhada acerca da utilização deste algoritmo para estimar o tamanho de uma rede.

(Psaltoulis et al., 2004) Para estimar o tamanho de grupos dinâmicos, em [40, 57] são propostos dois algoritmos baseados em técnicas de amostragem.

Hop Sampling O primeiro algoritmo — *Hop Sampling* — efectua uma propagação epidémica para o grupo e recolhe os tempos de resposta dos diferentes nós. Os tempos angariados são usados para estimar o logaritmo do tamanho do grupo. Neste algoritmo o resultado da estimativa é determinado de forma activa e calculado apenas num só nó — iniciador. O iniciador envia periodicamente uma mensagem para um determinado número de nós (*gossipTo*) escolhidos aleatoriamente, durante um número predefinido de iterações (*gossipFor*) ou até receber uma quantidade máxima de mensagens (*gossipUntil*). Após a recepção da primeira mensagem, os restantes nós iniciam também o envio periódico da mensagem de iniciação, seleccionando aleatoriamente os nós destino (*gossipTo*), terminando quando atingirem o máximo de iterações (*gossipFor*) ou o máximo de mensagens recebidas (*gossipUntil*) previamente definidos. Durante este processo é mantida em cada nó uma lista dos nós que já participaram no processo de propagação (nós "infectados" dos quais recebeu mensagens). Os destinatários das mensagens são escolhidos aleatoriamente, sendo excluindo os nós existentes na lista de participação (*fromList*). Cada nó, também mantêm a distância mínima até ao nó iniciador, medida em número de saltos (*MyHopcount*). O valor incrementado do número de saltos é enviado em todas as mensagens de propagação. Cada nó guarda o valor mínimo do número de saltos recebidos, sendo nas mensagens enviadas colocado o valor actual incrementado por um. Após a finalização do processo de propagação descrito, depois de decorrer um número predefinido de iterações (*gossipResult*), o iniciador recolhe os tempos de resposta (número de saltos — *MyHopcount*) de uma amostra de nós escolhidos aleatoriamente. A média dos valores recebidos é usada como estimativa do logaritmo do tamanho do grupo. Em alternativa ao processo de amostragem efectuado pelo nó iniciador, imediatamente após a conclusão do processo de propagação, cada nó pode decidir enviar a sua contagem de saltos para o nó inicial, de acordo com uma probabilidade predefinida (fazendo com que apenas uma amostra de nós responda ao iniciador).

Interval Density O segundo algoritmo — *Interval Density* — mede a densidade do espaço de identificadores, determinando o número de identificadores contidos num sub intervalo desse espaço. O nó iniciador recolhe passivamente informação acerca dos identificadores dos nós, espiando a informação recebida na execução de protocolos complementares. Nesta técnica é aplicada uma função de *hash* a cada identificador, de forma a efectuar uma correspondência com um ponto do intervalo [0, 1]. Através da determinação do número de identificadores X contidos num sub intervalo I de [0, 1], o iniciador estima o tamanho do grupo (X/I).

(Li et al., 2005) Em [43] é proposto um esquema de agregação baseado na criação e manutenção de uma "árvore" de encaminhamento para DHTs. A árvore é criada de baixo para cima (ao contrário dos esquemas habitualmente utilizados) recorrendo a utilização de uma *função parental* — que determina independentemente qual é o único "pai" de cada nó. Os autores assumem a existência de uma DHT eficiente com um espaço de nomes continuo e circular, sendo cada nó responsável por um intervalo de identificadores (entre ele próprio e o seu predecessor). O protocolo de construção da árvore de agregação processa-se da seguinte forma: aquando da chegada de um novo nó é lhe transmitido a *função parental* e a informação do intervalo de identificadores pelo qual é responsável; se o identificador da "raiz" está contido dentro do intervalo do novo nó, este último torna-se a nova raiz, senão ele determina qual é o seu pai recorrendo a *função parental*; o novo nó envia uma mensagem de registo para o seu pai que o adiciona a sua lista de "filhos". Tendo em conta o dinamismo da rede e a possibilidade de ocorrência de faltas, é cumprido um protocolo de manutenção da árvore, de forma a permitir uma rápida recuperação de situações de falha e a adaptação da estrutura da rede à chegada e partida de nós. De acordo com as características da árvore de agregação são propostos dois modos de operação para a concretização da agregação: modo por omissão — em que a agregação é executada de forma transparente (em *background*), aproveitando as mensagens trocadas continuamente durante o processo de manutenção da árvore; modo por pedido — quando um determinado nó efectua um pedido de agregação

especifico à rede. No primeiro modo de operação (*default mode*), durante o processo de manutenção, as mensagens de actualização de estado dos filhos são usadas para propagar os resultados parciais da agregação até à raiz onde é calculado o valor final. O resultado é posteriormente propagado pela rede, através das mensagens de confirmação de estado para os filhos. No segundo modo de operação (*on-demand mode*), o nó que pretende a execução de uma função de agregação envia o pedido para a raiz que o propaga para toda a rede (de pais para filhos). As "folhas" da árvore (nós sem filhos) executam a função de agregação e enviam o resultado para o seus pais. Cada pai espera pelo resultado dos seus filhos e calcula o valor da agregação da sua sub árvore, enviando o resultado para o seu próprio pai, até chegar à raiz da árvore. Após a recolha dos resultados parciais de todos os seus filhos, o nó raiz calcula o resultado final da agregação e envia-o para o nó que efectuou inicialmente o pedido de agregação. Para aumentar a robustez do algoritmo, reduzindo o impacto da ocorrência de um único ponto de falha (perdendo a ligação com toda a sub rede ligada por esse ponto), é sugerida a coexistência de uma família de múltiplas árvores (construídas através da *função parental*) para calcular o mesmo valor agregado.

(Mane et al., 2005) Para estimar o tamanho de redes *peer-to-peer* estáticas (em que o tamanho da população se mantêm durante o processo de estimativa), em [46] é proposto uma técnica baseada na utilização do método estatístico de captura-recaptura. O método de captura e recaptura permite efectuar a estimativa de populações através de amostragens (duas ou mais) da população em análise e da contagem dos elementos comuns nas diversas amostras. Para a obtenção de bons resultados, o método requer que as amostras sejam obtidas de forma aleatória e independente. Considerando esse facto, os autores recorrem a execução de *random walks* na rede para a recolha das amostras. Em termos práticos o algoritmo é executado em duas fase idênticas (captura e recaptura), sendo efectuados dois *random walks* consecutivos a partir de um nó inicial. Em cada uma destas fases, o nó inicial envia uma mensagem para um dos seus vizinhos escolhido aleatoriamente. Por sua vez, o vizinho escolhido reencaminha a mensagem para um dos seus

outros vizinhos (escolhido aleatoriamente) e assim sucessivamente, até ser atingindo o valor máximo do numero de saltos predefinido (parâmetro *time-to-live*) ou até a mensagem ser recebida por um nó que já participou na fase em execução. Durante este processo é registado o caminho seguido pela mensagem (identificação de todos os nós por onde passa). Ao atingir um dos já referidos critérios de paragem do *random walk* é enviada uma mensagem de resposta (com a lista dos nós "capturado") de volta para o nó inicial, percorrendo o caminho inverso de propagação da mensagem inicial. A informação recebida no final das duas fases (nós que participaram nos *random walks*) é utilizada pelo nó inicial, para calcular a estimativa do tamanho da rede $\widehat{N}$ (Equação 2.3, em que: N_1 - número de nós capturados na primeira amostra; N_2 - número de nós recapturados na segunda amostra; n_{11} - número de nos comuns nas duas amostras).

$$\widehat{N} = \frac{((N_1 + 1) \times (N_2 + 1))}{(n_{11} + 1)} \tag{2.3}$$

(Ghdsi et al., 2005) Em [23] é proposto um algoritmo epidémico para estimar o tamanho de redes P2P, de forma continua em todos os nós da rede, com capacidade de se auto-estabilizar de forma a suportar a entrada/saída de nós na rede e a ocorrência de faltas. Este método assume a existência de um anel de identificadores únicos, em que é conhecido o universo dos identificadores possíveis (e por conseguinte o tamanho do espaço de identificadores — N). A estimativa da rede é calculada a partir da distancia média entre nós (distancia média entre dois nós consecutivos do anel de identificadores — δ), relativamente ao espaço de identificadores considerado, sendo dada por $\frac{N}{\delta}$. Aquando a chegada de um novo nó à rede é determinada a estimativa inicial da sua distância em relação ao seu sucessor, que também ajusta a sua estimativa durante este processo. Durante a saída (voluntária) de um nó, a estimativa do sucessor desse nó também é ajustada. A média das distâncias é estimada em todos os nós, recorrendo a execução de um algoritmo de agregação semelhante ao "Push-Pull Gossiping" [31] (apresentado anteriormente). O processo de auto-estabilização e tolerância a faltas assenta na capacidade de manutenção de um invariante do sistema: a soma da distância estimada

em todos os nós tem de ser igual ao tamanho do espaço de identificadores. A ocorrência de faltas (perda da informação da estimativa da distância em algum nó) provoca a violação do invariante do sistema. Para permitir recuperar as estimativas das distâncias perdidas, os autores recorrem a utilização de uma DHT para armazenar as estimativas dos nós e restaurar o estado do invariante, visto as DHTs possuírem normalmente mecanismos automáticos de replicação e detecção de faltas. Para recuperar o estado do sistema, em situações de erro na determinação do valor da estimativa das distâncias, é executado um algoritmo de auto-estabilização. Este algoritmo consiste no envio periódico de um testemunho a partir de um determinado nó, para somar as estimativas actuais de todos os nós ao longo do anel. Quando o testemunho volta ao nó inicial com a informação da soma, este faz o ajuste necessário ao valor da sua estimativa, de forma a repor o invariante do sistema.

(Ahmed et al., 2006) Um componente importante da agregação é a manutenção contínua dos valores agregados, principalmente se estamos a tratar de sistemas dinâmicos (entrada/saída de nós da rede) ou quando os valores locais mudam constantemente ao longo do tempo (e.g. a temperatura). Esta preocupação é abordada em [3], sendo proposta uma técnica de manutenção incremental dos agregados baseado numa variante do esquema de Flajolet e Martin (FM) [21]. Os autores classificam a manutenção de agregados de três formas distintas: *one-shot* — o valor agregado é calculado uma vez e não é actualizado; *drop-and-recompute* — o valor agregado é calculado periodicamente, não sendo considerados valores de execuções anteriores; *incremental* — o valor agregado é mantido e actualizado à medida que ocorrem alterações. O algoritmo proposto combina um mecanismo de agregação com um protocolo de encaminhamento para propagação de actualizações. Como mecanismo de agregação, os autores utilizam uma variante do algoritmo de contagem de FM. Cada nó mantêm um vector de bits que durante o processo de propagação de actualizações é combinado com os vectores recebidos, a fim de obter um valor agregado actualizado. Para a transmissão das actualizações (vectores de bits) entre os nós da rede, são considerados diversos protocolos de encaminhamento, tais como: *flooding*, *gossip* e *random walk*.

É realçada a preferência de utilização da versão incremental dos referidos protocolos de encaminhamento, atendendo ao facto de apenas ser necessário propagar informação aquando da ocorrência de alterações nos valores agregados (locais). O algoritmo proposto opera de forma semelhante aos conhecidos algoritmos epidémicos, passando cada nó por três estados distintos. Quando um nó recebe uma actualização, ele calcula o novo valor agregado e determina se a diferença introduzida é superior a um valor de significância — *significance threshold* — predefinido (ou calculado pelo sistema). Caso a diferença introduzida seja superior ao *significance threshold*, o novo valor é transmitido para os vizinhos utilizando o protocolo de propagação incremental escolhido, caso contrário nenhuma acção é tomada. Apesar do algoritmo FM apresentar algumas propriedades desejáveis, nomeadamente a insensibilidade à ocorrência de duplicações, também possui problemas de precisão que vão piorando ao longo do tempo de utilização. Como *workaround* para combater este problema, os autores consideram a utilização de um parâmetro — *threshold* (representando o número de actualizações trocadas entre vizinhos) que depois de atingido implica a reinicialização do protocolo — *Drop-and-recompute* (descartando a informação anterior dos vectores de bits). Embora o referido estudo destaque a importância da manutenção da agregação, ele apenas considera a alteração local dos valores num sistema estático onde não ocorrem falhas. Note-se que neste estudo, os resultados das várias simulações são apresentados a partir de um sistema inicial estável (já com o valor agregado calculado em todos os nós), não evidenciando o comportamento do algoritmo até o sistema atingir o primeiro valor agregado estável.

(Brik et al., 2006) O I-LEAG (*Instance-Local Efficient Aggregation on Graphs*) [8] apresenta-se como uma técnica que enfatiza a computação local na determinação de agregados. O método proposto permite apenas o calculo de funções de agregação em redes estáticas (tamanho fixo). A execução do algoritmo requer a pré construção de uma hierárquica de partições e respectiva atribuição de nós pivot em cada partição — formando uma "árvore" lógica. O I-LEAG é executado em várias fases sequenciais. Cada fase corresponde a um nível de partição na hierarquia previamente definida, sendo

processado paralelamente em cada partição do mesmo nível. Em cada fase, é efectuada a verificação da existência de conflitos (valores agregados diferentes entre nós guardados). Caso um conflito seja detectado, ele é reportado para todos os pivots associados ao nível de partição da fase em execução, que por sua vez reencaminham o resultado para os seus vizinhos. Ao receber uma mensagem de actualização de um vizinho, cada nó actualiza o seu valor agregado e guarda a informação do valor do seu vizinho (permitindo a posterior detecção local de conflitos). Em cada fase é usado um temporizador, a fim de garantir que a próxima fase não começa sem que tenham sido transmitidas todas as mensagens da fase actual, assegurando que o valor agregado é igual em todos os nós da mesma partição. Se nenhum conflito é detectado, nenhuma mensagem é transmitida. Os conflitos são apenas detectados entre vizinhos que pertençam a partições diferentes na fase anterior e com valor agregado diferente nas respectivas partições do nível anterior. Apesar do esforço deste algoritmo para optimizar a agregação, a sua aplicabilidade está restrita as redes estáticas (de tamanho fixo), dependendo de uma estrutura específica (previamente definida) e, não sendo considerada a ocorrência de faltas.

(Massoulié et al., 2006) Em [49] são apresentados dois algoritmos de agregação, para estimar o tamanho de uma rede, baseados no método de *random walks*: "Random Tour" e "Sample & Collide".

Random Tour O "Random Tour" baseia-se no envio de uma mensagem a partir de um nó iniciador, executando um *random walk*, até retornar ao nó original (que efectuou o pedido de agregação). Ao longo do *random walk* a mensagem recolhe estatísticas locais que permitem posteriormente estimar em cada nó o tamanho da rede ou o resultado da soma de uma função genérica. Considerando uma função ϕ, o nó iniciador inicializa um contador com o valor ϕ/d_i (em que d_i representa o grau do nó i — número de vizinhos). Cada nó, ao receber a mensagem, incrementa o contador X da mensagem com ϕ/d_i ($X \leftarrow X + \phi/d_i$). A mensagem é reencaminhada sucessivamente para um dos vizinhos escolhido aleatoriamente, até retornar ao nó inicial. Ao receber a sua mensagem, a partir do valor do contador, o nó iniciador calcula

a estimativa $\widehat{\Phi}$ ($\widehat{\Phi} = d_i X$).

Sample & Collide O "Sample & Collide" baseia-se na contagem de amostras aleatórias, inspirado no método do "BdayParadox" [7]. No entanto, com o objectivo de recolher amostras não enviesadas utiliza um algoritmo baseado num *random walk* temporal contínuo (*Continuous Time Random Walk*), obtendo uma aproximação a uma amostragem aleatória uniforme. O algoritmo de amostragem procede da seguinte forma: no nó iniciador é predefinido um valor temporal T, sendo este enviado numa mensagem de amostragem para um vizinho escolhido aleatoriamente; após a recepção da mensagem de amostragem, em cada nó é escolhido um número aleatório U no intervalo $[0, 1]$ e ao valor temporal é subtraído o valor $-\log(U)/d_i$ ($T \leftarrow T + \log(U)/d_i$); o valor temporal actualizado é transmitido sucessivamente na mensagem de amostragem para um nó vizinho escolhido aleatoriamente, até atingir um valor inferior ou igual a zero; quando o valor temporal é inferior ou igual a zero, o nó actual é escolhido como nó amostrado, retornando a sua identificação para o nó iniciador. O algoritmo de estimativa do tamanho da rede, recolhe diversas amostragens (usando o método descrito) até que a mesma amostra seja recolhida repetidamente um número predefinido de vezes l. A qualidade da amostragem está dependente da correcta escolha do valor temporal inicial T. A estimativa do tamanho da rede é calculada pela resolução da Equação 2.4, através de uma pesquisa binária, ou pela Formula 2.5 (mais "leve" computacionalmente), em que C_l é o número de estimativas até que uma esteja repetida l vezes. O parâmetro l (número de repetições da mesma amostra) define a precisão da estimativa, estando a qualidade da estimativa dependente da capacidade do método de amostragem fornecer amostras uniformemente distribuídas. De acordo com os autores, o "Sample & Collide" apresenta maior eficiência que o "Random Tour", no entanto o valor estimado apresenta uma desvio padrão teórico de 10% (em relação ao valor real). Os dois algoritmos podem ser utilizados em ambientes dinâmicos.

$$\sum_{i=0}^{C_l - l - 1} \frac{i}{N-1} - l = 0 \tag{2.4}$$

$$\widehat{N} = C_l^2/2l \tag{2.5}$$

(Mosk-Aoyama and Shah, 2006) Para a computação distribuída de funções separáveis (*separable functions*), desde que estas possam ser escritas como uma combinação linear de funções individuais, em [54] é apresentado um esquema de agregação baseado nas propriedades de variáveis aleatórias exponenciais. Este algoritmo reduz o problema da agregação à determinação do mínimo de um conjunto de valores. Na execução desta técnica é suportado qualquer tipo de protocolo de disseminação, visto o calculo de mínimos ser independente da ordem e da ocorrência de duplicados. Tendo em conta as características de tolerância a faltas e a escalabilidade, os autores consideram a utilização de um protocolo epidémico (*gossip*) para a disseminação da informação. O algoritmo "COMP" processa-se basicamente da seguinte forma: inicialmente, cada nó mantêm um vector de tamanho r e, gera r números aleatórios independentes com uma distribuição exponencial de ratio y_i, em que y_i representa o valor actual de cada nó i (sendo: $y_i \geq 1$); durante o processo de disseminação de informação, cada nó envia o seu vector para os seus vizinhos; ao receber um vector, cada nó i actualiza o seu vector com o mínimo W_l^i entre o valor recebido e o valor existente em cada posição do vector l; a estimativa do valor agregado $\widehat{y_i}$ é dada em cada nó i pela Formula 2.6. Os autores apresentam um estudo teórico exaustivo da performance do algoritmo, sendo este dependente da topologia da rede e tanto mais rápido quanto mais rápido for o protocolo de disseminação. No entanto, não é dada a noção da precisão dos valores estimados, não sendo garantido a obtenção de um valor exacto em todos os nós da rede.

$$\widehat{y_i} = \frac{r}{\sum_{l=1}^{r} W_l^i} \tag{2.6}$$

(Chen et al., 2006) O DRG (*Distributed Random Grouping*) [12] é um algoritmo de agregação para redes de sensores (sem fios). O algoritmo baseia-se na criação probabilística de grupos, com o intuito de permitir a tolerância a faltas e suportar alterações dinâmicas da topologia da rede. O DRG tenta tirar partido da natureza das comunicação sem fios — *broadcast*, em que

todos os nós dentro do raio de alcance de uma transmissão rádio são susceptíveis de a "ouvir". O modo de funcionamento do algoritmo define três estados de execução distintos para cada nó: líder, membro e inactivo. Assim sendo, a execução do algoritmo pode ser dividida em três etapas principais, correspondendo cada uma a um estado possível dos nós. Numa primeira etapa, cada nó no estado inactivo decide independentemente tornar-se líder (com uma probabilidade predefinida) de um grupo e envia uma mensagem (*broadcast*) para todos os seus vizinhos, esperando posteriormente pela resposta dos eventuais membros. Na segunda etapa, os nós no estado inactivo que recebem a mensagem de um líder, respondem ao primeiro líder enviando o seu valor agregado actual e tornam-se membros do seu grupo (mudando de estado). Na terceira fase, ao receber as respostas dos seus membros (com o valor agregado), o líder calcula o novo valor agregado do grupo e envia o resultado (*broadcast*) para os elementos do grupo, voltando ao estado inactivo. Ao receber o resultado do seu líder, os membros do grupo actualizam o seu valor agregado e voltam ao estado inactivo. Ao longo do tempo, a criação aleatória de grupos, com determinação local e centralizada do valor agregado de cada grupo, permite a convergência e a estimativa em todos os nós de um valor agregado global. O desempenho deste algoritmo depende da probabilidade de um nó se tornar líder (capacidade de criar grupos — quantidade e tamanho dos grupos). Além disso, entre outros, para permitir uma completa tolerância a faltas é necessário considerar a definição de alguns tempos de espera máximos na recepção de algumas mensagens (para os líderes pela resposta dos membros e para os membros pelo resultado do líder). Este facto não é abordado pelos autores, apesar de intuitivamente ser possível notar que esses eventuais tempos de espera devem influenciar fortemente o desempenho do algoritmo. Na Secção 4.3 é apresentada informação mais detalhada acerca desta técnica de agregação.

(Baquero et al., 2006) Em [5] é introduzido um algoritmo de agregação (soma de números reais e contagem) distribuído, independente do esquema de comunicação e tolerante a faltas, baseado na teoria dos números extremos para a produção da estimativa. O algoritmo "Extrema Propagation" assenta

na geração de um vector de números reais aleatórios de acordo com uma distribuição especifica (e.g. exponencial ou gaussiana) e na determinação dos valores mínimos dos vectores trocados entre nós vizinhos (semelhante ao algoritmo [54]). A execução do algoritmo processa-se numa sucessão de rondas até convergir (atingir um critério de paragem definido), operando da seguinte forma: inicialmente cada nó gera um vector de K números aleatórios de acordo com uma distribuição especifica (e.g. exponencial com parâmetro $\lambda = 1$ para o caso da estimativa do tamanho da rede, fazendo corresponder o valor de cada nó ao parâmetro de criação da distribuição — λ) e envia-o para todos os seus vizinhos; em cada ronda, ao receber um vector de outro nó, são determinados os valores mínimos (entre o vector local e o vector recebido), sendo o resultado guardado e enviado para todos os vizinhos. Em cada ronda, o vector resultante do cálculo dos mínimos entre vectores é comparado com o vector anteriormente guardado, sendo contabilizada a não ocorrência de alterações. O número de rondas sem alterações no vector é usado como critério de paragem, tendo um valor inicial predefinido, podendo variar de acordo com a topologia e o tamanho da rede. Para calcular a estimativa do valor agregado é utilizada uma função (estimador) obtida a partir da teoria dos números extremos. Em termos genéricos, a função para estimar a soma de valores $\widehat{Sum}$, utilizando um vector de mínimos de tamanho K com número gerados aleatoriamente com distribuição exponencial, é dada pela Formula 2.7. Este método de agregação foca-se na velocidade de convergência e não na precisão, conseguindo produzir a estimativa num número de rondas pouco acima do mínimo teórico (D — diâmetro da rede) necessário para todos os nós possuírem o conhecimento da estimativa. Os passos extras necessários são utilizados para detectar o final do processo de estimativa (embora ela já esteja calculada). A precisão deste algoritmo é controlado pelo tamanho das mensagens trocadas. Por exemplo: para estimar o tamanho de uma rede, utilizando um vector de K números, é possível produzir a estimativa $\widehat{N}$ com um desvio padrão de $N/\sqrt{K-2}$. Neste método o valor agregado é calculado de forma "grosseira" em todos os nós de uma rede estática. Tendo em conta a tolerância a perdas de mensagens e suporte para ligações mais lentas, os autores consideram o uso de um *timeout* (tempo de espera pelas mensagens

dos vizinhos em cada ronda) em conjunto com a possibilidade do algoritmo permitir a falha de um número predefinido de mensagens.

$$\widehat{Sum} = \frac{K-1}{\sum_{l=1}^{N} x[i]} \tag{2.7}$$

2.2.2 Taxonomia

De acordo com as características e modo de execução dos algoritmos existentes, foi elaborada uma taxonomia que classifica os algoritmos considerando dois aspectos fundamentais — comunicação e computação. Do ponto de vista da comunicação, é efectuada uma divisão dos algoritmos considerando a sua dependência de uma estrutura de comunicação específica (árvore, anel ou outra) e o padrão de comunicação utilizado nas trocas de informação necessárias para a sua execução (ver Tabela 2.1). Do ponto de vista da computação, a classificação determina o tipo de agregação efectuada (genérica, contagem ou outra) e o princípio de computação subjacente ao processamento de cada algoritmo (ver Tabela 2.2). Para cada uma das perspectivas consideradas (comunicação e computação) é definida uma classificação dupla (vertical e horizontal), conforme se pode verificar nas respectivas tabelas taxonómicas. A taxonomia apresentada foi elaborada considerando as características de todos os algoritmos anteriormente descritos.

Comunicação

Ao nível da comunicação (Tabela 2.1), a classificação vertical divide os algoritmos em duas categorias:

Sem estrutura — Esta categoria agrupa os algoritmos que não dependem de nenhuma estrutura de rede específica, funcionando em rede com qualquer tipo de topologia.

Com estrutura — Esta categoria reúne os algoritmos que necessitam de uma estrutura de rede e encaminhamento previamente definida para a sua execução. Caso a estrutura não exista, ela deverá ser forçosamente criada antes da execução do algoritmo. Esta dependência condi-

ciona a utilização dos algoritmos em determinadas situações: nomeadamente, se consideramos a utilização destes algoritmos em redes móveis é necessário garantir a adaptação das estruturas de encaminhamento utilizadas ao dinamismo da rede. Além disso, os algoritmos herdam problemas subjacentes ao tipo de estrutura utilizada, por exemplo, numa estrutura em árvore a falha de um único nó implica a perda de comunicação com toda a sua sub árvore. Esta categoria encontra-se dividida em três subgrupos que identificam estruturas de rede específicas: árvore, anel e DHT (*Distributed Hash Tables*).

Tipicamente, na utilização de uma estrutura em árvore, a estimativa do valor agregado é calculada num único nó (raiz), tendo posteriormente de ser propagada para os restantes nós, caso se pretenda que todos possuam o resultado da estimativa.

A estrutura em anel consiste numa relação virtual entre sucessores e predecessores criada de forma aleatória, definido uma relação de dependência entre nós da qual os algoritmos tiram partido.

Para além das duas estruturas de rede anteriormente especificadas, também são indicadas as DHT, pelo facto de ter sido identificado um algoritmo (para estimar o tamanho da rede) utilizado especificamente neste tipo de sistemas.

A classificação horizontal da Tabela 2.1 define o padrão de comunicação utilizado pelos diversos algoritmos de agregação, sendo dividida em quatro grupos distintos:

Flooding/Broadcast — Nesta categoria estão incluídos os algoritmos que difundem informação para todos os elementos da rede, promovendo a participação de todos os nós no processo de agregação. A propagação da informação é iniciada num nó ("raiz" ou "líder"), sendo enviadas mensagens para todos os seus vizinhos — "um para todos". Este tipo de comunicação induz uma elevada carga na rede durante o processo de agregação, envolvendo em alguns casos uma certa centralização da informação trocada.

Random walks — Este padrão de comunicação está normalmente associado à recolha de amostras na rede, implicando apenas um envolvimento parcial dos elementos da rede no processo de agregação. A difusão da informação assemelha-se à circulação de um testemunho. As mensagens são enviadas de nó em nó, sendo o processo de comunicação iniciado num nó que transmite a informação apenas para um dos vizinhos escolhido — "um para um", até atingir um critério de paragem predefinido (número máximo de saltos, selecção de um nó anteriormente escolhido ou regresso ao nó inicial). A quantidade de trocas de mensagens associada aos algoritmos deste grupo é normalmente reduzida, não existindo um envolvimento de toda a rede no processo de agregação. Devido a esse envolvimento parcial, os algoritmos deste grupo utilizam normalmente processos probabilísticos para produzir as suas estimativas, apresentando um nível de erro conhecido constante, não convergindo ao longo do tempo para o valor agregado exacto.

Gossip — Este tipo de comunicação é também conhecida por comunicação epidémica, devido ao método de disseminação da informação ser inspirado no modo de propagação de uma epidemia. Assim sendo, um nó inicial ("infectado") envia a sua mensagem para um subconjunto dos seus vizinhos escolhidos aleatoriamente ("contagiados") — "um para alguns", passando esses últimos a repetir o processo de propagação da informação, acabando por envolver quase a totalidade da rede no processo. Este padrão de comunicação permite uma difusão robusta (tolerante a faltas) e escalável da informação por toda a rede, de forma totalmente distribuída. O facto deste modelo de comunicação ser por natureza tolerante a faltas, não implica que os algoritmos de agregação que o utilizam também o sejam, vistos estes poderem definir princípios e invariantes que o padrão de comunicação não consegue por si só garantir. Este modelo de comunicação, não é tão "pesado" quanto ao *flooding*, conseguindo no entanto atingir níveis próximos de difusão da informação, em termos de velocidade de propagação e abrangência da rede.

Entrada/Saída — Esta categoria agrupa alguns casos específicos, onde a computação e manutenção da agregação são efectuadas apenas aquando a entrada/saída de nós na rede. Os algoritmos deste grupo são extremamente económicos em termos de comunicação, não sendo efectuado qualquer tipo de disseminação de informação a fim de calcular o valor agregado. No entanto, as estimativas produzidas por estes algoritmos são pouco exactas e dispersas ao longo da rede, podendo existir uma grande variação entre os valores estimados por cada nó.

Computação

Do ponto de vista da computação (Tabela 2.2), a classificação vertical divide os algoritmos de acordo com o tipo de funções agregadas possíveis de calcular, ou seja, de acordo com a sua versatilidade e abrangência em termos computacionais. Neste contexto, definiram-se principalmente dois grupos: "Agregação" e "Contagem", sendo ainda criado um terceiro grupo "Outros" para incluir algumas situações particulares. Os algoritmos do grupo "Agregação" são os mais genéricos, permitindo calcular diversas funções de agregação: médias, contagens, máximos, mínimos e somas. Os algoritmos de "Contagem" apenas permitem efectuar contagens, como por exemplo, determinar o número de elementos numa rede ou contar o número de votos. O grupo "Outros" é constituído por um único caso particular, desenhado para um cenário muito especifico e utilizado apenas para calcular uma média (arredondada ao valor inteiro).

A classificação horizontal da Tabela 2.2 divide os algoritmos de agregação em seis grupos principais, de acordo com os princípios e fundamentos computacionais usados no processo de agregação:

Averaging — Neste grupo estão incluídos os algoritmos que se baseiam no principio da conservação da massa para a concretização do processo de agregação. A computação efectuada consiste numa simples divisão do valor de cada nó (massa) e repartição desse valor pelos seus vizinhos e ele próprio. A realização continua deste procedimento por parte de todos os elementos da rede provoca uma distribuição do valor global da

rede, fazendo com que todos os valores convirjam para o valor médio da rede, daí a designação *averaging*. O valor médio para o qual todos os nós convergem progressivamente ao longo do tempo corresponde ao valor agregado exacto. Em termos computacionais este processo consiste essencialmente numa simples divisão aritmética, sendo de rápida execução e utilizando normalmente reduzidos recursos de processamento e memória. A execução desta técnica permite obter estimativas precisas em todos os nós da rede, praticamente exactas, desde que o processo seja executado durante um tempo mínimo necessário (número de iterações), tendo em consideração as características da rede (tamanho, grau de conectividade e topologia) e o padrão de comunicação utilizado na transmissão dos valores parciais (massa). A robustez deste método está fortemente ligado a sua capacidade de conservar a "massa" global do sistema. A perda de "massa" provocada pela falha de nós ou perdas de mensagens origina um desvio no valor para o qual os nós convergem, introduzindo um factor de erro no valor estimado (correspondente à alteração da "massa" global verificada). Como tal, neste tipo de algoritmos é importante ter em atenção o princípio da conservação da massa e assegurar a sua subsistência, sendo este o principal invariante do sistema.

Fusão de itens — Esta categoria inclui os algoritmos que executam o processo de agregação baseado na fusão (agregação) de um conjunto de dados (itens) armazenados em memória (*cache* ou base de dados) e trocados entre os vários elementos da rede. Esta técnica requer maiores recursos ao nível da capacidade de memória, sendo guardada a informação de vários nós e o resultado da troca de itens entre si: bases de dados de valores distintos trocados com todos os vizinhos ou *cache* de itens de nós distintos (limitada a um valor predefinido, sendo a informação mais antiga descartada). Em termos computacionais, a fusão dos itens pode consistir na aplicação directa de uma função de agregação, obtendo um valor parcial para cada item, ou simplesmente no armazenamento dos valores distintos de cada item (sendo posterior-

mente determinado o valor agregado à partir dos dados armazenados, depois de atingido um número predefinido de trocas de informação).

Amostragem — Neste grupo, a determinação da estimativa do valor agregado é efectuada através da recolha de amostras dos valores existentes na rede e da posterior aplicação de um método probabilístico específico. Os algoritmos deste grupo são fortemente influenciados pelo método probabilístico utilizado na computação do valor agregado, herdando as suas propriedades. Assim sendo, a precisão da estimativa obtida corresponde essencialmente à do método probabilístico aplicado. O valor das estimativas não converge ao longo do tempo, apresentando um factor de erro constante e equivalente ao do processo probabilístico utilizado. Conforme se pode verificar na Tabela 2.2, os algoritmos deste grupo ficam limitados ao cálculo de uma função de agregação específica — contagem. Esta categoria está subdividida em vários grupos de acordo com o método probabilístico aplicado ou o valor amostrado: *Birthday Problem* [14] — problema probabilístico baseado na probabilidade de ocorrência de uma repetição numa determinada população (e.g. num grupo de 23 pessoas escolhidas ao acaso, a probabilidade de duas delas terem nascido no mesmo dia do ano é de 50%, sendo superior a 99% para um grupo de 57 pessoas); *Capture-recapture* [60] — método probabilístico baseado na recolha de múltiplas amostras de uma população fechada (assume que o tamanho da população se mantêm constante durante todo o processo) e contagem do número de elementos repetidos nas amostras "capturadas"; Distância (saltos) — utilização da distância ao nó iniciador para determinar o tamanho da rede através de um processo probabilístico simples; Mensagens enviadas/recebidas — utilização da contagem do número de mensagens recebidas ou enviadas (com sucesso num cenário especifico: rede sem fios com ocorrência de colisões de mensagens enviadas em simultâneo) para estimar o tamanho da rede através de uma função probabilística básica. Tipicamente, as amostras são recolhidas num único nó, sendo este o responsável pela aplicação do respectivo método probabilístico e calculo da estimativa da

contagem, podendo posteriormente enviar o resultado para os restantes nós da rede.

Contagem — Esta categoria inclui os algoritmos que recorrem a realização de uma contagem (incremento/decremento de um contador ou estimador) para determinar a estimativa do valor agregado. De acordo com o tipo de contagem praticada foi efectuada uma divisão em três subgrupos: Determinística — contagem efectiva do número de elementos; Estimador — contagem realizada através do incremento/decremento de um estimador; Probabilística — contagem efectuada através de uma técnica probabilística, neste caso a contagem de Flajolet e Martin (FM) [20, 21]. Tendo em conta a sua natureza, a utilização deste tipo de técnicas está orientada à concretização de um tipo de agregação específico — contagem, o que se verifica efectivamente para quase todos os subgrupos (Determinística e Estimador) desta categoria com a excepção de um subgrupo (Probabilística FM). Isto deve-se ao facto dos algoritmos do último subgrupo utilizarem a contagem probabilística FM não com o principal objectivo de contar elementos, mas sim com o de detectar duplicados.

A contagem determinística é efectuada através da circulação de um testemunho por toda a rede, que vai registando continuamente a sua passagem por cada nó (actualização de contadores), implicando o registo de dados para a rede toda. No caso de uma rede de grandes dimensões, o armazenamento efectuado no testemunho pode corresponder a uma quantidade considerável de informação, sendo esta transmitida de nó em nó, podendo consistir um problema em termos de desempenho. Além disso, todo o processo está dependente da existência e integridade do testemunho. Caso o testemunho seja perdido devido a ocorrência de uma única falha, todo o processo será comprometido e terá de ser reiniciado. Por outro lado, basta que o testemunho passe uma única vez por todos os nós da rede, para que seja possível produzir um estimativa exacta, existindo no entanto outro problema: como é que se pode garantir que de facto o testemunho passou por todos os nós?

A utilização de um estimador é efectuada num cenário especifico, sendo determinada a estimativa do tamanho de uma rede (contagem) apenas através do registo da entrada/saída de nós da rede (incremento/decremento de uma função estimadora). Em termos de recursos computacionais, esta técnica é pouco exigente, requerendo apenas a capacidade de execução da função estimadora (incremento/decremento) e o armazenamento do valor estimado. Apesar da sua simplicidade, esta técnica produz estimativas pouco precisas e dispersas ao longo de todos os nós da rede.

A contagem probabilística FM permite estimar o número de elementos distintos guardados, sendo utilizada normalmente não com o objectivo de concretizar uma contagem, mas sim com o intuito de resolver o problema da ocorrência de duplicação de valores em funções de agregação não idempotentes. Do ponto de vista computacional, a contagem FM requer a manutenção de uma estrutura especifica (mapa de bits) para a distinção de valores guardados (*hash* dos valores). À semelhança dos restantes métodos probabilísticos, o FM introduz um factor de erro na estimativa dos valores distintos, sendo este por conseguinte propagado no cálculo do valor agregado final.

Densidade de Identificadores — Os algoritmos pertencentes a este grupo apenas efectuam contagens, estimando o tamanho da rede com base na medição da densidade dos identificadores únicos de cada nó, recorrendo a utilização de métodos probabilísticos. A aplicação desta técnica assume a existência de um universo de identificadores (aleatórios) uniformemente distribuídos. Do ponto de vista computacional, esta técnica pode implicar a execução de procedimentos semelhantes aos que são executados na contagem probabilística FM (nomeadamente, a aplicação de uma função de *hash* aos identificadores e a determinação do número de elementos distintos) ou a simples medição da distância entre identificadores sucessivos.

Redução Min./Max. — Nesta categoria, a computação do agregado é reduzida à determinação do máximo/mínimo de um vector de valores

aleatórios, gerados de acordo com uma distribuição especifica (e.g. exponencial ou gaussiana) utilizando como parâmetro o valor actual de cada nó. Ao longo da execução, o processo de agregação resume-se à determinação dos mínimos ou máximos dos vectores trocados entre nós. A estimativa do valor agregado é produzida, pela aplicação de uma formula derivada da teoria dos números extremos. Em termos computacionais, a execução deste tipo de algoritmos implica a existência de um gerador de números aleatórios, capacidade aritmética para o cálculo da estimativa e determinação de mínimo/máximos e, capacidade de memória para armazenar um vector de números. A precisão da estimativa varia de acordo com o tamanho do vector utilizado, quanto maior for o vector mais precisa será a estimativa produzida, implicando a utilização de maiores recursos de memória e tempo de processamento do vector. Este tipo de algoritmo é extremamente rápido, dependendo do protocolo utilizado na disseminação dos vectores, conseguindo produzir estimativas num número de iterações pouca acima do mínimo teórico (diâmetro da rede).

Tabela 2.1: Classificação do ponto de vista da comunicação

	Sem Estrutura	**Com Estrutura**		
		Arvore	**Anel**	**DHT**
Flooding / Broadcast	Propagate2All [7] RandomizedReports [7] (Kutylowski et al., 2003) [41] (Kutylowski et al., 2002) [35] DRG [12] Interval Density [40, 57] (Ahmed et al., 2006) [3] COMP [54] Extrema Propagation [5] *Flow Updating Broadcast*	(Madden et al. 2002) [44] SingleTree [7] MultipleTrees [7] (Considine et al., 2004) [13] (Li et al., 2005) [43] I-LEAG [8]		
Random Walk	(Dolev et al. 2006) [16] BdayParadox [7] RandomIncWalk [7] (Mane et al., 2005) [46] Random Tour [49] Sample & Collide [49] Interval Density [40, 57] (Ahmed et al., 2006) [3]			
Gossip	(Jelasity and PreuB, 2002) [32] Push-Sum Protocol [38] (Jelasity et al., 2004) [29] Push-Pull Gossiping [31] Hop Sampling [40, 57] Interval Density [40, 57] (Ahmed et al., 2006) [3] COMP [54] Extrema Propagation [5] *Flow Updating*		(Ghdsi et al., 2005) [23]	
Entrada/Saída			(Horiwitz and Malkhi, 2003) [28]	(Gurmeet and Manku, 2003) [48]

Tabela 2.2: Classificação do ponto de vista da computação

		Agregação	**Contagem**	**Outros**
Averaging		(Madden et al. 2002) [44] SingleTree [7] Push-Sum Protocol [38] Push-Pull Gossiping [31] (Li et al., 2005) [43] I-LEAG [8] Random Tour [49] DRG [12] *Flow Updating* *Flow Updating Broadcast*		(Kutylowski et al., 2003) [41][a]
Fusão de itens	**Base de Dados**	(Jelasity and PreuB, 2002) [32]		
	Cache	(Jelasity et al., 2004) [29]		
Amostragem	**Birthday Problem**[b]		BdayParadox [7] Sample & Collide [49]	
	Capture-recapture		(Mane et al., 2005) [46]	
	Distância (saltos)		Hop Sampling [40, 57] RandomIncWalk [7]	
	Mensagens enviadas/recebidas		RandomizedReports [7] (Kutylowski et al., 2002) [35]	
Contagem	**Determinística**		(Dolev et al. 2006) [16]	
	Estimador ($\log n_0$)		(Horiwitz and Malkhi, 2003) [28]	
	Probabilística (FM) [20] [21]	Propagate2All [7] MultipleTrees [7] (Considine et al., 2004) [13] (Ahmed et al., 2006) [3]	(Gurmeet and Manku, 2003) [48]	
Densidade de Identificadores			Interval Density [40, 57] (Ghdsi et al., 2005) [23]	
Redução Min/Max		COMP [54] Extrema Propagation [5]		

[a] Estima apenas a média (arredondada ao valor inteiro) de valores existentes em cada nó de uma rede de sensores *single-hop* com R canais de comunicação independentes.

[b] Introduzido por Von Mise em 1932 como referido por Anirban DasGupta em [14], que também apresenta uma revisão contemporânea do "birthday problem" assim como algumas extensões e variantes como "the strong birthday problem"

Capítulo 3

Flow Updating

Neste capítulo é exposta a principal contribuição deste estudo, sendo apresentado um novo algoritmo de agregação — *Flow Updating*, constituído essencialmente por duas componentes que o distingue dos restantes algoritmos da mesma categoria: troca e actualização de fluxos (em vez das trocas de "massa"); *broadcast* de actualizações (em vez da habitual troca de mensagens par a par). Estes conceitos são devidamente aprofundados e descritos ao longo deste capítulo: inicialmente é efectuada uma abordagem global das principais noções envolvidos na concepção do *Flow Updating* e diferenças relativamente aos restantes algoritmos de *Averaging*; finalmente é feita uma descrição mais pormenorizada de duas versões do novo algoritmo (*broadcast* e *unicast*) efectivamente implementadas para efeitos de simulação e comparação com outros.

Como foi possível verificar na secção anterior (2.2), existe uma grande variedade de algoritmos de agregação com diferentes características: exactidão (precisão da estimativa obtida); velocidade (tempo até atingir a estimativa); robustez (capacidade de tolerar faltas); flexibilidade (capacidade de adaptação ao dinamismo da rede) e escalabilidade. No entanto, não existe neste momento nenhum algoritmo que consiga conjugar todas as referidas características num nível óptimo. Tipicamente, algoritmos flexíveis e robustos produzem estimativas menos precisas, pelo contrário, algoritmos que produzem estimativas quase exactas são mais lentos e menos tolerantes à ocorrência

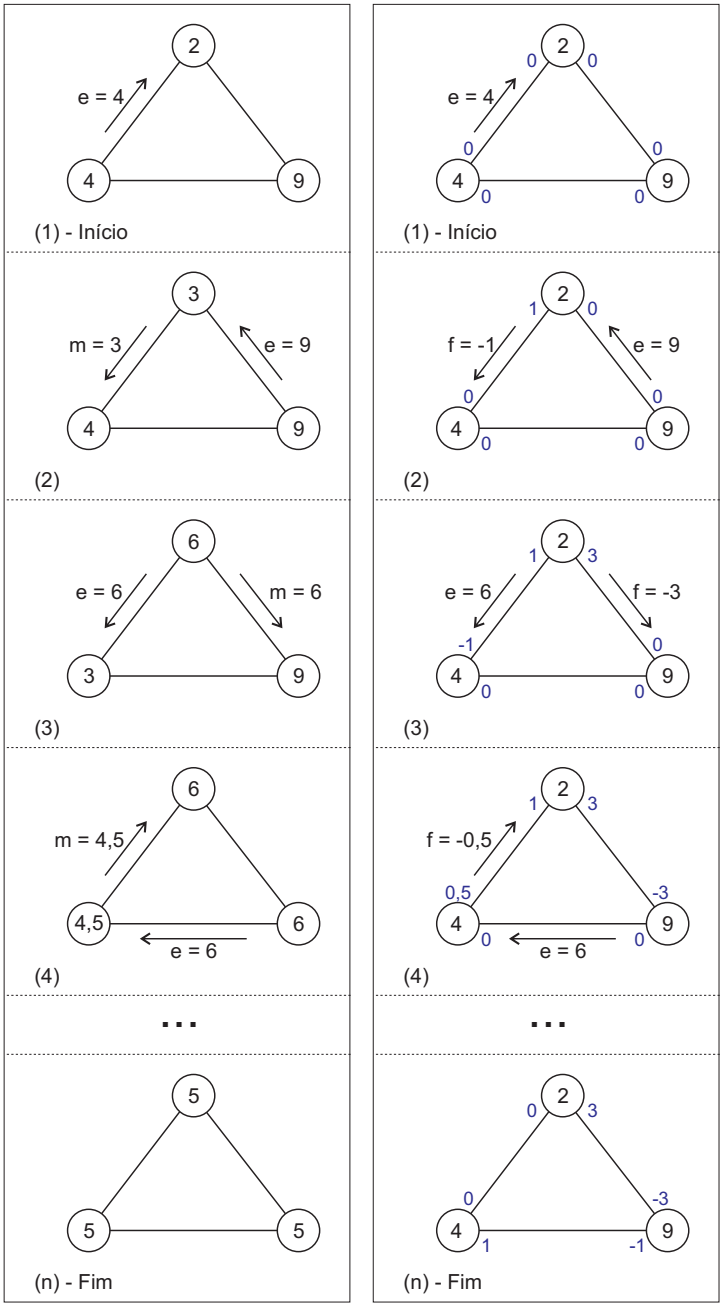
de faltas e alterações na rede, sendo o seu desempenho global gravemente afectado nessas circunstâncias. Considerando este facto, o novo algoritmo foi desenvolvido com o objectivo de conjugar a capacidade de obtenção de estimativas precisas com a capacidade de tolerar faltas sem necessidade de recorrer à utilização de técnicas (uso de *timeouts* e recepção de mensagens de confirmação) que afectam o desempenho do mesmo (ver Capítulo 5).

O algoritmo proposto apresenta-se como uma nova variante dos algoritmos de *Averaging*. A ideia principal do *Flow Updating* baseia-se no registo e actualização de contribuições trocadas entres nós da rede — fluxos[1], em vez da habitual actualização directa do valor agregado efectuada pelos algoritmos desta categoria. Enquanto que noutros algoritmos o resultado da estimativa do valor agregado é normalmente representado por uma simples variável de estado, sendo este alterado directamente ao longo da execução do algoritmo (ver Figura 3.1(a)), no *Flow Updating* o resultado da estimativa é sempre calculado a partir do valor inicial e da informação dos fluxos, sendo apenas estes últimos actualizados durante a execução do algoritmo (ver Figura 3.1(b)). No *Flow Updating*, em cada nó, a estimativa do valor agregado é dada pelo resultado da soma do valor inicial com todos os fluxos registados localmente (Formula 3.1).

$$\widehat{v} = v_0 + \sum_{i=1}^{n} \mathcal{F}_i \tag{3.1}$$

A Figura 3.1 ilustra a principal diferença entre a nova abordagem e o método tipicamente usado nos algoritmos de *Averaging*. Ambas as figuras representam um processo de cálculo da média de um grafo, constituído por três nós com valores iniciais diferentes (4, 2, 9), utilizando o mesmo esquema de troca de mensagens entre nós. Em ambas as situações, a agregação é efectuada através de um processo iterativo de actualização de estimativas par a par, semelhante ao método utilizado no *Push-Pull Gossiping* [31]: um nó envia a sua estimativa actual para um vizinho, que lhe responde com a informação necessária para que ambos assumam o novo resultado da média

[1] Designação utilizada em analogia ao conceito de fluxos de rede [4] na teoria de grafos [9, 26]



(a) Processo de *Averaging* com troca de "massa"

(b) Processo de *Averaging* com actualização de fluxos

Figura 3.1: Diferença entre algoritmos com troca de "massa" e actualização de fluxos.

das estimativas entre eles. Ao fim de n iterações todos os nós estimam o valor correcto da média do grafo (5). Note-se que em ambos os casos a estimativa produzida no final é igual, no entanto, existe uma importante diferença ao nível da informação que é guardada e actualizada. As figuras exibem a evolução do processo de agregação, representando o estado local de cada nó e as mensagens em transito.

A Figura 3.1(a) representa de forma genérica o conceito de troca de "massa", tipicamente usado nos algoritmos de *Averaging*. No método apresentado a soma da estimativa recebida com o valor local é dividida em duas parcelas iguais, sendo uma delas usada para actualizar o valor local e a outra enviada como resposta ao nó vizinho. Ao receber uma mensagem de resposta a estimativa enviada, o nó actualiza directamente a sua estimativa local de acordo com o valor recebido. Este método baseia-se na efectiva distribuição da "massa" de cada nó pelos seus vizinhos na determinação do valor agregado final, conservando a massa global do sistema.

A Figura 3.1(b) representa a ideia base do *Flow Updating*, ilustrando o processo de troca e actualização de fluxos. Neste método o valor local (inicial) é conservado, sendo apenas alterado o valor do fluxo mantido localmente para cada nó vizinho. Inicialmente, todos os fluxos têm o valor zero. Ao receber uma mensagem com a estimativa de um vizinho, o nó determina o valor da nova estimativa e calcula o valor do fluxo que deve existir entre si e o seu vizinho, sendo assumido localmente um valor de fluxo simétrico (sinal inverso) ao valor enviado. Cada fluxo expressa uma contribuição positiva ou negativa, correspondendo a um ganho ou perda relativamente ao valor local para atingir o valor agregado global (a cada ganho está associada uma perda do mesmo valor e vice-versa). Ao receber uma mensagem, o nó actualiza apenas o valor do fluxo local de acordo com o valor recebido, sobrepondo-se este último ao valor guardado para o respectivo vizinho. À semelhança do método anterior, este processo respeita o princípio da conservação da massa[2], no entanto não efectua nenhuma distribuição efectiva de "massa", mas sim

[2]Uma analogia ao que é conhecido na química como a Lei de Lavoisier, sendo este princípio publicado pela primeira vez na dissertação de Mikhail Lomonosov "Reflexion on the solidity and fluidity of bodies" em 1760.

uma distribuição das variações que tem de ser aplicadas a esta última para obter o resultado pretendido.

Com base no novo conceito do *Flow Updating* (actualização e troca de fluxos) é possível implementar novos algoritmos com características semelhantes as dos algoritmos de *Averaging* existentes, tal como a convergência em todos os nós da rede para um valor exacto, mas com capacidade de tolerar perdas de mensagens (como é descrito posteriormente neste capítulo).

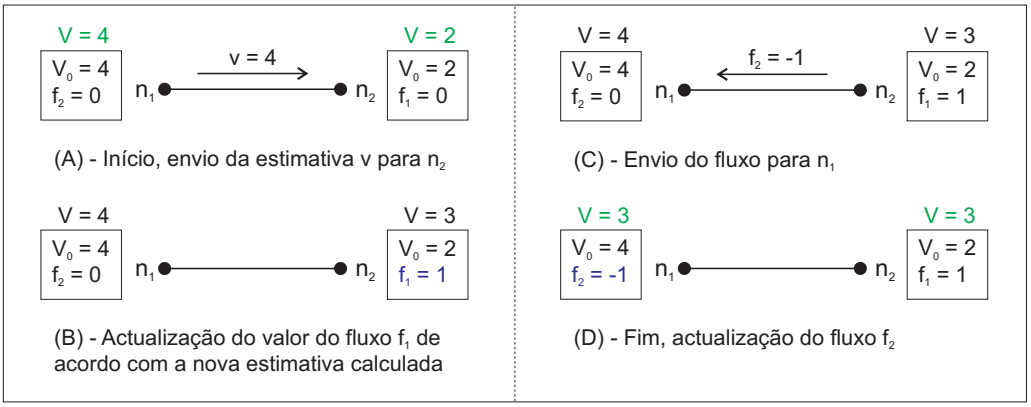


Figura 3.2: Exemplo de utilização do conceito do *Flow Updating*

Na Figura 3.2 é exibido mais pormenorizadamente um exemplo de execução de uma possível aplicação da ideia principal do *Flow Updating*, sendo concretizada num algoritmo com trocas de mensagens *push-pull* (semelhante ao *Push-Pull Gossiping*). Neste exemplo, são considerados apenas dois nós n_1 e n_2 com valores iniciais diferentes (respectivamente, $v_0 = 4$ e $v_0 = 2$) e fluxos com o valor zero ($f_2 = 0$ e $f_1 = 0$), para os quais se pretende calcular a média dos valores iniciais. O processo de agregação ilustrado é executado iterativamente através da troca bidireccional par a par de mensagens: uma estimativa (*push*) e a correspondente actualização de fluxo (*pull*). Assim sendo, o exemplo da Figura 3.2 processa-se da seguinte forma: no primeiro passo (A), o nó n_1 envia o valor da sua estimativa (calculada pela Formula 3.1) para o nó n_2; no segundo passo (B), ao receber a estimativa de n_1, o nó n_2 calcula o valor da nova estimativa (de acordo com o valor recebido e a estimativa local — $(4 + 2)/2 = 3$), determina o novo valor do fluxo f_1

necessário para produzir a nova estimativa e actualiza o seu fluxo com o novo valor ($f_1 = 1$); no terceiro passo (C), o nó n_2 envia para o nó n_1 o valor do fluxo calculado anteriormente com o sinal invertido; no último passo (D), ao receber o fluxo de n_2 em resposta a sua estimativa, o nó n_1 actualiza o valor do seu fluxo ($f_2 = -1$). No final da iteração representada, ambos os nós produzem a mesma estimativa final ($v = 3$), convergindo para o valor exacto e mantendo a massa do sistema — a soma das estimativas iniciais é igual a soma das estimativas finais ($4 + 2 = 3 + 3$).

Como foi possível observar nos exemplos descritos (Figuras 3.1(b) e 3.2), o *Flow Updating* apoia-se no principio da conservação da massa, para a produção de sucessivas estimativas e trocas de fluxos com o objectivo de produzir um valor único em todos os nós da rede. De forma a garantir a convergência para um valor único correcto (média de todos os valores da rede), é essencial manter ao longo da execução de qualquer algoritmo que assente no principio da conservação da massa o seguinte invariante: "A soma dos valores estimados por todos os nós da rede é constante".

Nos algoritmos que realizam uma efectiva distribuição de "massa" (Figura 3.1(a)), tais como o *Push-Sum Protocol* [38] e o *Push-Pull Gossiping* [31], a perda de uma mensagem com informação da "massa" transmitida afecta-os gravemente, provocando uma quebra do invariante e por conseguinte um desvio no valor global para o qual os nós vão passar a convergir. Desta forma, estes algoritmos necessitam de considerar a utilização de mecanismos auxiliares de detecção e recuperação de mensagens perdidas, a fim de garantir a manutenção do invariante e consequente convergência para o valor correcto. Ao contrário desses algoritmos, o *Flow Updating* é imune a perda de mensagens, não necessitando de mecanismo auxiliares, sendo recuperado o invariante logo que é efectuado o envio de um fluxo com sucesso entre os nós onde ocorreu a perda (independentemente do sentido), provocando apenas um pequeno atraso na velocidade de convergência do algoritmo (ver Secção 5.4).

A Figura 3.3 demonstra a imunidade do *Flow Updating* perante a ocorrência de perdas de mensagens. No exemplo apresentado não é considerada a perda das mensagens com a estimativa — *push*, visto que a perda deste

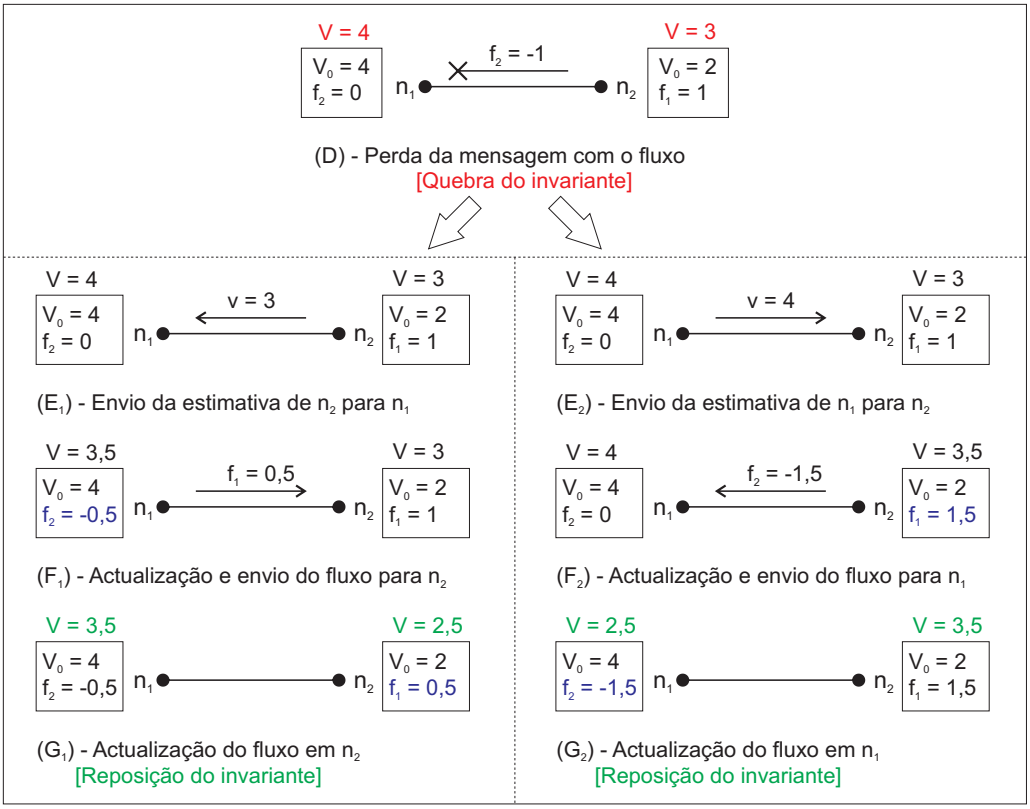


Figura 3.3: Imunidade a perdas de mensagens do *Flow Updating*

tipo de mensagens não provoca qualquer alteração no estado dos nós, sendo equivalente a não ocorrência da iteração. Na Figura 3.3 é apresentada a continuação da execução do exemplo exposto na Figura 3.2, considerando a perda da mensagem com a informação do fluxo no último passo (D). Como é possível verificar, a ocorrência da perda do fluxo enviado (f_2) provoca uma quebra do invariante ($4 + 2 \neq 4 + 3$). No entanto, não é tomada qualquer tipo de acção especifica perante essa ocorrência, continuando simplesmente a execução normal do algoritmo. De forma a demonstrar a recuperação do invariante do sistema independentemente do sentido da comunicação, são exibidas duas alternativas de execução do algoritmo com trocas de mensagens em sentidos opostos. Ambas as alternativas efectuam uma segunda iteração do algoritmo: nos passos (E_1 e E_2) um dos nós envia a sua estimativa para o outro — *push*; nos passos (F_1 e F_2), o destinatário determina e actualiza

o novo valor do fluxo, de acordo com a média da estimativa dos dois nós, e envia o valor simétrico do fluxo em resposta a estimativa recebida — *pull*; nos passos (G_1 e G_2), o nó recebe com sucesso o valor do fluxo em resposta a sua mensagem, actualizando localmente o seu fluxo com o valor recebido. No final, após a concretização com sucesso de uma única iteração, o invariante é recuperado ($4+2=3,5+2,5$; $4+2=2,5+3,5$). Embora as estimativas em ambos os nós não correspondam ao valor correcto, apesar da perda de mensagem, ambos se aproximaram do valor correcto da média (sendo atingido na próxima iteração do algoritmo). Como foi possível verificar, a troca e actualização de fluxos ao invés da efectiva troca de "massa", permite ao *Flow Updating* ser imune a perdas de mensagens. Isto deve-se ao facto das actualizações de fluxo serem idempotentes, estando o seu valor apenas dependente da última mensagem de actualização de fluxo enviada/recebida, sobrepondo-se às anteriores. A perda de mensagens pode provocar uma assimetria nos valores dos fluxos entre os nós (*entrada* $\neq$ *saída*), sendo o equilíbrio dos fluxos recuperado logo que é trocado com sucesso um fluxo no mesmo canal de comunicação onde ocorreu a perda.

Diversas variantes do *Flow Updating* podem ser implementadas, usando esquemas de troca de mensagens distintos, sendo anteriormente referenciado a titulo de exemplo um esquema de iteração bidireccional par a par — *push-pull*. Independentemente do esquema de comunicação utilizado, o principal objectivo deste algoritmo consiste em permitir a convergência de todos os nós da rede para um valor único, de forma eficiente e robusta, dirimindo as assimetrias dos valores locais (iniciais) entre cada nó, através do registo de fluxos trocados entre nós vizinhos (nós directamente conectados). Em cada nó é mantido um registo individual actualizado da última contribuição trocada com cada um dos seus vizinhos, representando as diferentes contribuições de cada nó na convergência para um valor único, fruto das diferenças existentes entre os valores iniciais de cada um e da evolução das suas estimativas de acordo com as contribuições enviadas/recebidas. Tendo em conta tudo isto, foi efectuada uma implementação especifica do *Flow Updating*, com o intuito de aumentar a performance do processo de agregação, sendo esta detalhada de seguida (Secção 3.1).

3.1 Algoritmo com *broadcast*

Nesta secção é efectuada a descrição detalhada do *Flow Updating Broadcast*, sendo exposto na Tabela 3.1 o pseudo código do algoritmo executado em cada nó (de acordo com a implementação concretizada para simulação e comparação com outros algoritmos — ver Capítulo 4). O *Flow Updating* assume que cada nó conhece o conjunto dos seus vizinhos (nós com os quais está directamente conectado), necessitando de distinguir cada um dos seus vizinhos (bastando o uso de identificadores locais)[3], para guardar o valor do fluxo de cada um. Visto que cada nó conhece os seus vizinhos, é desejável que após a produção de uma nova estimativa todos os valores locais (fluxos de todos os vizinhos) sejam determinados e actualizados de imediato. Isto permite ao nó assumir prontamente o novo resultado da estimativa (através da aplicação da Formula 3.1 com todos os fluxos actualizados) e, possibilita o envio imediato de todos os valores dos fluxos actualizados para os seus vizinhos. Desta forma, para tirar partido do modo de comunicação utilizado em redes sem fios — *Broadcast* — e da possibilidade de actualização de vários vizinhos em simultâneo (com uma única mensagem), foi primeiramente pensada a implementação do *Flow Updating* com *broadcast*.

O algoritmo é executado periodicamente e continuamente pelos nós da rede, estando a sua execução dividida num conjunto de acções associadas a quatro evento específicos: "On Init" — evento de inicialização do algoritmo, "On Tick" — evento de activação periódico (uma vez por ronda), "On Receive" — evento associado a recepção de mensagens e, "On Estimate" — evento associado a solicitação do valor estimado.

O evento "**On Init**" corresponde a iniciação do algoritmo, em que o vector para guardar as estimativas dos vizinhos $\mathcal{V}$ e o vector dos fluxos $\mathcal{F}$ enviados e recebidos são inicializados com o valor zero.[4] O valor agregado inicial v_0 é atribuído pela função InitValue() dependendo do tipo de agregação

[3]Não é necessário garantir a existência de identificadores únicos em toda a rede, apenas locais, o que possibilita a utilização de um simples esquema de geração aleatório de ID (tendo em conta algumas considerações apresentadas em [33]).

[4]Por questões de simplicidade, é considerado que cada posição dos vectores está associada a único nó vizinho, correspondendo a sua identificação.

Tabela 3.1: Algoritmo — *Flow Updating Broadcast*

On Init:

1. Set initial values:

$\mathcal{V} \leftarrow 0$

$\mathcal{F} \leftarrow 0$

$v_0 \leftarrow$ InitValue()

On Tick:

1. Get neighbors to update:

$I \leftarrow$ GetNeighbors()

2. Send adjusting, if any neighbor found ($I \neq \phi$):

2.1. Calculate new estimated value v':

$v' \leftarrow (\text{Estimate}() + \sum_{i=1}^{n} \mathcal{V}_i)/(n+1)$

2.2. For all neighbors (i) in I do:

2.2.1. Calculate and store neighbor flow $\mathcal{F}_i$:

$\mathcal{F}_i \leftarrow \mathcal{F}_i - (v' - \mathcal{V}_i)$

2.2.2. Store new estimated value v':

$\mathcal{V}_i \leftarrow v'$

2.3. Broadcast message $(v', \{(i, -\mathcal{F}_i) | i \in I\})$.

On Receive:

1. Get message data $(v', \mathcal{F}')$.

2. Get message sender Id "i".

3. Store flow value $\mathcal{F'}_j$ associated to node "j":

$\mathcal{F}_i \leftarrow \mathcal{F'}_j$

4. Store estimated value v':

$\mathcal{V}_i \leftarrow v'$

On Estimate:

1. Return value:

return $v_0 + \sum_{i=1}^{n} \mathcal{F}_i$

(contagem ou média) que se pretende executar. No caso da contagem, para estimar o tamanho da rede, se ao v_0 for atribuído o valor 1 em apenas um dos nós e o valor 0 aos restantes, após a execução do algoritmo o valor da estimativa do tamanho da rede (contagem) é dado em cada nó por $1/v$. No

caso da média, por exemplo para estimar a temperatura média do ambiente onde se encontram todos os nós, ao v_0 é atribuído o valor da temperatura local lida por cada nó, após a execução do algoritmo o valor da estimativa da temperatura (média) é dado em cada nó pelo valor v (calculado pela Formula 3.1).

O evento "**On Tick**" é despoletado periodicamente, estando associado a execução de um conjunto de acções por todos os nós. Em cada ronda, cada nó determina de acordo com a informação das suas estimativas locais se é necessário propagar alguma actualização. Desta feita, cada nó verifica se a estimativa dos seus vizinhos guardada localmente corresponde à sua própria estimativa do valor agregado, identificando todos os vizinhos com estimativas diferentes (em vez da simples selecção de todos os vizinhos).

O processo de verificação e identificação dos vizinhos é efectuado pela função GetNeighbor() (ponto 1), sendo seleccionado o conjunto dos vizinhos I com estimativa diferente relativamente à média das estimativas actuais. Esta optimização na implementação da função GetNeighbor() tem como objectivo evitar o envio de mensagens entre nós com o mesmo valor. Na prática, no processo de estimativa do tamanho da rede, em que inicialmente apenas um nó tem um valor diferente (1) e os restantes o mesmo valor (0), esta optimização permite poupar o envio de uma boa percentagem de mensagens nas primeiras rondas (dependendo do tamanho da rede) — justificando a sua utilização, tendo durante o resto da execução do algoritmo um comportamento equivalente a selecção de todos os nós vizinhos. Caso não seja encontrado nenhum nó com diferente estimativa, a função retorna um conjunto vazio; caso seja retornado as identificações de um conjunto válido de vizinhos desactualizados é executado um conjunto de acções para proceder a actualização dos mesmos (ponto 2).

No processo de actualização, primeiro é calculada a nova estimativa do valor agregado v' (ponto 2.1, em que n representa o número de vizinhos), de seguida para cada um dos nós pertencentes ao conjunto é calculado e registado (no vector de fluxos) o valor do fluxo $\mathcal{F}_i$ (ponto 2.2.1) necessário para que estes possam convergir para a mesma estimativa. Para todos eles é também registada localmente a nova estimativa do valor agregado, no vector dos val-

ores estimados, sendo associada ao nó respectivo $\mathcal{V}_i$ (ponto 2.2.3). Por fim, a informação de actualização de todos os vizinhos para os quais foram determinados novos valores de fluxo é enviada por *broadcast*, numa única mensagem de difusão para todos os vizinhos (estando associada a identificação de cada um ao respectivo valor de fluxo) (pontos 2.3). A inversão do sinal do fluxo entre o valor enviado (registado pelo destinatário a menos da perda de mensagem) e o valor registado localmente representa a entrada/saída de fluxo. Um valor de fluxo positivo num dos nós (adição ao valor inicial) tem de ser contrabalançado por um valor negativo no outro nó (subtracção ao valor inicial), de forma a manter a aplicação do princípio da conservação da massa nas estimativas produzidas por cada um.

No evento "**On Receive**" é apresentado o conjunto de acções executadas pelo algoritmo aquando da recepção de uma mensagem enviada por outro nó fruto da execução do mesmo. Ao receber uma mensagem de agregação, é extraída o informação dos fluxo $\mathcal{F}'$, o valor da nova estimativa v' (ponto 1) e a identificação do emissor "i" (ponto 2). Relativamente a informação dos fluxos, cada nó considera apenas a informação que lhe é destinada $\mathcal{F}'_j$ (associada ao seu identificador "j"). Os valores recebidos são registados localmente no respectivo vector de valores e associados ao emissor: o fluxo recebido é registado no vector $\mathcal{F}_i$ (ponto 3); o valor estimado é registado no vector $\mathcal{V}_i$ (ponto 4).

Despoletando o evento "**On Estimate**", em qualquer momento a estimativa do valor agregado pode ser fornecida em qualquer nó pela Formula 3.1, em que v_0 representa o valor agregado inicial e n corresponde ao número de vizinhos do nó (tamanho do vector $\mathcal{F}_i$). Ao longo da execução do algoritmo a estimativa produzida pelos nós é cada vez mais precisa, convergindo em todos eles para o valor agregado exacto.

3.2 Algoritmo sem *broadcast*

Nesta secção é descrita uma variante *unicast* do algoritmo apresentado anteriormente, de forma a permitir uma comparação justa, em igualdade de circunstâncias, do *Flow Updating* com outros algoritmos que não usam o

broadcast. O pseudo código do *Flow Updating* sem *broadcast* é apresentado na tabela 3.2, sendo realçadas as diferenças relativamente ao algoritmo anterior (*On Tick* — pontos 1, 2, 2.4 e *On Receive* — ponto 1 e 3).

Tabela 3.2: Algoritmo — *Flow Updating*

On Init:

1. Set initial values:

$\mathcal{V} \leftarrow 0$

$\mathcal{F} \leftarrow 0$

$v_0 \leftarrow$ InitValue()

On Tick:

1. Get neighbor to update:

$i \leftarrow$ GetNeighbor()

2. Send adjusting, if neighbor found ($i \neq NULL$):

2.1. Calculate new estimated value v':

$v' \leftarrow (\text{Estimate}() + \sum_{i=1}^{n} \mathcal{V}_i)/(n+1)$

2.2. Calculate and store neighbor flow $\mathcal{F}_i$:

$\mathcal{F}_i \leftarrow \mathcal{F}_i - (v' - \mathcal{V}_i)$

2.3. Store new estimated value v':

$\mathcal{V}_i \leftarrow v'$

2.4. Send message $(v', -\mathcal{F}_i)$ to neighbor i.

On Receive:

1. Get message data (v', f).

2. Get message sender Id "i".

3. Store flow value f:

$\mathcal{F}_i \leftarrow f$

4. Store estimated value v':

$\mathcal{V}_i \leftarrow v'$

On Estimate:

1. Return value:

return $v_0 + \sum_{i=1}^{n} \mathcal{F}_i$

A ideia base do algoritmo anteriormente descrito mantêm-se; a principal diferença reside no processamento da actualização de um único vizinho por

cada iteração, sendo enviada apenas uma mensagem apenas para este. Visto que nesta versão do algoritmo apenas um vizinho é escolhido por iteração, torna-se importante optimizar a escolha desse vizinho, de forma a tentar maximizar o desempenho do algoritmo. À semelhança do algoritmo anterior, o processo de verificação das estimativas e escolha é efectuado pela função GetNeighbor() (ponto 1), sendo neste caso escolhido um único vizinho, correspondendo ao que apresenta a maior discrepância de estimativa relativamente a média das estimativas actuais. Este método de escolha de vizinhos revelou-se mais adequado que a simples escolha aleatória de um vizinhos independentemente da sua estimativa, apresentando melhor desempenho que este último. Outras eventuais heurísticas podem ser utilizadas na implementação da função GetNeighbor(), de acordo com determinadas conveniências e circunstâncias de execução do algoritmo, no entanto apenas foram avaliadas e consideradas as já referidas.

Capítulo 4

Simulação

Nesta secção é descrito o modelo de simulação utilizado e é apresentada de forma mais detalhadas a implementação dos algoritmos utilizados para comparação com o *Flow Updating*.

Foi implementado um novo simulador de rede, apesar da existência de uma grande variedade de simuladores de rede, tais como: REAL [39], ns-2 [1], ATEMU [56], Avrora [64, 63], SENSE [11], EmStar [25, 24, 18] e SENS [62]. Diversos factores motivaram o desenvolvimento de um simulador próprio, nomeadamente: garantir a satisfação de todos os requisitos do modelo de simulação definidos para a comparação dos algoritmos; permitir a implementação especifica de determinados automatismos, ao nível da geração de diferentes topologias de rede, execução do processo de simulação e geração de resultados de forma personalizada (integração com outras aplicações gráficas: *gnuplot*); facilitar o encadeamento de todas as fases envolvidas na simulação e obtenção de resultados de acordo com os objectivos pretendidos. O principal propósito do simulador de rede é permitir uma comparação justa e imparcial, em condições idênticas, entre os vários algoritmos estudados, efectuando uma simulação de alto nível num ambiente controlado.

4.1 Simulador de rede

O simulador de rede implementado é um elemento auxiliar importante que possibilitou a obtenção de resultados concretos, no entanto ele não constitui o principal objecto deste estudo, como tal, é efectuada uma exposição resumida do mesmo.

O simulador de rede foi implementado em Java, sendo constituindo por três módulos principais: gerador de redes, motor de simulação e gerador de resultados.

O gerador de redes permite basicamente a criação de redes com diferentes topologias, estando neste momento implementada a geração de redes: *2D/Mesh* — rede baseada na proximidade geográfica, os nós são espalhados aleatoriamente formando uma "malha", sendo cada nó apenas capaz de comunicar directamente com aqueles que se encontram dentro de uma determinada distância (e.g. redes de sensores); *Random* — rede totalmente aleatória, tanto ao nível da dispersão dos nós como ao nível do estabelecimento de ligações directas entre os nós; *Attach/Small-world* — rede representativa de cenários reais, onde tipicamente alguns nós têm tendência para concentrar um maior número de ligações (e.g. Internet, onde alguns dispositivos concentradores têm um maior número de ligações). O módulo está implementado de forma a facilitar o desenvolvimento de novas topologias de redes (criando apenas uma nova classe que implemente uma interface especifica). Foi ainda considerada a possibilidade de importar redes criadas por outros geradores de redes conhecidos, tais com: BRITE [50, 51, 52], GT-ITM [10, 68], Inet [67], Barabasi Graph Generator [17], Bessie [2], Pajek [6] e JUNG [55].

O motor de simulação é responsável pela execução do processo de simulação propriamente dito, respeitando o modelo de simulação definido (ver Secção 4.2). A implementação deste módulo foi pensada de forma a possibilitar a expansão das funcionalidades actuais e facilitar o desenvolvimento de outros modelos de simulação (e.g. criação de um modelo de simulação que suporte o dinamismo e mobilidade dos elementos da rede). O motor de simulação controla a execução dos algoritmos simulados e gere todas as trocas de mensagens efectuadas entre os nós da rede, de acordo com os parâmetros

e configurações definidos.

O gerador de resultados trata da recolha de estatísticas e resultados das simulações executadas, permitindo a geração de diversos gráficos e registos de execução (tabela com registos estatísticos de um nó ao longo da simulação, nas diversas iterações). Para além da geração de vários gráficos tendo em conta os formatos de outras aplicações externas — *gnuplot*, este módulo permite também a invocação directa dessas aplicações, através da configuração dos comandos apropriados, possibilitando a visualização automática dos resultados imediatamente após o término da simulação.

Todo os parâmetros de simulação estão definido num único ficheiro de configuração. O ficheiro de configuração determina o tipo e características da topologia de rede gerada, quais os algoritmos simulados e os seus respectivos parâmetros de execução, a quantidade de repetições efectuadas, os critérios de paragem das simulações, as restrições e dependências no envio e recepção de mensagens, a percentagem de perdas de mensagens e todas as definições dos diferentes gráficos gerados (tipos de gráficos, algoritmos representados, legendas, títulos, escalas e valores exibidos — iterações, quantidade de mensagens enviadas, quantidade de mensagens recebidas, número total de mensagens, quantidade de mensagens descartadas, quantidade de mensagens perdidas e *Root Mean Square Error*).

4.2 Modelo de simulação

Modelou-se um sistema de comunicação de alto nível, em que um conjunto de nós (rede) comunicam entre si, trocando mensagens, através de ligações simétricas (canal de comunicação bidireccional). De acordo com uma topologia bem definida (*2D/Mesh*, *Random* ou *Attach/Small-world*), é gerada uma rede estática com um número predefinido de nós, estabelecendo todos eles pelo menos uma ligação com outro nó. Assume-se que cada nó conhece apenas os nós com os quais está directamente ligado — vizinhos. Considera-se a ocorrência da troca assíncrona de mensagens, assumindo uma latência variável em cada iteração da simulação, para cada ligação estabelecida (na medida em que duas mensagens enviadas em simultâneo a partir de canais

de comunicação diferentes podem chegar ao mesmo nó numa ordem diferente em diferentes iterações) — ordem aleatória de envio/chegada de todas as mensagens. A troca de mensagens entre nós pode ser efectuada por *unicast* (envio de uma mensagem para um único destinatário) ou *broadcast* (envio da uma mensagens para vários destinatários, a mesma mensagem é recebida por todos os vizinhos). A execução do processo de simulação está dividida em iterações. Todos os nós podem comunicar, na mesma iteração, através de todas a ligações disponíveis (directamente estabelecidas com os seus vizinhos). As mensagens enviadas numa iteração, só são recebidas na iteração seguinte — a iteração é representativa do tempo mínimo de trânsito de uma mensagem. Em cada iteração, os nós poderão processar todas as mensagens recebidas (ou um número máximo de mensagens predefinido, podendo as mensagens excedentes ser descartadas ou processadas na próxima iteração). Para além da divisão em iterações, considera-se também a divisão da simulação em diversas rondas constituídas por um número fixo de iterações previamente configurado para cada algoritmo. Cada nó fica no estado activo, despoletando o evento "On tick", apenas uma vez por ronda (numa iteração aleatória dessa ronda). Para permitir o estudo do efeito da ocorrência de faltas, considera-se a possibilidade de ocorrência de perdas de mensagens em todas as comunicações, tendo em conta um valor de probabilidade predefinido. No caso de se tratar de comunicação por *broadcast* é considerado que a ocorrência de faltas (perdas) afecta directamente o nó emissor, fazendo com que nenhum dos destinatários receba a mensagem, assumindo-se um cenário pessimista e intransigente.

4.3 Algoritmos comparados

O *Flow Updating* foi comparado em ambiente de simulação com três algoritmos representativos da mesma classe de protocolos de agregação — *Averaging* (ver Secção 2.2.2): *Push-Sum Protocol* [38], *Push-Pull Gossiping* [31] e *DRG* [12]. Para comparar os algoritmos, foi implementada uma versão dos mesmos para fornecer a estimativa do tamanho da rede. A implementação dos algoritmos foi efectuada, atendendo as especificidades do simulador de

rede — programação *event-driven*. Desta feita, a estrutura dos algoritmos comparados é semelhante àquela que já foi apresentada anteriormente na descrição do *Flow Updating* (ver Capítulo 3), estando dividida em quatro componentes correspondendo cada uma a um evento específico: "On Init" — evento de inicialização do algoritmo, "On Tick" — evento de activação periódico (uma vez por ronda), "On Receive" — evento associado a recepção de mensagens e, "On Estimate" — evento associado a solicitação do valor estimado. O pseudo código da implementação no simulador dos algoritmos confrontados com o *Flow Updating* é exposto nas tabelas 4.1, 4.2 e 4.3.

Tabela 4.1: Algoritmo — Push-Sum Protocol

On Init:

1. Set initial values:

$w \leftarrow$ InitWeight()

$v \leftarrow$ InitValue()

On Tick:

1. Get neighbor uniformly at random:

$nodeId \leftarrow$ GetRandomNeighbor()

2. Calculate values to send (v' and w')

$v' \leftarrow v/2$

$w' \leftarrow w/2$

3. Set message data (v', w').

4. Send message to target neighbor "$nodeId$" and self.

On Receive:

1. Get message data (v' and w').

2. Update values v and w:

$v \leftarrow v+v'$

$w \leftarrow w+w'$

On Estimate:

1. Return value:

return w/v

O *Push-Sum Protocol* [38] baseia-se numa disseminação epidémica de duas variáveis v (a soma dos valores agregados) e w (o peso associado), como é referido na Secção 2.2. Inicialmente ("On Init") as variáveis v e w são inicializadas pelas funções InitValue() e InitWeight() respectivamente. Para o caso específico da estimativa do tamanho da rede, o valor do peso w é inicializado em todos os nós com o valor 1, o valor da soma agregada v é inicializado com o valor 1 apenas num nó e 0 nos restantes. Periodicamente ("On Tick") cada nó escolhe um vizinho aleatoriamente (ponto 1) e envia metade dos valores v' e w' (ponto 2) para o nó escolhido e para ele próprio (pontos 3 e 4). Ao receber uma mensagem ("On Receive"), os nós extraem os valores recebidos v' e w' (ponto 1) e somam-nos aos valores locais correspondentes (v e w), actualizando os valores locais (ponto 2). A estimativa do tamanho da rede é fornecida em todos os nós pelo resultado da divisão w/v ("On Estimate").

O *Push-Pull Gossiping* [31] assenta essencialmente no mesmo princípio do *Push-Sum Protocol*, propagando os valores agregados de forma epidémica, mas efectuando trocas síncronas entre pares de nós, de acordo com o mencionado na Secção 2.2. Este algoritmo fomenta o intercâmbio bidireccional dos dados agregados entre nós, através da troca de dois tipos de mensagens "PUSH" e "PULL". O envio de uma mensagem de agregação "PUSH" para um determinado nó, pressupõe sempre a recepção de uma mensagem de agregação "PULL" vinda do nó escolhido. No início ("On Init"), o valor agregado v é inicializado pela função InitValue(), sendo nesta situação atribuído o valor 1 a um único nó e o valor 0 aos restantes. Em cada ronda ("On Tick"), cada nó escolhe aleatoriamente um vizinho (ponto 1) e envia-lhe uma mensagem "PUSH" com o seu valor agregado v (ponto 2). Na recepção de mensagens ("On Receive"), os nós executam acções diferentes de acordo com o tipo de mensagem recebida. Caso seja recebida uma mensagem do tipo "PUSH" (ponto 2.1), o nó envia de volta para o emissor uma mensagem do tipo "PULL" com o seu valor agregado v e actualiza posteriormente o seu valor agregado v tendo em conta o valor v' recebido. Se receber uma mensagem do tipo "PULL" (ponto 2.2), o nó actualiza simplesmente o seu valor agregado v atendendo ao valor recebido v'. Em qualquer momento, a esti-

mativa do tamanho da rede ("On Estimate") é dada em todos os nós pela fracção $1/v$.

Tabela 4.2: Algoritmo — Push-Pull Gossiping

On Init:

1. Set initial values:

$v \leftarrow$ InitValue()

On Tick:

1. Get neighbor uniformly at random:

$nodeId \leftarrow$ GetRandomNeighbor()

2. Set message data (v).

3. Send "PUSH" message to target neighbor "$nodeId$".

On Receive:

1. Get message data (v').

2. Check message type $msgType$:

2.1. If $msgType =$"$PUSH$":

2.1.1. Set message data (v).

2.1.2. Send "PULL" to sender.

2.1.3. Update value v.

$v \leftarrow (v+v')/2$

2.2. If $msgType =$"$PULL$":

2.2.1. Update value v

$v \leftarrow (v+v')/2$

On Estimate:

1. Return value:

return $1/v$

O *DRG* [12] baseia-se num conceito diferente dos dois algoritmos anteriores, tentando tirar proveito da natureza das comunicação em redes sem fios — *broadcast*. O algoritmo baseia-se na criação aleatória de grupos temporários, liderados por um nó central que recolhe os valores dos elementos pertencentes ao grupo e calcula o valor agregado do grupo, distribuindo pos-

teriormente o resultado para o grupo, conforme o descrito na Secção 2.2. O processo de agregação é efectuado por etapas, correspondendo cada uma a um estado de execução diferente em cada nó: IDLE, LEADER e MEMBER. Inicialmente ("On Init"), todos os nós estão no estado "IDLE", sendo a função INITVALUE() responsável pela atribuição do valor agregado inicial v (para o caso da estimativa do tamanho da rede é atribuído o valor 1 a um único nó e o valor 0 aos restantes). Para além da variável de agregação v são inicializadas outras variáveis auxiliares, necessárias para a determinação do valor agregado de cada grupo, tais como: a identificação do grupo $grpId$, a soma dos valores agregados do grupo $grpAvg$ e o tamanho do grupo $grpSize$. Em cada ronda ("On Tick"), são executados conjuntos de acções diferentes de acordo com o estado actual de cada nó — "IDLE" ou "LEADER". Assim sendo, periodicamente, todos os nós no estado "IDLE" podem decidir tornarem-se "LEADER" de acordo com uma probabilidade pg predefinida, executando a função BECOMELEADER(PG) (ponto 1.2.1). Se um nó resolve torna-se líder de um grupo (ponto 1.2.1.1), ele gera um identificador único para o seu grupo $grpId$, executando a função GENERATEID() e, envia uma mensagem de criação do grupo "GCM" (*Group Call Message*) com o identificador gerado, por *broadcast* para todos os seus vizinhos. Caso um nó se encontre no estado "LEADER" no inicio da ronda, ele aguarda pela recepção de mensagem de confirmação de junção ao grupo "JACK" (*Joining Acknowledgment*) dos seus membros, até atingir um tempo máximo predefinido $timeoutJACK$ (ponto 1.1.1). Quando o tempo máximo de espera é atingido, o líder calcula o novo valor agregado do grupo v (ponto 1.1.1.1.1) e envia-o por *broadcast* numa mensagem "GAM" (*Group Assignement Message*) para todos os seus membros (pontos 1.1.1.1.2 e 1.1.1.1.3). Depois de enviar a mensagem, o líder volta para o seu estado inicial — "IDLE", terminando a existência do grupo e reiniciando as variáveis auxiliares utilizadas (ponto 1.1.1.1.4). Na recepção de mensagens ("On Receive"), os nós executam acções diferentes de acordo com o seu estado de execução e o tipo de mensagem recebida. Caso o nó esteja no estado "IDLE" (ponto 1.1), ele apenas considera a primeira mensagens "GCM" recebida, juntando-se ao grupo à qual diz respeito (pontos 1.1.1.1.1 e 1.1.1.1.2), respondendo ao emissor com o envio de uma mensagem "JACK"

com o seu valor agregado local v (pontos 1.1.1.1.3 e 1.1.1.1.4) e alterando o seu estado para "MEMBER" (ponto 1.1.1.1.5). No caso do nó estar no estado "MEMBER" (ponto 1.2), ele aguarda pela recepção de uma mensagem do tipo "GAM" do grupo ao qual pertence (ponto 1.2.2.1) com o resultado do valor agregado do grupo v' (ponto 1.2.2.1.1). Após a recepção dessa mensagem, o nó actualiza a sua estimativa local v com o valor recebido, termina o seu vínculo ao grupo e volta para o estado "IDLE", reiniciando as variáveis auxiliares utilizadas (ponto 1.2.2.1.2). Se o nó se encontra no estado "LEADER" (ponto 1.3), ele aguarda pela recepção das mensagens "JACK" de todos os vizinhos que se juntaram ao grupo (ponto 1.3.1.1) com seus valores agregados locais v' (ponto 1.3.1.1.1) e, calcula os valores parciais da soma dos valores agregados $grpAvg$ e tamanho do grupo $grpSize$ (ponto 1.3.1.1.2). A estimativa do tamanho da rede ("On Estimate") é fornecida em todos os nós da rede pelo resultado da operação $1/v$.

Tabela 4.3: Algoritmo — DRG

On Init:

1. Set initial values:

$mode \leftarrow$"$IDLE$"

$v \leftarrow$InitValue()

$grpId \leftarrow NULL$

$grpAvg \leftarrow v$

$grpSize \leftarrow 1$

On Tick:

1. Check node *mode*:

1.1. If $mode =$"$LEADER$":

1.1.1. Check if "JACK" message *timeout* is reach:

1.1.1.1. If $\neg(timeout < timeoutJACK)$:

1.1.1.1.1. Calculate new value v:

$v \leftarrow grpAvg/grpSize$

1.1.1.1.2. Set message data $(grpId, v)$.

(continua na próxima página)

Tabela 4.3: Algoritmo — DRG *(continuação)*

1.1.1.1.3. Broadcast "GAM" message to all neighbors.

1.1.1.1.4. Set values $mode$, $grpId$, $grpSize$ and $grpAvg$:

$mode \leftarrow$"$IDLE$"

$grpId \leftarrow NULL$

$grpAvg \leftarrow v$

$grpSize \leftarrow 1$

1.2. If $mode =$"$IDLE$":

1.2.1. Decide to become leader with probability pg:

$mode \leftarrow$BecomeLeader(pg)

1.2.1.1. If $mode =$"$LEADER$":

1.2.1.1.1. Set value $grpId$:

$grpId \leftarrow$GenerateId()

1.2.1.1.2. Set message data ($grpId$).

1.2.1.1.3. Broadcast "GCM" message to all neighbors.

On Receive:

1. Check node $mode$:

1.1. If $mode =$"$IDLE$":

1.1.1. Check message type $msgType$:

1.1.1.1. If $msgType =$"GCM":

1.1.1.1.1. Get message data ($grpId'$).

1.1.1.1.2. Set value $grpId$:

$grpId \leftarrow grpId'$

1.1.1.1.3. Set message data ($grpId$, v).

1.1.1.1.4. Send JACK message to leader.

1.1.1.1.5. Set value $mode$:

$mode \leftarrow$"$MEMBER$"

1.2. If $mode =$"$MEMBER$":

1.2.1. Get message data ($grpId'$).

1.2.2. Check message type "$msgType$" and group Id $grpId'$:

1.2.2.1. If $msgType =$"GAM"$\wedge grpId = grpId'$

(continua na próxima página)

Tabela 4.3: Algoritmo — DRG *(continuação)*

1.2.2.1.1. Get message data (v').

1.2.2.1.2. Set values v, $grpAvg$, $grpId$ and $mode$:

$v \leftarrow v'$

$grpAvg \leftarrow v$

$grpId \leftarrow NULL$

$mode \leftarrow$"$IDLE$"

1.3. If $mode =$ "$LEADER$'':

1.3.1. Check message type "$msgType$" and group Id $grpId'$:

1.3.1.1. If $msgType =$"$JACK$"$\wedge grpId = grpId' \wedge timeout < timeoutJACK$

1.3.1.1.1. Get message data (v').

1.3.1.1.2. Calculate values $grpAvg$ and $grpSize$:

$grpAvg \leftarrow grpAvg + v'$

$grpSize \leftarrow grpSize + 1$

On Estimate:

1. Return value:

return $1/v$

Capítulo 5

Resultados & Discussão

Nesta secção são apresentados e discutidos os resultados da comparação entre o *Flow Updating* (nas suas duas variantes: *unicast* e *broadcast*) e alguns algoritmos concorrentes: *Push-Sum Protocol* [38], *Push-Pull Gossiping* [31] e *DRG* [12]. Os algoritmos comparados pertencem à mesma classe, de acordo com a classificação taxonómica apresentada na Secção 2.2.2 — *Averaging\Gossip*, sendo estes os mais representativos da sua categoria.

Todos os algoritmos foram avaliados num ambiente de simulação implementado especificamente para o efeito (ver Capítulo 4), garantido as mesmas condições de execução, de forma a proporcionar uma comparação relativa justa e imparcial. Foram reproduzidos diversos cenários de simulação, considerando a aplicação dos algoritmos em redes com diferentes características. Assim sendo, os algoritmos foram comparados em redes com diferentes topologias (*2D/Mesh*, *Random* e *Attach*), diferentes tamanhos (quantidade de nós que constituem a rede) e diferentes graus de conectividade (quantidade de ligações médias entre nós). A realização das simulações em diferentes estruturas de rede, considerando uma razoável variação das condições de simulação, é importante na avaliação dos diversos algoritmos, permitindo aferir o relacionamento do desempenho dos algoritmos com as características das redes sobre as quais eles são executados.

Para medir o desempenho dos algoritmos avaliados, foram considerados essencialmente três critérios: precisão da estimativa, tempo de execução e

número de mensagens trocadas (comunicação). Cada uma destas métricas pretende fornecer uma avaliação dos algoritmos sobre prismas diferentes, podendo servir de orientação para decidir sobre a sua utilização em situações distintas. De acordo com o contexto de utilização, poderá ser mais vantajoso considerar o uso de um algoritmo de agregação que possua melhor comportamento e desempenho relativamente a um dos critérios, em detrimento dos restantes. A utilização dos referidos critérios de avaliação permite ajuizar diferentes aspectos, nomeadamente:

Precisão — A precisão é um dos critérios mais importantes na análise deste tipo de algoritmos, permitindo assumir um determinado nível de confiança no valor da estimativa produzida. Alguns algoritmos convergem ao longo da execução, podendo produzir resultados exactos em todos os nós. Outros algoritmos, independentemente do tempo de execução ou do número de mensagens trocadas, apresentam sempre resultados aproximados com uma margem de erro conhecida (não convergindo para um resultado exacto). Em algumas situações pode ser favorável abdicar da precisão da estimativa produzida, em favor de uma maior velocidade de execução ou da troca de um menor número de mensagens. Neste casos pode-se recorrer a algoritmos que produzam uma estimativa menos precisa, mas que tenham um melhor desempenho ao nível do tempo de execução e/ou troca de mensagens (como é referido em [49]). É de notar que mesmo nessas situações, pode ser considerada a possibilidade de usar algoritmos mais "precisos", que convergem para um valor exacto, antecipando o final da execução do processo, após atingir um critério de paragem predefinido. Noutras situações é fundamental a obtenção de uma estimativa precisa e fiável. Neste caso, apenas se pode considerar a utilização de algoritmos com capacidade de produzir estimativas precisas, ao contrário da situação anterior (de acordo com a natureza do algoritmo usado, um aumento do tempo de execução e/ou quantidade de mensagens trocadas pode não ser suficiente para aumentar a precisão da estimativa). Como medida da precisão da estimativa obtida nas simulações efectuadas, foi utilizado o *Root Mean*

Square Error (RMSE) representando a taxa de erro média das estimativas produzidas em todos os nós da rede face ao valor a estimar.

Tempo — A medida do tempo de execução do algoritmo até produzir a estimativa do valor agregado (com um grau de precisão predefinido), faculta uma noção temporal da latência do algoritmo e da velocidade de convergência do mesmo. Embora neste estudo apenas seja efectuada uma avaliação dos algoritmos em redes estáticas, supondo a execução dos algoritmos num ambiente dinâmico, com entrada e saída de nós, a capacidade de resposta ao dinamismo da rede poderá estar intrinsecamente relacionada com esta medida. Nessa situação, se consideraremos que sempre que ocorra uma alteração na rede (detectada por um processo externo), o algoritmo tenha de ser reiniciado por completo para determinar a nova estimativa, então o tempo de resposta a essa modificação poderá ser no mínimo aproximado à soma do tempo de execução do algoritmo com o tempo de detecção da alteração da rede. Por isso, talvez esta medida seja aquela que melhor caracterize a capacidade de adaptação do algoritmo às alterações do tamanho da rede, embora esta insinuação careça de uma prova teórica ou empírica a ser desenvolvida num trabalho futuro. Como medida do tempo de execução é utilizado o número de iterações. De acordo com os pressupostos estabelecidos anteriormente (ver Secção 4.2), uma iteração corresponde aproximadamente ao tempo de envio de uma mensagem, supondo que o tempo de envio de uma mensagem t_e é igual em toda a rede para todas as ligações directas (entre nós vizinhos), independentemente da distância, sendo desprezado o tempo de computação t_c dos nós, assumindo $t_e \gg t_c$.

Comunicação — A quantidade de mensagem trocadas pode ser vista como uma medida do consumo energético (importante principalmente nas redes de sensores) e da escalabilidade dos algoritmos. Em termos de consumos energéticos numa rede de sensores, existe uma grande diferença entre os gastos de comunicação e de computação. O consumo energético na transmissão de dados é bastante maior do que o do pro-

cessamento de dados, podendo o envio de um simples bit corresponder ao processamento de 125 instruções [27][1]. Tendo em conta esta noção, torna-se vantajoso favorecer o processamento de dados, em detrimento do envio de dados. Esse é um dos motivos que favorece o recurso ao processamento distribuído em vez do processamento centralizado neste tipo de redes. Com isto, é evidentemente fundamental minimizar o número de trocas de mensagens em redes de sensores (visto ser o principal consumidor energético), a fim de aumentar a autonomia da mesma. Do ponto de vista da escalabilidade, a necessidade de efectuar uma grande quantidade de trocas de mensagens para o funcionamento de um determinado sistema pode constituir um entrave à escalabilidade do mesmo [53]. Assim sendo, é desejável equilibrar as trocas de mensagens necessárias para atingir uma determinada estimativa (evitando sobrecargas e centralização das mesmas), a fim de maximizar o nível de escalabilidade. Para medir o nível de comunicação dos algoritmos é utilizado a quantidade de mensagens enviadas.

Nas próximas secções são discutidos os resultados de algumas simulações efectuadas. Primeiro, é apresentado uma comparação global de todos os algoritmos em estudos de acordo com as variações da topologia, tamanho e grau de conectividade da rede. De seguida, é efectuada uma avaliação especifica de alguns algoritmos, expondo alguns problemas e fragilidades, sendo análise alguns aspectos específicos, tais como, a tolerância a faltas e a utilização de "Ilhas".

5.1 Avaliação global

Na avaliação global dos algoritmos testados, foram reproduzidos 18 cenários de simulação distintos, sendo todos os algoritmos executados até alcançarem

[1]Existe também um consumo energético elevado aquando a recepção de mensagens, embora seja de ordem inferior ao gasto no envio (cerca de metade). Por uma questão de simplicidade, este facto não é considerado. Para isso, teriam de ser consideradas estratégias de mais baixo nível, para ligar\desligar os componentes RF (Rádio Frequência), podendo esta optimização ficar a cargo da gestão do sistema operativo.

um critério de paragem predefinido (número máximo de iteração ou valor mínimo de RMSE). Cada um dos cenários corresponde a uma rede com características especificas ao nível da topologia, tamanho e grau de conectividade. Neste contexto, foram consideradas três topologias de rede (*Random*, *Attach* e *2D*), três tamanhos (100, 1000 e 10000) e dois diferentes graus de conectividade ($\log n$ e 10). Cada cenário de simulação é representado por uma das diversas combinações das referidas características de rede, possibilitando a análise do desempenho dos algoritmos de acordo com a variação dessas mesmas características.

A utilização de diferentes topologias permite analisar a influência das estruturas de comunicação no desempenho dos algoritmos, sendo consideradas as redes: *Random* — estrutura uniforme aleatória; *Attach* — estrutura com uma maior concentração de número de ligações em alguns nós; *2D* — estrutura baseada na proximidade geográfica. Os vários tamanhos de rede usados (100, 1000 e 10000) permitem verificar o comportamento dos algoritmos ao nível da escalabilidade. Em termos de conectividade, foram considerados apenas dois valores, com o objectivo de permitir uma análise do desempenho dos algoritmos de acordo com a variação do grau de conectividade da rede. O primeiro valor utilizado $\log n$ é encarado como um valor mínimo, correspondendo ao grau de conectividade que alguns nós devem possuir, de forma a garantir a conectividade da rede, considerando um probabilidade de falha de $1/2$ em todos os nós da rede [36]. O segundo valor utilizado 10 corresponde a um grau de conectividade médio constante e relativamente mais elevado que o primeiro (mais do dobro).

Todos os parâmetros de simulação dos algoritmos foram ajustados, de forma a proporcionarem o melhor desempenho para os cenários considerados. Cada simulação foi repetidas 10 vezes, sendo apresentado o resultado médio da execução das várias repetições. Em cada repetição foi utilizada uma rede diferente (gerada pelo simulador), mas com os mesmo parâmetros de criação (tipo de topologia, tamanho e grau de conectividade), sendo as mesma redes utilizadas nas simulações de todos os algoritmos, garantindo as mesmas condições de simulação em todas as repetições. Durante o processo de simulação, os algoritmos foram executados continuamente até atingirem

um valor de RMSE mínimo predefinido (0,01) ou um número máximo de iteração (variável de acordo com o cenário de simulação considerado). Para ilustrar os resultados das simulações, ao longo desta secção são apresentados diversos gráficos e tabelas com os resultados obtidos. Os gráficos apresentam a variação da precisão das estimativas dos algoritmos (RMSE) em relação ao tempo (iterações) e custo de comunicação (número de mensagens enviadas), para todos os cenários avaliados. As tabelas mostram os valores finais resultantes de cada simulação, isto é, RMSE ($RMSE(\hat{n})$), número de iterações ($COUNT(iter)$), número de mensagens enviadas ($COUNT(msg)$), valor médio estimado por todos os nós ($MEAN(v)$), valor mínimo estimado ($MIN(v)$) e valor máximo estimado ($MAX(v)$).

Observando todos os gráficos expostos neste capítulo, à primeira vista podemos observar que em todos eles o *Push-Pull Gossiping* apresenta um comportamento curioso, estabilizando num valor de RMSE mais elevado e atingindo uma menor precisão que os restantes. Este facto é posteriormente discutido e analisado na Secção 5.2.

5.1.1 Redes *Random*

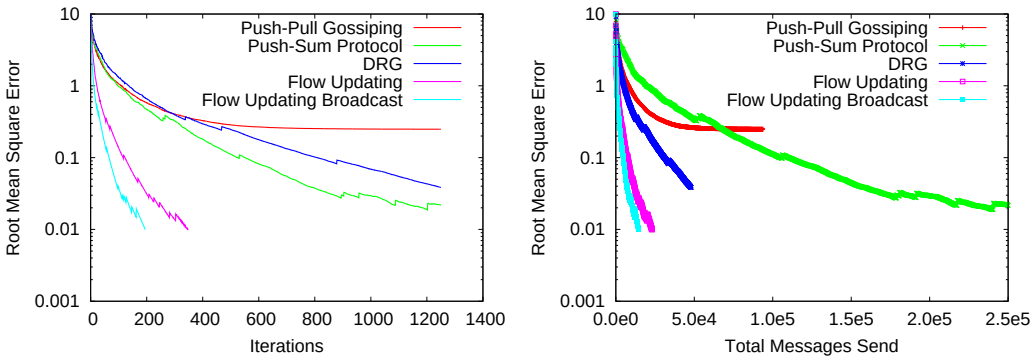
Analisando os resultados obtidos para a topologia *Random* (Tabela 5.1) podemos verificar que o algoritmo com melhor desempenho é o *Flow Updating Broadcast*, tendo melhor tempo de execução em todos os cenários e gastando um menor número de mensagens nas redes com um grau de conectividade baixo. No entanto, nos cenários de rede *Random* com grau de conectividade mais elevado, o *DRG* é aquele que gasta o menor número de mensagem para produzir a estimativa do tamanho da rede com a precisão pretendida. Verifica-se uma diferença no desempenho dos algoritmos neste tipo de rede de acordo com o grau de conectividade da mesma.

Com um grau de conectividade $\log n$, os algoritmos *Flow Updating Broadcast* e *Flow Updating* destacam-se dos restantes em termos de número de iterações e quantidade de mensagens enviadas necessárias para atingir a estimativa final, sendo os mais "rápidos" e "económicos" ao nível da comunicação como se pode verificar nos Gráficos 5.1, 5.3 e 5.5.

Tabela 5.1: Resultados globais em redes *Random*.

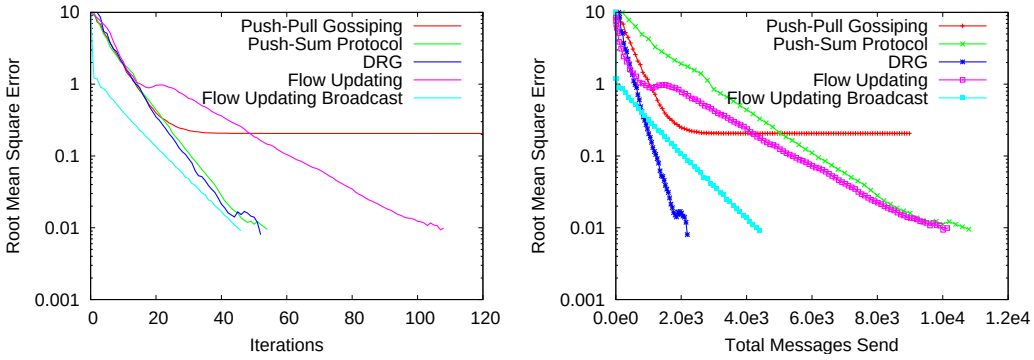
		Push-Pull Gossiping	Push-Sum Gossip	DRG	Flow Updating	Flow Updating Broadcast
	$RMSE(\hat{n})$	0,25	0,01	0,03	0,01	0,01
	$COUNT(iter)$	1250	859	1043	201	105
$(n = 100)$	$COUNT(msg)$	93678	171920	40012	13076	8003
$(d = \log n)$	$MEAN(v)$	84,04	100,05	100,00	100,00	100,00
	$MIN(v)$	83,40	98,38	94,91	98,11	97,89
	$MAX(v)$	84,93	103,46	107,34	102,42	103,03
	$RMSE(\hat{n})$	0,21	0,01	0,01	0,01	0,01
	$COUNT(iter)$	120	47	44	100	44
$(n = 100)$	$COUNT(msg)$	8979	9580	1906	9426	4330
$(d = 10)$	$MEAN(v)$	102,24	100,01	100,01	99,79	100,02
	$MIN(v)$	102,24	97,10	97,20	96,36	97,90
	$MAX(v)$	102,24	102,16	102,17	101,19	102,37
	$RMSE(\hat{n})$	0,24	0,01	0,01	0,01	0,01
	$COUNT(iter)$	550	301	465	70	31
$(n = 1000)$	$COUNT(msg)$	412584	604600	181490	47461	22776
$(d = \log n)$	$MEAN(v)$	1041,69	1000,43	999,99	1000,01	999,90
	$MIN(v)$	1038,69	967,95	947,61	955,00	971,06
	$MAX(v)$	1050,28	1113,12	1108,54	1090,90	1124,44
	$RMSE(\hat{n})$	0,20	0,01	0,01	0,01	0,01
	$COUNT(iter)$	120	62	54	102	56
$(n = 1000)$	$COUNT(msg)$	89915	125000	23262	92857	53701
$(d = 10)$	$MEAN(v)$	981,06	1000,02	999,99	998,03	1000,00
	$MIN(v)$	981,04	931,86	936,74	934,38	968,13
	$MAX(v)$	981,08	1032,90	1074,08	1030,61	1030,93
	$RMSE(\hat{n})$	0,14	0,01	0,02	0,01	0,01
	$COUNT(iter)$	300	198	278	66	31
$(n = 10000)$	$COUNT(msg)$	2247859	3982000	1104904	459435	240746
$(d = \log n)$	$MEAN(v)$	10330,07	10001,23	10000,02	9999,80	10000,05
	$MIN(v)$	9265,05	7890,49	5059,05	8172,75	9196,94
	$MAX(v)$	10861,60	12199,79	12890,99	11796,06	10574,04
	$RMSE(\hat{n})$	0,18	0,01	0,01	0,01	0,01
	$COUNT(iter)$	120	76	69	103	67
$(n = 10000)$	$COUNT(msg)$	900061	1532000	294433	906072	635701
$(d = 10)$	$MEAN(v)$	9226,76	10000,48	9999,98	9981,12	10000,05
	$MIN(v)$	9225,33	8608,86	7866,50	8953,59	9650,76
	$MAX(v)$	9236,32	14052,98	13784,53	10607,46	10385,07

Nos cenários com um grau de conectividade mais elevado (valor 10), o *Flow Updating Broadcast*, *DRG* e *Push-Sum Protocol* apresentam desempenhos muito próximos em termos temporais, embora o *Flow Updating Broadcast* seja ligeiramente melhor (ver Gráficos 5.2(a), 5.4(a) e 5.6(a)). Do ponto de vista da comunicação, o *DRG* é aquele que apresenta melhores resultados, seguido do *Flow Updating Broadcast* e do *Flow Updating*, como se pode observar nos Gráficos 5.2(b), 5.4(b) e 5.6(b). O *Push-Sum Protocol* é o que necessita enviar um maior número de mensagens para produzir uma estimativa com a precisão desejada.



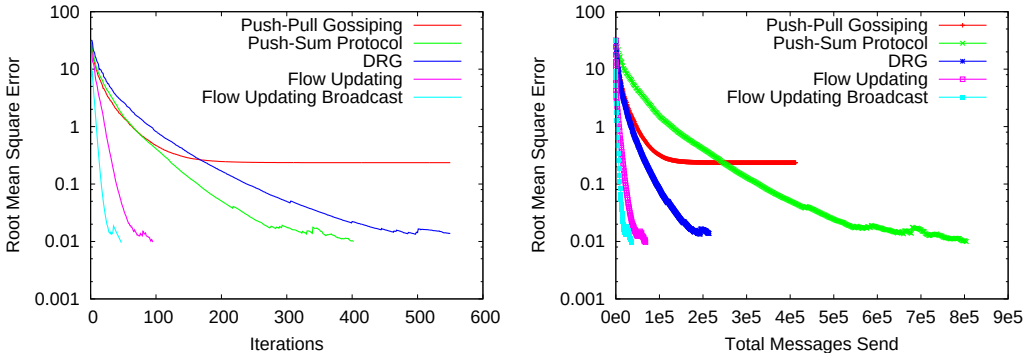
(a) RMSE Vs Iterações (b) RMSE Vs Mensagens Enviadas

Figura 5.1: Resultados numa rede *Random* de tamanho 100 e grau 2 ($\log n$).



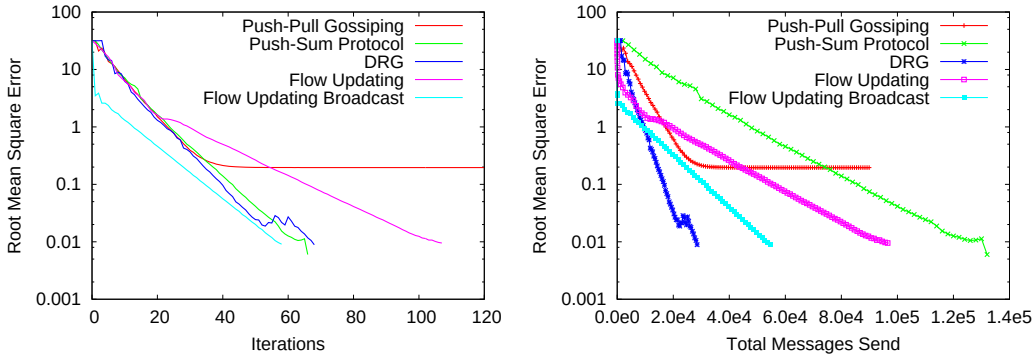
(a) RMSE Vs Iterações (b) RMSE Vs Mensagens Enviadas

Figura 5.2: Resultados numa rede *Random* de tamanho 100 e grau 10.



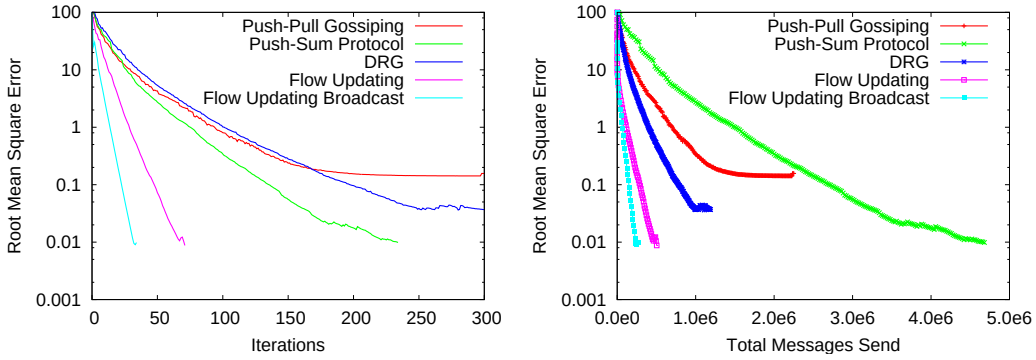
(a) RMSE Vs Iterações (b) RMSE Vs Mensagens Enviadas

Figura 5.3: Resultados numa rede *Random* de tamanho 1000 e grau 3 ($\log n$).



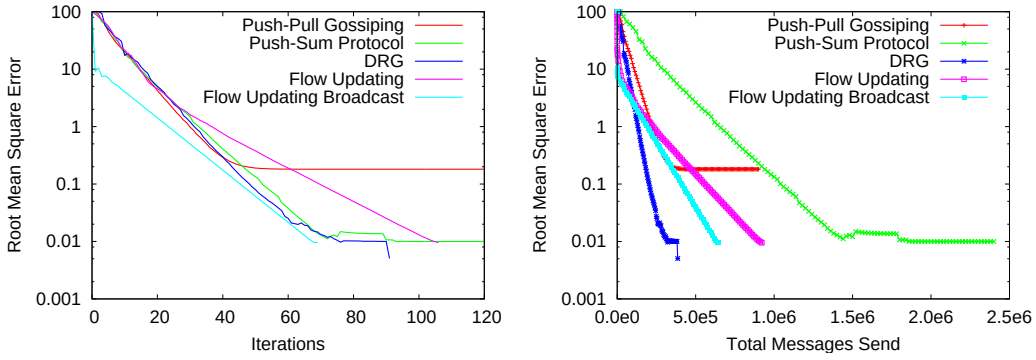
(a) RMSE Vs Iterações (b) RMSE Vs Mensagens Enviadas

Figura 5.4: Resultados numa rede *Random* de tamanho 1000 e grau 10.



(a) RMSE Vs Iterações (b) RMSE Vs Mensagens Enviadas

Figura 5.5: Resultados numa rede *Random* de tamanho 10000 e grau 4 ($\log n$).



(a) RMSE Vs Iterações (b) RMSE Vs Mensagens Enviadas

Figura 5.6: Resultados numa rede *Random* de tamanho 10000 e grau 10.

Os resultados demonstram que o *Flow Updating* não beneficia do aumento do grau de conectividade médio da rede. Por exemplo, em redes de maiores dimensões (1000 e 10000 nós) com grau de conectividade maior, o algoritmo apresenta um pior desempenho comparativamente as mesma redes com grau de conectividade inferior. Esta influência negativa também é observada no *Flow Updating Broadcast*, apesar do seu bom desempenho. Contrariamente, os restantes algoritmos apresentam um melhoria do seu desempenho com o aumento do grau de conectividade médio da rede.

Neste tipo de redes, todos os algoritmos apresentam o mesmo comportamento relativamente ao aumento do tamanho da rede. Tendo em conta um aumento no tamanho da rede de 10 vezes, as diferença da quantidade de iterações para atingir o valor final varia numa ordem bastante inferior ao da quantidade de nós existentes, no entanto o número de mensagens enviadas varia na mesma ordem. Assim sendo, os algoritmos demonstram boas características de escalabilidade apenas em relação ao tempo de execução, pois os custos de comunicação variam linearmente com o aumento do tamanho da rede.

5.1.2 Redes *Attach*

Os resultados obtidos em redes *Attach* são em termos genéricos semelhantes aos obtidos nas redes *Random*. Como se pode verificar pela análise dos resultados obtidos (Tabela 5.2), o *Flow Updating Broadcast* é o algoritmo com melhor desempenho a todos os níveis, no entanto, a concentração de ligações em alguns nós (característica deste tipo de redes) influencia o desempenho de alguns algoritmos. Também neste tipo de topologia, o aumento do grau de conectividade influencia negativamente o *Flow Updating* (com e sem *broadcast*).

Nos cenários com grau de conectividade $\log n$, o *Flow Updating Broadcast* apresenta o melhor desempenho relativamente ao tempo de execução e custos de comunicação, destacando-se da concorrência principalmente em redes de maiores dimensões, como se pode confirmar nos Gráficos 5.7, 5.9 e 5.11.

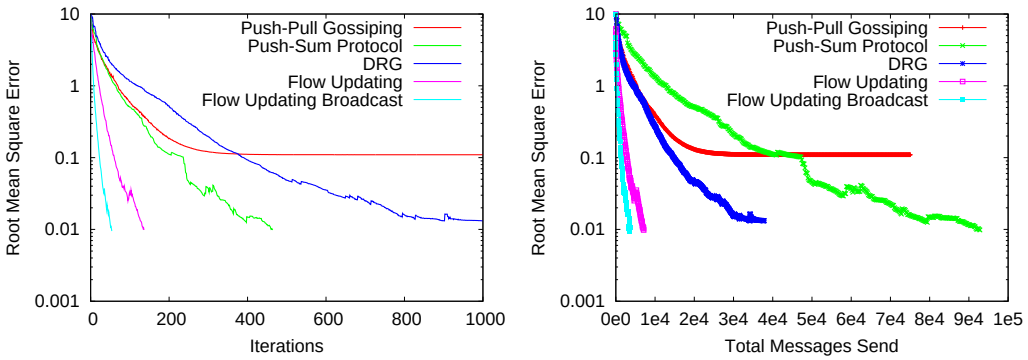
Nas simulações com grau de conectividade 10, o *Flow Updating Broad-*

Tabela 5.2: Resultados globais em redes *Attach*.

		Push-Pull Gossiping	Push-Sum Gossip	DRG	Flow Updating	Flow Updating Broadcast
	$RMSE(\hat{n})$	0,11	0,01	0,01	0,01	0,01
	$COUNT(iter)$	1000	316	727	103	38
$(n = 100)$	$COUNT(msg)$	75065	63460	27605	5238	2540
$(d = \log n)$	$MEAN(v)$	98,95	100,07	100,00	100,01	99,99
	$MIN(v)$	98,94	97,90	98,13	98,18	97,76
	$MAX(v)$	98,99	103,64	103,54	103,00	103,64
	$RMSE(\hat{n})$	0,22	0,01	0,01	0,01	0,01
	$COUNT(iter)$	250	51	69	186	43
$(n = 100)$	$COUNT(msg)$	18730	10440	2919	17554	4082
$(d = 10)$	$MEAN(v)$	93,27	100,06	100,00	99,92	99,98
	$MIN(v)$	93,27	97,47	96,67	93,14	97,58
	$MAX(v)$	93,27	103,94	104,49	100,63	102,45
	$RMSE(\hat{n})$	0,22	0,01	0,07	0,01	0,01
	$COUNT(iter)$	500	316	500	185	28
$(n = 1000)$	$COUNT(msg)$	374997	633800	190383	115513	19127
$(d = \log n)$	$MEAN(v)$	951,95	1000,14	1000,00	1000,00	999,92
	$MIN(v)$	940,83	927,13	583,12	887,49	933,96
	$MAX(v)$	974,98	1220,42	1585,25	1038,79	1105,74
	$RMSE(\hat{n})$	0,26	0,00	0,01	0,27	0,01
	$COUNT(iter)$	250	157	165	250	50
$(n = 1000)$	$COUNT(msg)$	187502	315800	68232	233017	46945
$(d = 10)$	$MEAN(v)$	867,36	1000,15	1000,02	997,00	1000,07
	$MIN(v)$	867,34	995,86	908,58	115,32	968,56
	$MAX(v)$	867,36	1146,57	1371,47	1029,36	1036,79
	$RMSE(\hat{n})$	0,19	0,02	0,30	0,03	0,01
	$COUNT(iter)$	600	572	591	528	30
$(n = 10000)$	$COUNT(msg)$	4500369	11448000	2278835	3970556	218285
$(d = \log n)$	$MEAN(v)$	9899,86	10002,80	10000,13	9999,91	9999,88
	$MIN(v)$	9844,82	7984,00	557,15	3227,66	9210,33
	$MAX(v)$	9941,17	43363,74	54857,20	10034,85	11060,86
	$RMSE(\hat{n})$	0,20	0,01	0,01	1,68	0,01
	$COUNT(iter)$	300	294	277	300	60
$(n = 10000)$	$COUNT(msg)$	2249959	5876000	1139919	2784208	557067
$(d = 10)$	$MEAN(v)$	10097,33	10001,82	9999,89	9973,84	9999,94
	$MIN(v)$	10097,21	8343,61	9774,96	67,34	9632,43
	$MAX(v)$	10097,52	50566,66	53004,52	11086,57	10392,37

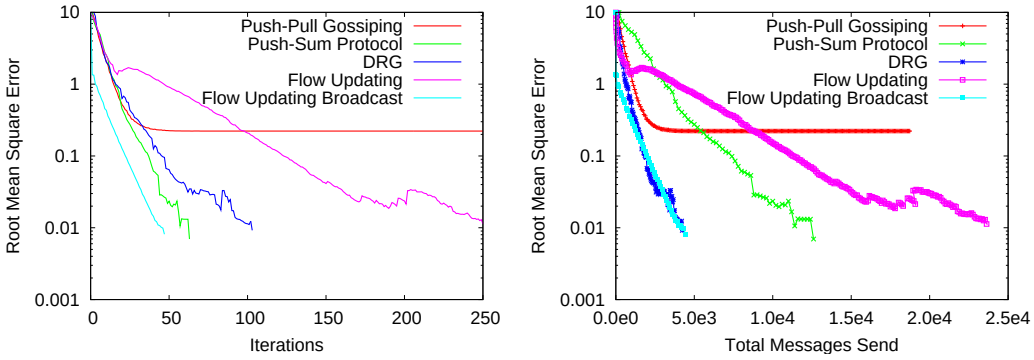
cast apresenta o melhor desempenho ao nível do tempo de execução, seguido do *Push-Sum Protocol*, *DRG* e *Flow Updating*, como se pode observar nos Gráficos 5.8(a), 5.10(a) e 5.12(a). Ao nível da comunicação, o *Flow Updating Broadcast* e *DRG* tem resultados semelhantes, sendo os algoritmos que enviam o menor número de mensagens para produzir o valor agregado final em toda a rede, com a precisão desejada (ver Gráficos 5.8(b), 5.10(b) e 5.12(b)).

O *Flow Updating* revela uma degradação do seu desempenho em cenários de simulação com redes maiores, o que não acontecia nas redes *Random*. No *Flow Updating* cada nova estimativa do valor agregado é calculada considerando todos os nós vizinhos, sendo os valores dos fluxos tendencialmente



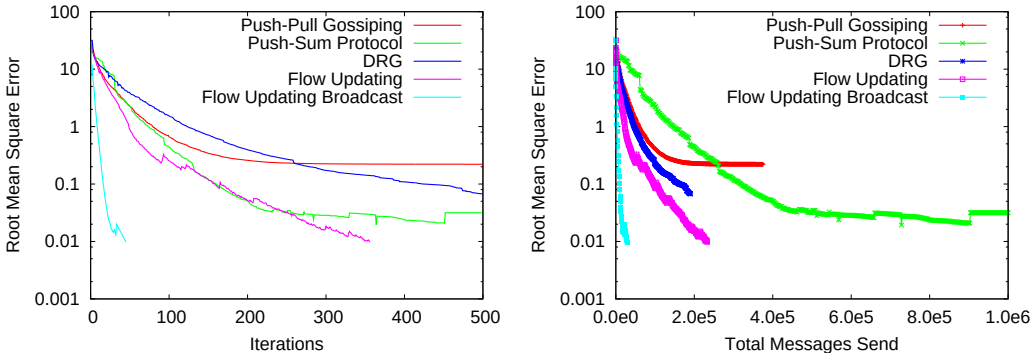
(a) RMSE Vs Iterações (b) RMSE Vs Mensagens Enviadas

Figura 5.7: Resultados numa rede *Attach* de tamanho 100 e grau 2 ($\log n$).



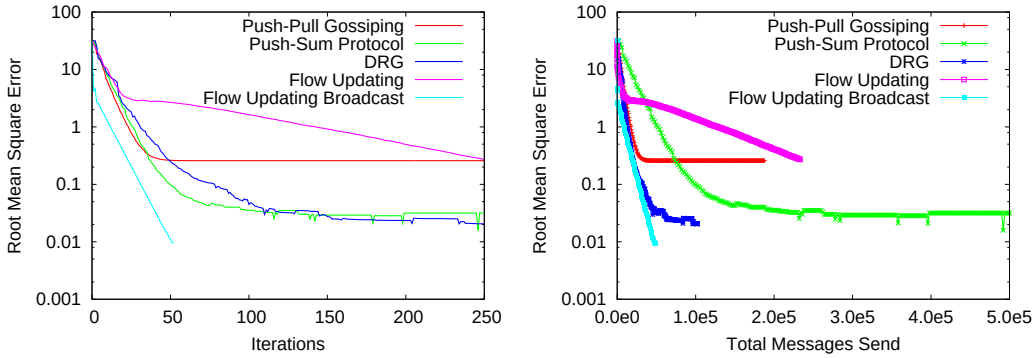
(a) RMSE Vs Iterações (b) RMSE Vs Mensagens Enviadas

Figura 5.8: Resultados numa rede *Attach* de tamanho 100 e grau 10.



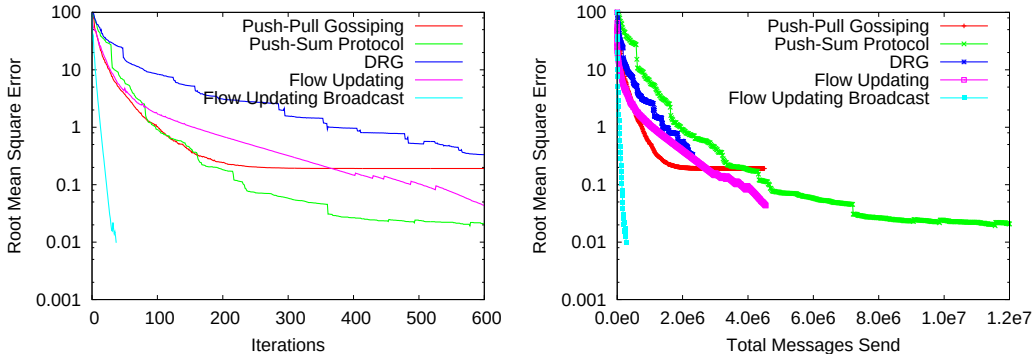
(a) RMSE Vs Iterações (b) RMSE Vs Mensagens Enviadas

Figura 5.9: Resultados numa rede *Attach* de tamanho 1000 e grau 3 ($\log n$).



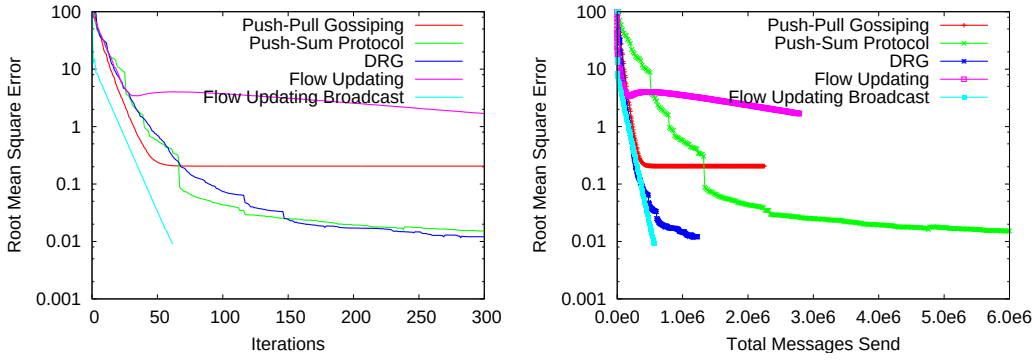
(a) RMSE Vs Iterações (b) RMSE Vs Mensagens Enviadas

Figura 5.10: Resultados numa rede *Attach* de tamanho 1000 e grau 10.



(a) RMSE Vs Iterações (b) RMSE Vs Mensagens Enviadas

Figura 5.11: Resultados numa rede *Attach* de tamanho 10000 e grau 4 ($\log n$).



(a) RMSE Vs Iterações (b) RMSE Vs Mensagens Enviadas

Figura 5.12: Resultados numa rede *Attach* de tamanho 10000 e grau 10.

mais pequenos em nós com uma maior quantidade de vizinhos. Assim sendo, neste tipo de redes onde alguns nós concentram um grande número de ligações (vizinhos), a maior divisão efectuada na determinação de uma nova estimativa do valor agregado nos nós concentradores faz com que os seus vizinhos dêem continuação ao processo partindo de estimativas mais pequenas, podendo provocar um maior desvio do valor agregado final e sendo provavelmente responsável pela degradação do desempenho do algoritmo. A verdadeira causa desta degradação será averiguada num trabalho futuro, sendo o raciocínio sugerido meramente hipotético.

Do ponto de vista da escalabilidade, os resultados nas redes *Attach* são semelhantes aos da rede *Random*. Tendo em conta a simulação em redes com quantidades de nós diferentes, o tempo de execução varia numa proporção bastante inferior ao tamanho da rede, enquanto que o número de mensagens enviadas varia na mesma proporção.

5.1.3 Redes *2D*

Nos cenários de simulação em redes *2D*, a análise dos resultados das simulações efectuadas (Tabela 5.3) evidencia o melhor desempenho das duas variantes do *Flow Updating* em todos os aspectos (tempo, comunicação e precisão), destacando-se da concorrência neste tipo de topologia.

Em cenários de simulação com grau de conectividade $\log n$, é clara a diferença de desempenho entre o *Flow Updating* (com e sem *broadcast*) e os restantes algoritmos avaliados, como se pode confirmar nos Gráficos 5.13, 5.15 e 5.17.

Nas simulações em redes com grau médio de conectividade 10, a versão *brodcast* do *Flow Updating* tira maior proveito do aumento do grau de conectividade apresentando o melhor desempenho, seguido da versão base, como se pode verificar nos Gráficos 5.14, 5.16 e 5.18.

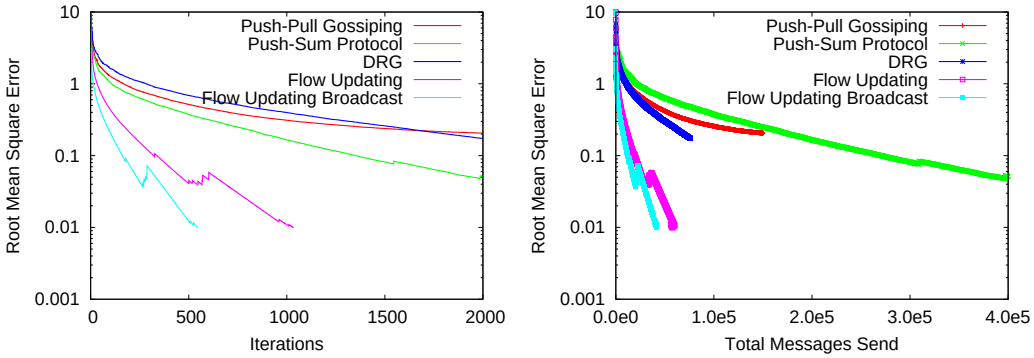
Nas redes *2D*, a influência negativa do aumento do grau de conectividade no *Flow Updating* não é verificada, ao contrário do que acontecia nas redes anteriormente analisadas (*Random* e *Attach*). Inclusive, é possível observar uma melhoria da performance do algoritmo em cenários de maior conectividade.

Neste tipo de topologia, os resultados globais de todos os algoritmos são

Tabela 5.3: Resultados globais em redes *2D*.

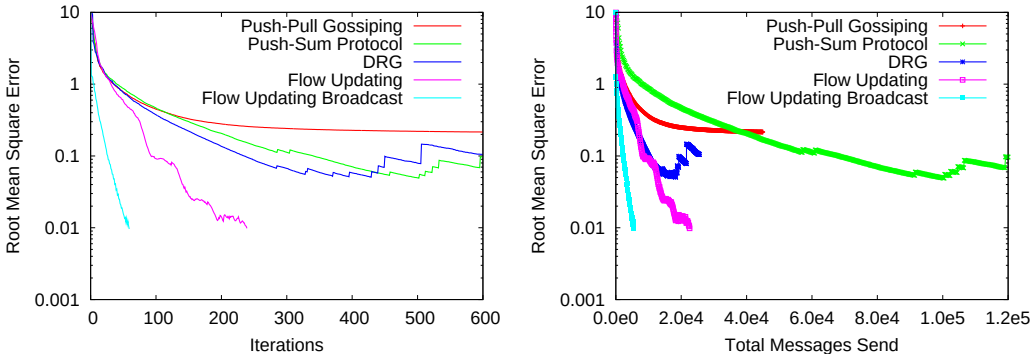
		Push-Pull Gossiping	Push-Sum Gossip	DRG	Flow Updating	Flow Updating Broadcast
	$RMSE(\hat{n})$	0,21	0,04	0,17	0,01	0,01
	$COUNT(iter)$	2000	1953	2000	662	340
$(n = 100)$	$COUNT(msg)$	149992	390600	76187	38803	26578
$(d = \log n)$	$MEAN(v)$	100,46	100,12	100,01	100,00	100,00
	$MIN(v)$	90,61	94,27	78,99	98,57	98,56
	$MAX(v)$	111,95	106,56	132,22	101,56	101,63
	$RMSE(\hat{n})$	0,22	0,03	0,02	0,01	0,01
	$COUNT(iter)$	600	490	412	175	53
$(n = 100)$	$COUNT(msg)$	44979	98160	17617	16567	5066
$(d = 10)$	$MEAN(v)$	89,07	100,06	100,00	100,01	100,00
	$MIN(v)$	87,97	96,88	97,42	97,05	96,84
	$MAX(v)$	90,25	104,13	103,05	103,09	103,47
	$RMSE(\hat{n})$	1,12	0,82	1,06	0,16	0,03
	$COUNT(iter)$	1500	1500	1500	1500	1422
$(n = 1000)$	$COUNT(msg)$	1124865	3000000	589671	1220981	1274763
$(d = \log n)$	$MEAN(v)$	937,73	1008,52	1000,13	1000,00	1000,00
	$MIN(v)$	251,59	324,14	240,93	796,97	959,00
	$MAX(v)$	14408,94	9884,06	27655,87	1445,46	1085,10
	$RMSE(\hat{n})$	0,75	0,89	0,69	0,01	0,01
	$COUNT(iter)$	700	700	700	608	167
$(n = 1000)$	$COUNT(msg)$	524803	1400000	298667	577955	156237
$(d = 10)$	$MEAN(v)$	955,94	1000,94	1000,09	999,98	1000,01
	$MIN(v)$	376,23	314,97	387,57	980,83	982,26
	$MAX(v)$	7150,71	19087,40	7394,44	1021,41	1017,62
	$RMSE(\hat{n})$	2,61	2,44	2,67	0,90	0,47
	$COUNT(iter)$	1500	1500	1500	1500	1500
$(n = 10000)$	$COUNT(msg)$	11250158	30000000	6036506	13519724	14217472
$(d = \log n)$	$MEAN(v)$	8978,97	9959,88	10000,52	10000,56	9999,99
	$MIN(v)$	605,51	626,91	501,59	2886,90	4779,63
	$MAX(v)$	32795301,61	37199920,11	212285001,25	130962,22	28250,82
	$RMSE(\hat{n})$	2,78	3,24	2,74	0,90	0,21
	$COUNT(iter)$	800	800	800	800	800
$(n = 10000)$	$COUNT(msg)$	5999841	16000000	3413133	7101503	7594466
$(d = 10)$	$MEAN(v)$	9986,59	10024,82	10000,45	9998,78	10000,10
	$MIN(v)$	597,12	454,86	607,62	2897,45	6920,95
	$MAX(v)$	226399447782	75533462541748	208310746493	182650,21	15662,81

bastante inferiores em comparação com os obtidos nas redes anteriormente analisadas. Este facto, é provavelmente devido a diferença de diâmetro das redes (distância máxima entre dois vértices da rede), sendo maior nas redes *2D*. Teoricamente, um maior diâmetro implica uma maior distância entre os nós da rede e, consequentemente a necessidade de um maior número de iterações e mensagens para fazer chegar a informação de uma extremidade da rede à outra (passando por uma maior número de vizinhos). Esta características é que devem estar na origem da baixa performance dos algoritmos neste tipo de rede em relação às anteriores.



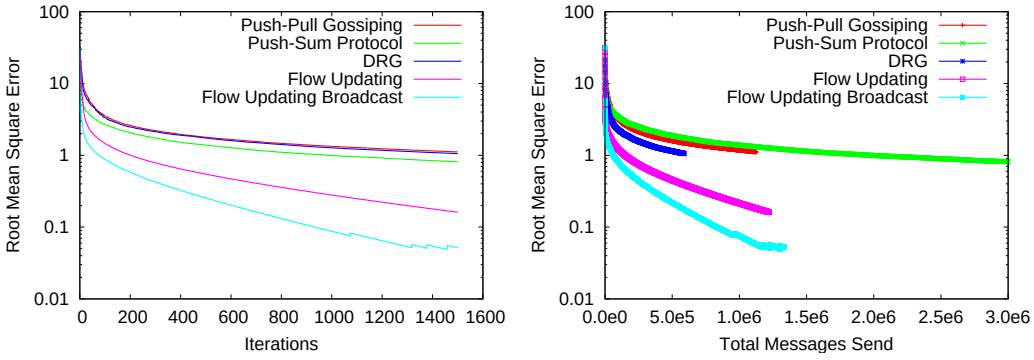
(a) RMSE Vs Iterações (b) RMSE Vs Mensagens Enviadas

Figura 5.13: Resultados numa rede *2D* de tamanho 100 e grau 2 ($\log n$).



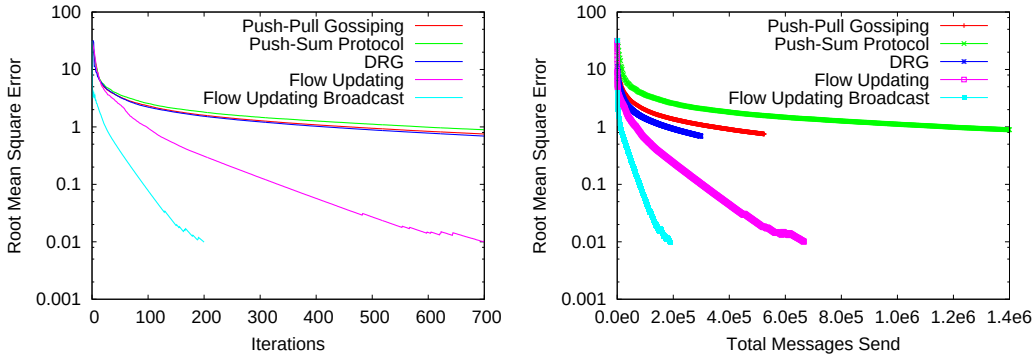
(a) RMSE Vs Iterações (b) RMSE Vs Mensagens Enviadas

Figura 5.14: Resultados numa rede *2D* de tamanho 100 e grau 10.



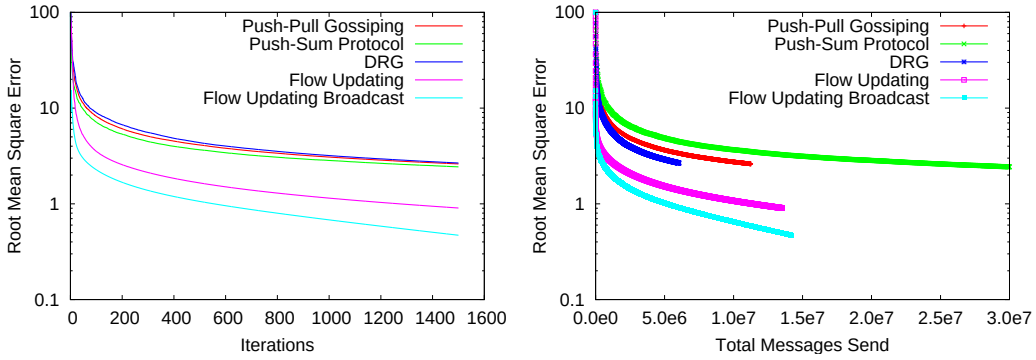
(a) RMSE Vs Iterações (b) RMSE Vs Mensagens Enviadas

Figura 5.15: Resultados numa rede *2D* de tamanho 1000 e grau 3 ($\log n$).



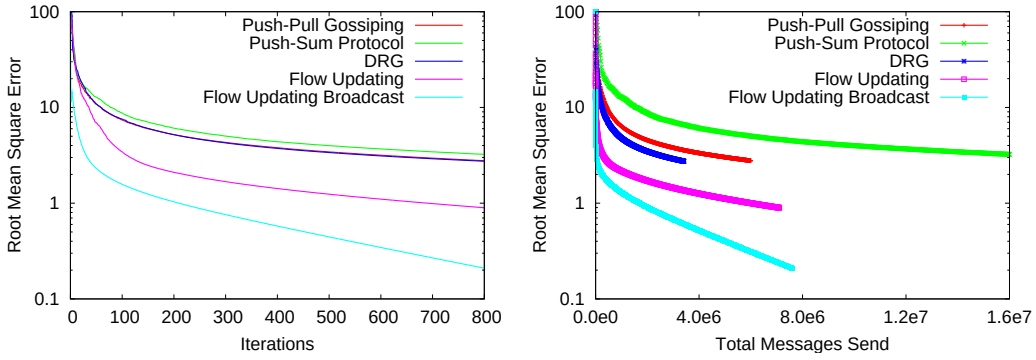
(a) RMSE Vs Iterações (b) RMSE Vs Mensagens Enviadas

Figura 5.16: Resultados numa rede *2D* de tamanho 1000 e grau 10.



(a) RMSE Vs Iterações (b) RMSE Vs Mensagens Enviadas

Figura 5.17: Resultados numa rede *2D* de tamanho 10000 e grau 4 ($\log n$).



(a) RMSE Vs Iterações (b) RMSE Vs Mensagens Enviadas

Figura 5.18: Resultados numa rede *2D* de tamanho 10000 e grau 10.

5.2 Anomalia do *Push-Pull Gossiping*

De acordo com o que já foi referido e observado na secção anterior 5.1, o *Push-Pull Gossiping* apresenta um problema de precisão, verificando-se um desvio no valor para o qual o algoritmo deveria convergir (valor exacto)[2]. Neste caso, curiosamente, a optimização de um algoritmo anterior, "forçando" a actualização de valores par a par, acabou por introduzir um problema que o seu predecessor *Push-Sum Protocol* não possui.

O *Push-Pull Gossiping* tem sido alvo de estudo e comparação em alguns artigos publicados [30, 31, 42]. No entanto, nos estudos até aqui realizados, por questões de simplicidade, nos processos de simulação a interacção entre pares de nós do algoritmo (*push-pull*) foi implementada de forma atómica, ocultando um problema latente. Contrariamente ao efectuado nos estudos anteriores, apesar do seu alto nível e simplicidade, neste estudo o simulador de rede implementa uma troca efectiva de mensagens, sendo a iteração entres pares de nós realizada através de trocas de mensagens sequenciais. Esta modulação mais realista das trocas de mensagens, desvendou a existência de um problema de atomicidade no algoritmo *Push-Pull Gossiping*, causado pela intercalação de mensagens no mesmo nó — *interleaving*.

O *Push-Pull Gossiping*, à semelhança dos restantes algoritmos de *Averaging*, baseia-se no princípio da conservação da "massa". Assim sendo, ao longo da execução, a soma global dos valores de todos os nós deve ser constante (considerando a execução em redes estáticas), tendo este invariante de ser mantido para garantir a convergência de todos os nós para o valor agregado correcto. No caso particular do *Push-Pull Gossiping*, a manutenção do invariante implica que o intercâmbio e actualização de informação par a par característico do algoritmo — *push-pull* — seja atómico. A ocorrência de qualquer tipo de actualização do valor agregado a meio do processo de *push-pull* provoca uma eventual violação do invariante.

O seguinte exemplo, ilustra o problema do *Push-Pull Gossiping*: Considere uma rede com três nós (n_1, n_2 e n_3) com os respectivos valores iniciais ($v_1 = 0$, $v_2 = 1$ e $v_3 = 0$) e, a sucessão de acções apresentadas nas Tabelas 5.4 e 5.5, representando cenários de execução distintos, de acordo com a es-

[2]Este problema foi apresentado publicamente no formato "Poster" no EuroSys 2007 [34]

pecificação do algoritmo (ver Tabela 4.2). Ambos os cenários considerados representam duas execuções válidas do processo de *push-pull* entre os mesmos nós ($n_2 \leftrightarrow n_3$ e $n_1 \leftrightarrow n_2$)[3], mas com uma ordem diferente de troca de mensagens. Na primeira situação, Tabela 5.4, a interacção entre nós é efectuada sequencialmente, garantindo a atomicidade do *push-pull*. Na segunda situação, Tabela 5.5, o *push-pull* entre nós é efectuado de forma concorrente e quase simultânea, existindo intercalação das mensagens envolvidas nos dois processos — *interleaving*. Inicialmente, a soma total dos valores da rede é igual a 1 ($v_1 + v_2 + v_3$) em ambos os cenários, no entanto o valor final é diferente ($v_1 + v_2 + v_3 = 1,25$) na execução concorrente (Tabela 5.5), verificando-se um inadvertido aumento da "massa" do sistema, comprovando a violação do princípio da conservação da "massa". A análise da evolução dos valores em cada nó da rede e, a comparação final da soma dos valores da rede entre os dois cenários, demonstra claramente a necessidade de garantir a atomicidade do *push-pull* para a manutenção do invariante do sistema.

Tabela 5.4: *Push-Pull* — Execução sequencial.

	Acções	v_1	v_2	v_3
1	Envio mensagem PUSH de n_2 para n_3 ($v' = 1$)	**0**	**1**	**0**
2	Envio mensagem PULL de n_3 para n_2 ($v' = 0$)	0	1	0
3	Actualização (push) de n_3 ($v_3 = (0 + 1)/2$)	0	1	0,5
4	Actualização (pull) de n_2 ($v_2 = (0,5 + 0)/2$)	0	0,5	0,5
5	Envio mensagem PUSH de $n1$ para n_2 ($v' = 0$)	0	0,5	0,5
6	Envio mensagem PULL de n_2 para n_1 ($v' = 1$)	0	0,5	0,5
7	Actualização (push) de n_2 ($v_2 = (1 + 0)/2$)	0	0,25	0,5
8	Actualização (pull) de n_1 ($v_1 = (0 + 1)/2$)	**0,25**	**0,25**	**0,5**

Para solucionar o problema exposto, são propostas duas soluções simples baseadas na espera e no cancelamento de mensagens, de forma a garantir a atomicidade da interacção par a par do algoritmo — *push-pull*. A primeira solução — *ordered wait* — baseia-se na espera de mensagens, fazendo com que

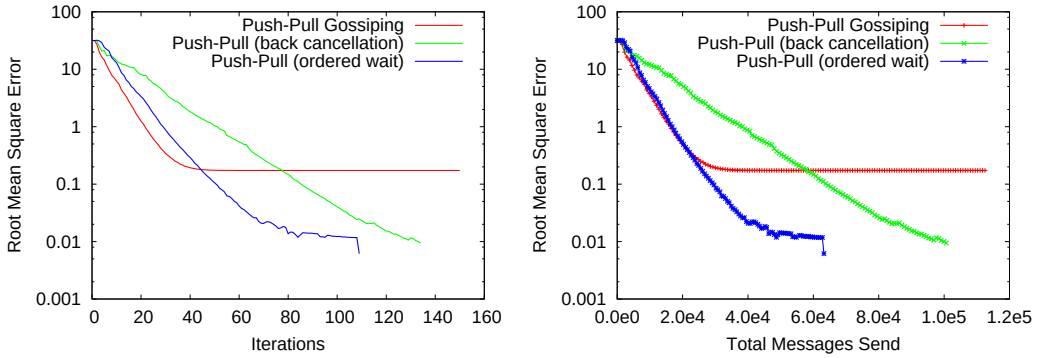
[3]Cada um dos processos de *push-pull* é representado de cor diferente nas tabelas de execução. As linhas horizontais entre as acções de cada tabela separam iterações diferentes (unidade temporal), de acordo com a definição do modelo de simulação considerado neste estudo.

Tabela 5.5: *Push-Pull* — Execução concorrente.

	Acções	v_1	v_2	v_3
1	Envio mensagem PUSH de n_2 para n_3 ($v' = 1$)	**0**	**1**	**0**
2	Envio mensagem PUSH de $n1$ para $n2$ ($v' = 0$)	0	1	0
3	Envio mensagem PULL de n_3 para n_2 ($v' = 0$)	0	1	0
4	Actualização (push) de n_3 ($v_3 = (0 + 1)/2$)	0	1	0,5
5	Envio mensagem PULL de n_2 para n_1 ($v' = 1$)	0	1	0,5
6	Actualização (push) de n_2 ($v_2 = (1 + 0)/2$)	0	0,5	0,5
7	Actualização (pull) de n_2 ($v_2 = (0,5 + 0)/2$)	0	0,25	0,5
8	Actualização (pull) de n_1 ($v_1 = (0 + 1)/2$)	**0,5**	**0,25**	**0,5**

após o envio da mensagem de *push* para o nó escolhido, todas as mensagens recebidas sejam armazenadas para posterior processamento, até à recepção da mensagem de resposta *pull*. Note-se que, por si só, esta solução pode originar situações de "bloqueio" (*deadlock*) do algoritmo, caso ocorra uma espera cíclica entre nós da rede. Por exemplo, se o nó n_1 estiver à espera do nó n_2, estando este último à espera do nó n_3, que por sua vez espera pelo primeiro n_1, ficando todos à espera uns dos outros, bloqueando assim a execução do algoritmo. Para evitar esta situação, é considerado uma relação de ordenação entre nós, podendo apenas ser efectuado o envio de mensagens *push* respeitando a ordem definida. Por exemplo, considerando a ordem $n_1 < n_2 < n_3$ e o envio de mensagem apenas para nós de ordem superior, o nó n_3 nunca poderá ficar à espera do nó n_1, uma vez que este é de ordem inferior e não lhe pode enviar mensagens *push*. A segunda solução — *cancellation* — baseia-se no cancelamento de mensagens após o envio da mensagem de *push* para o nó escolhido, fazendo com que todas as mensagens recebidas sejam ignoradas (sem actualização do valor agregado), até à recepção da mensagem de resposta *pull*. O cancelamento das mensagem (*push*) é implementado através do envio de mensagens de resposta (*pull*) com o mesmo valor agregado recebido, sem qualquer tipo de actualização do valor agregado local. O reenvio do valor recebido, faz com que a actualização no nó destino não provoque qualquer tipo de alteração no valor agregado.

Como se pode confirmar no Gráfico 5.19, a implementação das duas variantes do *Push-Pull Gossiping*, resolve o problema de atomicidade do algo-



(a) RMSE Vs Iterações (b) RMSE Vs Mensagens Enviadas

Figura 5.19: Comparação das variantes do *Push-Pull Gossiping* (rede *Random* de tamanho 1000 e grau 10).

ritmo base, ao custo de uma perda de desempenho. Como seria de esperar, de entre as duas alternativas propostas, o *Push-Pull (ordered wait)* é aquela que apresenta o melhor desempenho, enviando aproximadamente o mesmo número de mensagens que a versão base, para atingir uma estimativa com o mesmo grau de precisão (visto todas as mensagens enviadas contribuírem para determinação do valor agregado). No entanto, esta solução necessita de mais tempo (iterações) para atingir a estimativa, devido ao tempo de espera introduzido ao adiar o processamento de algumas mensagens. O *Push-Pull (cancellation)* apresenta um pior desempenho, devido ao facto do processo de cancelamento originar um grande desperdício de mensagens enviadas que não contribuem para a aproximação ao valor agregado global, atrasando a execução do algoritmo.

5.3 DRG Vs *Flow Updating Broadcast*

Nos resultados globais analisados anteriormente (Secção 5.1), é possível observar uma boa performance dos algoritmos que utilizam o *broadcast*, principalmente em relação ao número de mensagens enviadas, o que é perfeitamente justificável pelo facto do envio de numa única mensagem transmitir informação para todos os vizinhos. Em alguns cenários de comunicação, tais

como, as redes sem fios, o modo de comunicação é por natureza efectuado por *broadcast*, sendo as mensagens enviada para o meio e recebidas por qualquer dispositivo à "escuta" dentro do seu raio de alcance. Tipicamente, neste tipo de redes onde existe a partilha de um canal de comunicação, duas mensagens não podem ser transmitidas em simultâneo, causando interferência (erros) e uma eventual perda das mesmas — colisão.

Alguns algoritmos são elaborados especificamente para explorar este tipo de modelo de comunicação, como é o caso do *DRG*, criado especificamente para ser utilizado em redes de sensores sem fios. No caso particular do *DRG* (ver algoritmo 4.3), embora explore as comunicações por *broadcast*, ele não contempla por completo a ocorrência de colisões, referindo-se apenas a este problema na geração dos grupos de agregação — envio das mensagens do tipo "GCM", relacionando-o com a probabilidade de cada elemento se tornar líder. No entanto, o problema das colisões tem de ser considerado em toda a extensão do algoritmo, em todas as comunicações efectuadas. Nomeadamente, durante a criação dos grupos, todos os eventuais membros respondem ao líder — envio de mensagens do tipo "JACK", mas tendo em conta a possibilidade de ocorrência de colisões, as respostas dos membros não podem ser enviadas em simultâneo. Assim sendo, é desejável que o líder espere durante um tempo mínimo que lhe permita a recepção de todas as mensagens dos seus membros, sendo o tempo de espera proporcional ao número de membros (<*tempo de envio de uma mensagem*> × <*número de membros*>). No caso do envio da resposta do líder com o valor agregado do grupo — mensagem do tipo "GAM", a não recepção (perda/colisão) da mensagem por qualquer um dos membros pode provocar a permanência do mesmo num estado de espera indeterminado. Como tal, nesta situação é necessário recorrer ao uso de mecanismos auxiliares (*timeout* e/ou mensagens de confirmação — ACK), de forma a prevenir situação de "bloqueio" dos membros à espera de uma mensagem "perdida" (este problema não é abordado pelos autores do algoritmo).

Na análise global apresentada anteriormente (Secção 5.1), a ocorrência de colisões não foi considerada nos cenários de simulação avaliados. Tendo em conta a importância deste problema e a sua relevância em situações reais, foram configurados diferentes perfis de simulação com o objectivo de analisar

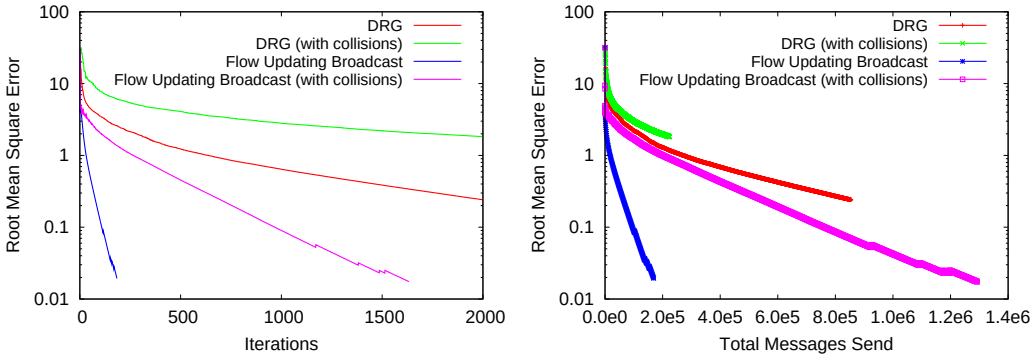
o impacto deste problema no desempenho dos algoritmos com *broadcast*, num cenário representativo de uma rede de sensores sem fios. Para isso, foi considerada uma rede *2D* com 1000 elementos e grau de conectividade médio igual a 10, onde o *DRG* e o *Flow Updating Broadcast* foram comparados. De forma a modular a ocorrência de colisões, foram alterados algumas parâmetros de simulação em cada um dos algoritmos:

- No *DRG* foi definido um tempo de espera mínimo para a recepção das mensagens de confirmação dos membros ($JACK_{timeout}$), assumindo um valor optimista e optimizado em cada nó, correspondente ao número de vizinhos de cada elemento da rede ($JACK_{timeout} = d$). Assim sendo, cada líder fica à espera um número de iterações igual ao seu número de vizinhos d, de forma a garantir a recepção de todas as mensagens de confirmação "JACK" sem ocorrência de colisões. Relativamente ao envio do resultado agregado do grupo pelo líder, à semelhança dos autores, não foi considerado o impacto do seu envio para nós vizinhos que não sejam membros do grupo.

- No *Flow Updating Broadcast* foi restringido o número de mensagens enviadas/recebidas em cada iteração por nó, sendo descartadas (perdidas) todas as mensagens para além do limite definido. Desta forma, em cada iteração, apenas foi permitido o envio e recepção de uma mensagem por nó, sendo as restantes descartadas.

Nos Gráficos 5.20 podem ser observados os resultados da simulação dos dois algoritmos numa rede de sensores sem fio, considerando dois perfis de execução distintos: sem colisões e com colisões.

Os resultados obtidos demonstram uma degradação do desempenho dos dois algoritmos perante a ocorrência de colisões. No caso do *DRG* verifica-se uma elevada degradação do desempenho, devido ao atraso na execução completa do algoritmo introduzido pelo maior tempo de espera do líder. Neste caso específico, em vez das 3 iterações necessárias para a execução completa do algoritmo numa situação óptima (sem colisões), só o processo de espera das mensagens de confirmação dos membros passa a demorar em média 10 iterações. Desta forma, a execução do algoritmo (com colisões) passa a ser



(a) RMSE Vs Iterações (b) RMSE Vs Mensagens Enviadas

Figura 5.20: Comparação do DRG com o *Flow Updating Broadcast* (rede *2D* de tamanho 1000 e grau 10 — com colisões).

completada em aproximadamente 12 iterações, ou seja 4 vezes mais, diminuindo o número de mensagens enviadas (por unidade de tempo) na mesma proporção (4 vezes menos). O *Flow Updating Broadcast* tem um desempenho bastante superior ao *DRG*, inclusive no caso da simulação com colisões, que apresenta uma melhor performance em todos os níveis que a versão sem colisões do *DRG*. Comparando os resultados dos dois perfis de simulação do *Flow Updating Broadcast* (sem/com colisões), verifica-se uma grande diferença ao nível do tempo de convergência e quantidade de mensagens enviadas para atingir a mesma precisão. Esta diferença deve-se à elevada quantidade de mensagens descartadas ($87,8\%$) no cenário com colisões, não contribuindo para a convergência do valor agregado, sendo necessário cerca de $7,6$ vezes mais iterações e enviar $8,9$ vezes mais mensagens para atingir a mesma precisão na estimativa (mesma quantidade de mensagens recebidas).

A análise dos resultados comprova que no cenário específico para o qual o *DRG* foi concebido, rede de sensores sem fios, o *Flow Updating* apresenta o melhor desempenho, ultrapassando o seu concorrente, mesmo num cenário mais problemático e realista onde é considerada a ocorrência de colisões de mensagens.

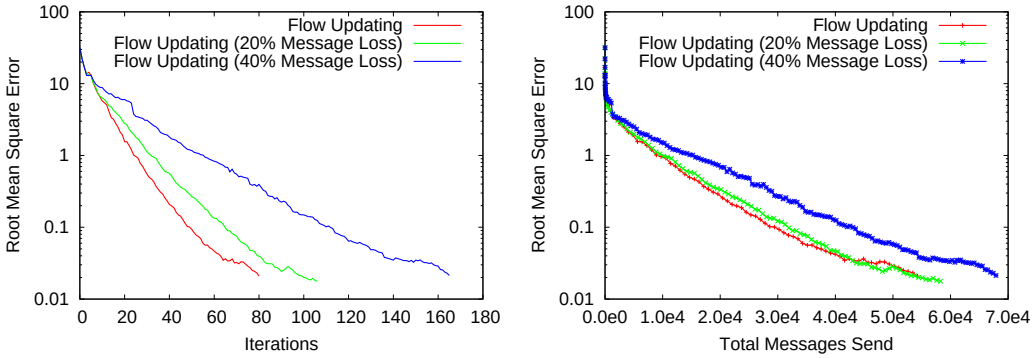
5.4 Tolerância a faltas

A robustez dos algoritmos de agregação apresentados, à semelhança de qualquer tipo de sistema distribuído, é determinada pela sua capacidade de execução em situações adversas, nomeadamente aquando da ocorrência de faltas. Existem diferentes tipos de faltas, podendo ocorrer ao nível dos próprios processo e entidades de computação (exemplo: elemento de uma rede móvel que fica sem bateria) ou ao nível da comunicação (exemplo: interferências externas ou colisões de mensagens em redes sem fios provocando a sua perda). Nesta secção é analisada a capacidade de tolerância a faltas do *Flow Updating*, considerando apenas a ocorrência de falhas não bizantinas ao nível da comunicação, ou seja, perda de mensagens.

A maioria dos algoritmos referidos neste estudo, incluindo os concorrentes analisados anteriormente (*Push-Sum Protocol*, *Push-Pull Gossiping* e *DRG*), não abordam de forma exaustiva a ocorrência de falhas de comunicação, nem as suas consequências. No *Push-Sum Protocol* é assumida a existência de um detector de faltas. Quando é detectada a perda de uma mensagem, o seu valor é reenviado para o próprio nó, para garantir a manutenção do invariante do sistema (conservação da "massa"). No *Push-Pull Gossiping* é estudada a ocorrência de faltas a diferentes níveis, afectando a qualidade da estimativa caso ocorram perdas de mensagens do tipo "PULL". Para aumentar a robustez do algoritmo perante este tipo de situação, os autores sugerem a execução de várias instâncias do algoritmo em simultâneo, provocando um aumento da carga global do sistema. No *DRG* a ocorrência de falhas de comunicação não é considerada em toda a sua extensão, como já foi referido na secção anterior 5.3.

Tipicamente, o processo de detecção e tratamento de falhas de comunicação, pode ser resolvido de forma simples através da utilização de *timeouts*, recepção de mensagens de confirmação (ACK) e reenvio de mensagens "perdidas". No entanto, a utilização deste mecanismo auxiliares tem por consequência a introdução de atrasos (aumento do número de iterações) e o aumento da comunicação no sistema (maior número de mensagens enviadas), afectando o desempenho global dos algoritmos mesmo em situações onde não

ocorra qualquer tipo de falta. Ao contrário dos demais algoritmos, o *Flow Updating* (versão com e sem *broadcast*) é imune à ocorrência de falhas de comunicação — perdas de mensagens, não necessitando de recorrer ao uso de outros mecanismo, afectando apenas o seu desempenho perante a efectiva ocorrência de faltas, como se pode confirmar nos Gráficos 5.21.



(a) RMSE Vs Iterações

(b) RMSE Vs Mensagens Enviadas

Figura 5.21: Tolerância a faltas do *Flow Updating* (rede *Random* de tamanho 1000 e grau 3 — $\log n$).

Para medir a capacidade de tolerância a faltas de comunicação do *Flow Updating*, foi simulada a execução do algoritmo numa rede *Random* com 1000 elementos e grau de conectividade 3 $(\log n)$[4], configurando três diferentes níveis de perda aleatória de mensagens em cada nó: sem perdas, 20% de perdas e 40% de perdas de mensagens. Os resultados da simulação demonstram uma degradação do desempenho do *Flow Updating* ao nível do tempo de execução e custos de comunicação provocada pela perda de mensagem, não afectando no entanto a qualidade da estimativa produzida. Em termos de tempo de execução verifica-se um atraso aproximado de 33% e 108%, para as respectivas situações de 20% e 40% de perdas de mensagens, como se pode observar no Gráfico 5.21(a). Relativamente à quantidade de mensagens trocadas, o aumento dos custos de comunicação é da mesma ordem do atraso introduzido pelas perdas de mensagens. O Gráfico 5.21(b) apresenta a relação entre as mensagens enviadas com sucesso (excluindo as mensagens perdidas) e

[4]Valor mínimo do grau de conectividade para garantir a conectividade da rede, considerando um probabilidade de falha de 1/2 em todos os nós da rede [36]

a precisão da estimativa, sendo necessário transmitir aproximadamente mais 7% e 25% de mensagens para perdas de 20% e 40% respectivamente. Esta quantidade adicional de mensagens trocadas entre nós, corresponde ao custo de comunicação adicional necessário para recuperar incorrecções da estimativa, provocadas pelas perdas de mensagens, sendo este processo realizado pelo algoritmo de forma transparente.

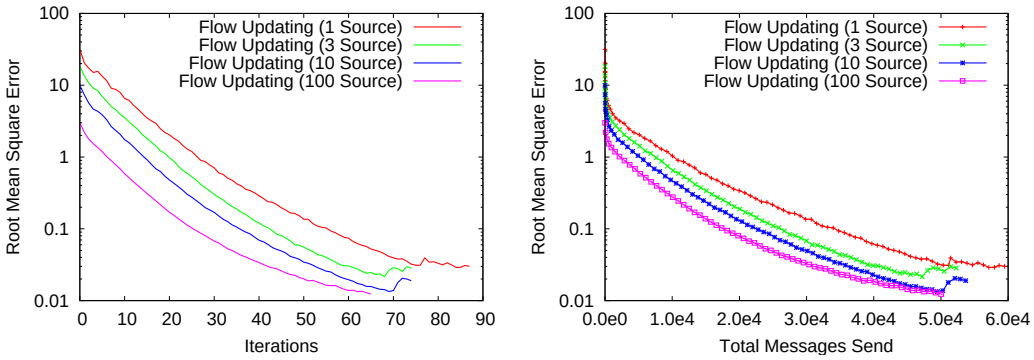
Apesar da degradação da performance introduzida pelas perdas de mensagens, o desempenho do *Flow Updating* é superior ao dos seus concorrentes a todos os níveis (tempo e comunicação). Neste caso especifico em que foi analisado, mesmo com 40% de perdas de mensagens apresenta melhores resultados que os seus adversários sem perdas de mensagens, necessitando em média de 165 iterações e enviar 114616 mensagens (das quais 45847 são perdidas), em vez das mais de 300 iterações e mais de 181400 mensagens enviadas pelos seus adversários para atingirem a mesma estimativa, como se pode confirmar na Tabela 5.1 e Gráficos 5.3.

5.5 Ilhas

Nesta secção é analisado o efeito da introdução de vários pontos de iniciação (nós com o valor 1) — ilhas, em vez da utilização de um único ponto (*source*), como foi o caso em todas as simulações expressas até ao momento. A utilização de um ponto de iniciação é necessária no caso específico da determinação do tamanho da rede — contagem. Nesta situação, utilizando os mesmos fundamentos do cálculo de médias (*averaging*) para a determinação de um valor agregado v, é possível estimar o tamanho da rede se um nó da rede for inicializado com o valor 1 e os restantes com o valor 0, sendo o resultado da estimativa da contagem dado em cada nó por $1/v$. Se em vez de apenas um nó forem inicializados c elementos com o valor 1, então o número total de elementos da rede pode ser estimado por c/v.

Nos Gráficos 5.22 é possível observar o efeito da introdução de diversas quantidades de ilhas (3, 10 e 100) na execução do *Flow Updating*, numa rede *Random* com 1000 elementos e grau de conectividade 3 ($\log n$). Os resultados visualizados indicam um aumento do desempenho do algoritmo

com o número de ilhas, principalmente na fase inicial da simulação. Este facto pode ser simplesmente explicado pela presença inicial de um maior número de nós, em vez de apenas um, envolvidos no processo de distribuição de fluxos pela rede, fazendo com que todos os nós recebam mais rapidamente contribuições e participem no processo de agregação.



(a) RMSE Vs Iterações (b) RMSE Vs Mensagens Enviadas

Figura 5.22: Utilização de Ilhas (*source*) no *Flow Updating* (rede *Random* de tamanho 1000 e grau 3 — $\log n$).

A melhoria do desempenho verificada, sugere a existência de uma vantagem em utilizar um maior número de ilhas, em vez de apenas uma. No entanto, coloca-se uma questão importante relativamente a iniciação do processo com c ilhas para a produção da estimativa correcta c/v: Como determinar o número de ilhas? Esta questão acaba por introduzir um problema idêntico ao da própria determinação do tamanho da rede, embora numa escala mais reduzida. Uma possível e simples solução para este problema, pode consistir no envio de um identificador único associado a cada ilha e, contagem em cada nó dos diferentes identificadores recebidos durante o processo de agregação. Note-se que, se todos os nós forem ilhas, o problema da determinação do número de ilhas torna-se equivalente à determinação do tamanho da rede, não existindo qualquer vantagem em recorrer ao seu uso. Em determinadas situações ou perante a definição de mecanismos de utilização mais elaborados, o recurso a utilização de ilhas pode eventualmente ser vantajoso, no entanto a exploração desta ideia carece de uma análise e estudo mais aprofundado, podendo consistir um ponto de melhoria num trabalho futuro.

Capítulo 6

Conclusões

Este estudo apresenta duas contribuições relevantes na área dos sistemas distribuídos, mais precisamente relacionadas com o problema da agregação em redes ponto a ponto. A primeira contribuição consiste na apresentação de uma taxonomia dos principais mecanismo e técnicas existentes para resolver o problema da agregação. A taxonomia sugerida está dividida de acordo com dois aspectos fundamentais: comunicação (referindo os protocolos e estruturas de comunicação usados) e computação (citando os fundamentos e princípios de computação nos quais os algoritmos se baseiam). A segunda contribuição corresponde à proposta de um novo algoritmo — *Flow Updating*, para resolver o problema da agregação, sendo apresentadas duas versões: *broadcast* e *unicast*. Para avaliar o *Flow Updating*, este foi comparado com alguns dos algoritmos mais relevantes (*Push-Sum Protocol*, *Push-Pull Gossiping* e *DRG*) pertencentes à mesma categoria, num ambiente de simulação criado especificamente para o efeito. Os resultados obtidos evidenciam um melhor desempenho global do *Flow Updating*, tanto em termos de velocidade de execução, como ao nível dos custos de comunicação.

A utilização do *Flow Updating* permite a obtenção de uma estimativa precisa em todos os nós da rede, convergindo todos para o resultado exacto, necessitando globalmente de menos iterações e de enviar menos mensagens que os restantes algoritmos de *Averaging*. A diferença de desempenho do *Flow Updating* é especialmente acentuada em redes *2D* e redes com baixo grau

conectividade médio, demonstrando uma performance notoriamente mais elevada comparativamente com os restantes algoritmos neste tipo de cenários. Para além do bom desempenho evidenciado em termos de precisão, tempo e comunicação, o *Flow Updating* apresenta ainda uma efectiva capacidade de tolerar faltas (perdas de mensagens), ao contrário dos restantes algoritmos analisados que ostentam algumas fragilidades a este nível. Perante a análise experimental efectuada, o algoritmo proposto assume-se como a melhor solução em relação aos seus concorrentes. O *Flow Updating* não depende de nenhuma estrutura de encaminhamento de rede específica, sendo uma solução robusta (tolerante a perdas de mensagens), precisa (convergindo em toda a rede para o valor exacto), rápida (com tempo de execução inferior ao da concorrência) e "leve" (envolvendo tipicamente uma menor carga de comunicação que os restantes algoritmos na disseminação de informação pela rede).

6.1 Trabalho Futuro

Apesar da boa performance revelada pelo *Flow Updating*, foram identificados alguns pontos de melhoria, nomeadamente, em relação à quebra verificada em redes (*Random* e *Attach*) com elevado grau de conectividade média. Hipoteticamente, esta redução de desempenho poderá ser devida à maior divisão (tanto maior quanto maior o grau de conectividade) e equitativa participação de todos os vizinhos directos na produção de uma nova estimativa do valor agregado (mesma para todos e tanto menor quanto maior o grau de conectividade), independentemente destes estarem ligados a mais ou menos elementos da rede. Todos os nós disseminam informação para parcelas da rede de tamanhos diferentes, necessitando de distribuir contribuições distintas (variações de "massa"), de modo a que todos os elementos em todas as parcelas tenham o mesmo valor final, idealmente ao fim do mesmo número de iterações. Assim sendo, a atribuição de "pesos" diferentes para cada vizinho, de modo a proporcionar uma participação heterogénea na estimativa de novos valores agregados, em harmonia com o fluxo que cada um necessita de disseminar para o resto da rede, poderia consistir um factor de melhoria

do algoritmo. A introdução deste conceito, pode eventualmente contribuir para uma melhoria do *Flow Updating*, necessitando de uma análise e estudo aprofundados.

A avaliação efectuada neste trabalho está restrita aos cenários de redes estáticas (quantidade e posição fixa dos elementos da rede), no entanto, tendo em conta a crescente implantação dos sistemas móveis e redes sem fios verificada, a consideração de cenários dinâmicos assume uma grande importância. Assim sendo, é necessário estudar o comportamento do *Flow Updating* em ambientes dinâmicos (contínua entrada e saída de nós da rede). Tipicamente, uma das alternativas utilizadas para lidar com este tipo de cenário resume-se simplesmente a uma periódica reinicialização do algoritmo (*drop-and-recompute*). No entanto, poderá ser vantajoso considerar a elaboração e investigação de outro mecanismo que permita uma manutenção contínua e incremental dos valores agregados, estendendo as capacidades do algoritmo.

Além das oportunidade de melhoria já referidas, a possibilidade da utilização de "Ilhas" (analisada na Secção 5.5) pode representar uma mais valia no caso específico da contagem (determinação do número de elementos). A exploração do conceito das "ilhas" poderá contribuir para uma melhoria do desempenho de diversos algoritmos de agregação.

Por fim, para fornecer a devida consistência e sustentabilidade ao novo algoritmo proposto — *Flow Updating*, para além da avaliação empírica exposta neste estudo, é necessário proceder à sua especificação formal. Esta especificação é essencial para a definição de limites e complexidade do algoritmo, facultando uma explícita afirmação das suas capacidade e âmbito de utilização.

Bibliografia

[1] The Network Simulator - ns-2.
Internet site: http://www.isi.edu/nsnam/ns/, (last access: 27 June 2007).

[2] Frank Adelstein, Frederick Hosch, Richard G. Golden III, and Loren Schwiebert. Bessie: Portable generation of network topologies for simulation. In *IC3N '98: Proceedings of the International Conference on Computer Communications and Networks*, page 787, Washington, DC, USA, 1998. IEEE Computer Society.

[3] Nabeel Ahmed, David Hadaller, and Srinivasan Keshav. Incremental maintenance of global aggregates. Technical report, University of Waterloo, June 2006.

[4] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice-Hall, 1993.

[5] Carlos Baquero, Paulo Almeida, and Raquel Menezes. Extrema propagation: Fast distributed estimation of sums and network sizes. Technical report, DI/CCTC and DMCT, University of Minho, May 2006.

[6] Vladimir Batagelj and Ulrik Brandes. Efficient generation of large random networks. *Physical Review E*, 71(036113), 2005.

[7] M. Bawa, H. Garcia-Molina, A. Gionis, and R. Motwani. Estimating aggregates on a peer-to-peer network. Technical report, Stanford University, Computer Science Department, 2003.

[8] Yitzhak Birk, Idit Keidar, Liran Liss, Assaf Schuster, and Ran Wolff. Veracity radius: capturing the locality of distributed computations. In *PODC '06: Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 102–111, New York, NY, USA, 2006. ACM Press.

[9] Béla Bollobás. *Modern Graph Theory*, volume 184 of *Graduate Texts in Mathematics*. Springer, 1998.

[10] Kenneth L. Calvert, Matthew B. Doar, and Ellen W. Zegura. Modeling Internet topology. *IEEE Communications Magazine*, 35:160–163, June 1997.

[11] G. Chen, J. Branch, M. J. Pflug, L. Zhu, and B. Szymanski. *Advances in Pervasive Computing and Networking*, chapter SENSE: A sensor network simulator, pages 249–267. Springer-Verlag, 2005.

[12] Jen-Yeu Chen, Gopal Pandurangan, and Dongyan Xu. Robust computation of aggregates in wireless sensor networks: Distributed randomized algorithms and analysis. *IEEE Transactions on Parallel and Distributed Systems*, 17(9):987–1000, 2006.

[13] Jeffrey Considine, Feifei Li, George Kollios, and John Byers. Approximate aggregation techniques for sensor databases. In *ICDE '04: Proceedings of the 20th International Conference on Data Engineering*, page 449, Washington, DC, USA, 2004. IEEE Computer Society.

[14] Anirban DasGupta. The matching, birthday and the strong birthday problem: a contemporary review. *Journal of Statistical Planning and Inference*, 130(1-2):377–389, March 2005.

[15] Shlomi Dolev, Elad Schiller, and Jennifer Welch. Random walk for self-stabilizing group communication in ad-hoc networks. In *SRDS '02: Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems (SRDS'02)*, page 70, Washington, DC, USA, 2002. IEEE Computer Society.

[16] Shlomi Dolev, Elad Schiller, and Jennifer L. Welch. Random walk for self-stabilizing group communication in ad hoc networks. *IEEE Transactions on Mobile Computing*, 05(7):893–905, 2006.

[17] D. Dreier. Manual of operation: Barabasi graph generator v1.0. Technical report, University of California Riverside, Department of Computer Science, January 2002.

[18] Jeremy Elson, Solomon Bien, Naim Busek, Vladimir Bychkovskiy, Alberto Cerpa, Deepak Ganesan, Lewis Girod, Ben Greenstein, Thomas Schoellhammer, Thanos Stathopoulos, and Deborah Estrin. Emstar: An environment for developing wireless embedded systems software. Technical Report CENS 0009, Center for Embedded Networked Sensing, University of California, Los Angeles, March 2003.

[19] P. Th. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec. Lightweight probabilistic broadcast. *ACM Trans. Comput. Syst.*, 21(4):341–374, 2003.

[20] Philippe Flajolet and G. Nigel Martin. Probabilistic counting. In *FOCS*, pages 76–82. IEEE, 1983.

[21] Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985.

[22] Ayalvadi J. Ganesh, Anne-Marie Kermarrec, and Laurent Massoulié. Peer-to-peer membership management for gossip-based protocols. *IEEE Trans. Comput.*, 52(2):139–149, 2003.

[23] Ali Ghodsi, Sameh El-Ansary, Supriya Krishnamurthy, and Seif Haridi. A self-stabilizing network size estimation gossip algorithm for peer-to-peer systems. Technical report, Swedish Institute of Computer Science, Stockholm, 2005.

[24] Lewis Girod, Jeremy Elson, Alberto Cerpa, Thanos Stathopoulos, Nithya Ramanathan, and Deborah Estrin. Emstar: A software environment for developing and deploying wireless sensor networks. In *USENIX*

Annual Technical Conference, General Track, pages 283–296. USENIX, 2004.

[25] Lewis Girod, Thanos Stathopoulos, Nithya Ramanathan, Jeremy Elson, Deborah Estrin, Eric Osterweil, and Tom Schoellhammer. A system for simulation, emulation, and deployment of heterogeneous sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 201–213, New York, NY, USA, 2004. ACM Press.

[26] Jonathan L. Gross and Jay Yellen. *Handbook of Graph Theory*. Discrete Mathematics and Its Applications Series Editor Kenneth H. Rosen. CRC Press, 2004.

[27] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. In *ASPLOS-IX: Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*, pages 93–104, New York, NY, USA, 2000. ACM Press.

[28] Keren Horowitz and Dahlia Malkhi. Estimating network size from local information. *Inf. Process. Lett.*, 88(5):237–243, 2003.

[29] Márk Jelasity, Wojtek Kowalczyk, and Maarten van Steen. An approach to massively distributed aggregate computing on peer-to-peer networks. In *Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP'04)*, pages 200–207, A Coruna, Spain, 2004. IEEE Computer Society.

[30] Márk Jelasity and Alberto Montresor. Epidemic-style proactive aggregation in large overlay networks. In *ICDCS '04: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, pages 102–109, Washington, DC, USA, 2004. IEEE Computer Society.

[31] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Trans. Comput. Syst.*, 23(3):219–252, 2005.

[32] Márk Jelasity and Mike Preuß. On obtaining global information in a peer-to-peer fully distributed environment. In B. Monien and R. Feldman, editors, *Euro-Par 2002 Parallel Processing, Proc. Eighth Int'l Conf., Paderborn, August 2002*, volume 2400 of *Lecture Notes in Computer Science*, pages 573–577, Berlin, 2002. Springer.

[33] Paulo Jesus, Carlos Baquero, and Paulo Almeida. ID generation in mobile environments (abstract). In *Conference on Mobile and Ubiquitous Systems - CSMU2006*, Guimarães, Portugal, June 2006.

[34] Paulo Jesus, Carlos Baquero, and Paulo Almeida. A study on aggregation by averaging algorithms (poster). In *EuroSys 2007 - 2nd EuroSys Conference*, Lisbon, Portugal, 21-23 March 2007.

[35] Tomasz Jurdzinski, Miroslaw Kutylowski, and Jan Zatopianski. Energy-efficient size approximation of radio networks with no collision detection. In *COCOON '02: Proceedings of the 8th Annual International Conference on Computing and Combinatorics*, pages 279–289, London, UK, 2002. Springer-Verlag.

[36] M. Frans Kaashoek and David R. Karger. Koorde: A simple degree-optimal distributed hash table. In M. Frans Kaashoek and Ion Stoica, editors, *IPTPS*, volume 2735 of *Lecture Notes in Computer Science*, pages 98–107. Springer, 2003.

[37] Jedrzej Kabarowski, Miroslaw Kutylowski, and Wojciech Rutkowski. Adversary immune size approximation of single-hop radio networks. In *TAMC 2006: Proceedings of the 3th International Conference on Theory and Applications of Models of Computation*, pages 148–158, Beijing, China, May 2006. Springer Berlin / Heidelberg.

[38] David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-based computation of aggregate information. In *FOCS '03: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, page 482, Washington, DC, USA, 2003. IEEE Computer Society.

[39] Srinivasan Keshav. REAL: A network simulator. Technical report, University of California, Berkeley, CA, USA, 1988.

[40] Dionysios Kostoulas, Dimitrios Psaltoulis, Indranil Gupta, Ken Birman, and Al Demers. Decentralized schemes for size estimation in large and dynamic groups. In *NCA '05: Proceedings of the Fourth IEEE International Symposium on Network Computing and Applications*, pages 41–48, Washington, DC, USA, 2005. IEEE Computer Society.

[41] Miroslaw Kutylowski and Daniel Letkiewicz. Computing average value in ad hoc networks. In Branislav Rovan and Peter Vojtás, editors, *MFCS*, volume 2747 of *Lecture Notes in Computer Science*, pages 511–520. Springer, 2003.

[42] E. Le Merrer, A. M. Kermarrec, and L. Massoulié. Peer to peer size estimation in large and dynamic networks: A comparative study. In *15th IEEE International Symposium on High Performance Distributed Computing*, pages 7–17, 2006.

[43] Ji Li, Karen Sollins, and Dah-Yoh Lim. Implementing aggregation and broadcast over distributed hash tables. *SIGCOMM Comput. Commun. Rev.*, 35(1):81–92, 2005.

[44] Samuel Madden, Robert Szewczyk, Michael J. Franklin, and David Culler. Supporting aggregate queries over ad-hoc wireless sensor networks. In *WMCSA '02: Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications*, page 49, Washington, DC, USA, 2002. IEEE Computer Society.

[45] Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: a scalable and dynamic emulation of the butterfly. In *PODC '02: Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 183–192, New York, NY, USA, 2002. ACM Press.

[46] Sandeep Mane, Sandeep Mopuru, Kriti Mehra, and Jaideep Srivastava. Network size estimation in a peer-to-peer network. Technical report,

Department of Computer Science - University of Minnesota, September 2005.

[47] Gurmeet S. and Manku. Symphony: Distributed hashing in a small world. In *USITS '03*, 2003.

[48] Gurmeet Singh Manku. Routing networks for distributed hash tables. In *PODC '03: Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 133–142, New York, NY, USA, 2003. ACM Press.

[49] Laurent Massoulié, Erwan Le Merrer, Anne-Marie Kermarrec, and Ayalvadi Ganesh. Peer counting and sampling in overlay networks: random walk methods. In *PODC '06: Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 123–132, New York, NY, USA, 2006. ACM Press.

[50] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. BRITE: An approach to universal topology generation. In *MASCOTS '01: Proceedings of the Ninth International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'01)*, page 346, Washington, DC, USA, 2001. IEEE Computer Society.

[51] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. BRITE: Universal topology generation from a user's perspective. Technical report, Boston University, Boston, MA, USA, 2001.

[52] Alberto Medina, Ibrahim Matta, and John Byers. On the origin of power laws in internet topologies. *SIGCOMM Comput. Commun. Rev.*, 30(2):18–28, 2000.

[53] Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, and Zhichen Xu. Peer-to-peer computing. Technical report, HP Labs, 2002.

[54] Damon Mosk-Aoyama and Devavrat Shah. Computing separable functions via gossip. In *PODC '06: Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 113–122, New York, NY, USA, 2006. ACM Press.

[55] J. O'Madadhain, D. Fisher, S. White, and Y. Boey. The JUNG (Java Universal Network/Graph) framework. Technical report, UC Irvine, 2003.

[56] J. Polley, D. Blazakis, J. Mcgee, D. Rusk, and J. S. Baras. ATEMU: a fine-grained sensor network simulator. In *IEEE SECON 2004, First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*, pages 145–152, 2004.

[57] Dimitrios Psaltoulis, Dionysios Kostoulas, Indranil Gupta, Ken Birman, and Al Demers. Practical algorithms for size estimation in large and dynamic groups. Technical report, University of Illinois, Urbana-Champaign, 2004.

[58] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM Press.

[59] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350, London, UK, 2001. Springer-Verlag.

[60] C. J. Schwarz and G. A. F. Seber. Estimating animal abundance: review III. *Statistical Science*, 14:427–56, 1999.

[61] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet

applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160, New York, NY, USA, 2001. ACM Press.

[62] Sameer Sundresh, Wooyoung Kim, and Gul Agha. Sens: A sensor, environment and network simulator. In *ANSS '04: Proceedings of the 37th annual symposium on Simulation*, page 221, Washington, DC, USA, 2004. IEEE Computer Society.

[63] Ben Titzer. Avrora: The AVR Simulation and Analysis Framework. Master's thesis, UCLA, June 2004.

[64] Ben L. Titzer, Daniel K. Lee, and Jens Palsberg. Avrora: scalable sensor network simulation with precise timing. In *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, page 67, Piscataway, NJ, USA, 2005. IEEE Press.

[65] Robbert van Renesse. The importance of aggregation. In André Schiper, Alexander A. Shvartsman, Hakim Weatherspoon, and Ben Y. Zhao, editors, *Future Directions in Distributed Computing*, volume 2584 of *Lecture Notes in Computer Science*, pages 87–92. Springer, 2003.

[66] Mark Weiser. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3(3):3–11, 1999.

[67] Jared Winick and Sugih Jamin. Inet-3.0: Internet topology generator. Technical Report UM-CSE-TR-456-02, EECS, University of Michigan, 2002.

[68] Ellen W. Zegura, Kenneth L. Calvert, and Samrat Bhattacharjee. How to model an internetwork. In *INFOCOM*, pages 594–602, 1996.

[69] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical report, University of California at Berkeley, Berkeley, CA, USA, 2001.