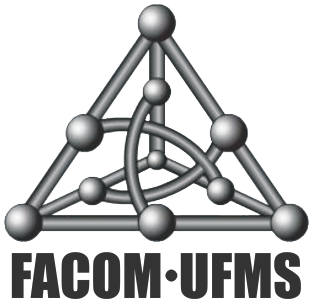
Busca Exaustiva em Redes P2P

Péricles Christian Moraes Lopes

Dissertação de Mestrado

Orientação: Prof. Ronaldo Alves Ferreira

Área de Concentração: Sistemas Distribuídos



Faculdade de Computação
Universidade Federal de Mato Grosso do Sul
16 de setembro de 2010

Universidade Federal de Mato Grosso do Sul
Faculdade de Computação

Péricles Christian Moraes Lopes
Orientador: Prof. Ronaldo Alves Ferreira

Busca Exaustiva em Redes P2P

Este exemplar corresponde à redação final da dissertação devidamente corrigida e defendida por Péricles Christian Moraes Lopes e aprovada pela comissão julgadora.

Comissão Julgadora:

- Prof. Ronaldo Alves Ferreira (FACOM/UFMS)
- Prof. Edmundo Roberto Mauro Madeira (IC-UNICAMP)
- Prof. Irineu Sotoma (FACOM/UFMS)

Campo Grande, MS
Setembro de 2010

Agradecimentos

Aos meus pais e família, pelo apoio e participação em toda a minha formação. Agradeço por todas as condições oferecidas e por me mostrar desde cedo a importância e valor dos estudos.

À minha amada Ana, pela compreensão, apoio e carinho durante todo esse período. Os dias se tornaram mais suaves a seu lado.

Ao professor Ronaldo Alves Ferreira, pela orientação digna durante o desenvolvimento deste trabalho, pelos conselhos sábios e oportunidades oferecidas durante o mestrado. Agradeço por não medir esforços, pela paciência, por estar sempre disposto a ajudar quando requisitado e ainda despertar em mim o senso da pesquisa e a importância de se realizar um trabalho com qualidade.

Aos professores da Faculdade de Computação da UFMS, em especial os professores Édson Norberto Cáceres, Henrique Mongelli e Marcelo Henriques de Carvalho, que colaboraram direta e indiretamente com a minha formação e me transmitiram conhecimento suficiente para que eu tivesse a capacidade em realizar este trabalho.

Aos professores Irineu Sotoma (FACOM/UFMS), Edmundo Roberto Mauro Madeira (IC-Unicamp) e Dorgival Guedes (DCC/UFMG) pelos conselhos e recomendações acerca da dissertação.

Aos funcionários da Faculdade de Computação (FACOM/UFMS), do Núcleo de Informática (NIN/UFMS) e do Ponto de Presença da Internet em Mato Grosso do Sul (POP-MS) que me forneceram suporte para o desenvolvimento deste trabalho.

Aos meus colegas do mestrado, pelas horas de estudo, dedicação, conversas, alegrias e aflições compartilhadas durante esta fase da minha vida.

À FUNDECT pelo apoio financeiro.

A todas as pessoas que de forma direta ou indireta colaboraram para o desenvolvimento deste trabalho.

"Ao final do jogo o rei e o peão voltam para a mesma caixa."

(Provérbio italiano)

Resumo

Apesar de inúmeros esforços nos últimos anos, buscas complexas eficientes em redes P2P de grande escala permanecem um problema em aberto e desafiador. Replicações massivas de dados e de mensagens de buscas são duas estratégias comuns utilizadas para melhorar taxas de sucesso e tempos de resposta das diversas técnicas propostas. Entretanto, estratégias de replicação pró-ativas podem gerar uma quantidade significativa de tráfego na rede se não forem tratadas com cuidado. Este trabalho propõe SplitQuest, um protocolo de busca exaustiva e controlada que utiliza uma estrutura leve para evitar replicações desnecessárias e acelerar a propagação de mensagens de buscas em redes P2P. Em SplitQuest, os pares da rede são organizados em grupos de replicação, nos quais cada par compartilha seu conteúdo com todos os demais membros e mensagens de buscas são propagadas uma única vez para cada grupo existente na rede.

Ao evitar duplicações de mensagens de buscas, direcionar as mensagens para grupos disjuntos e explorar a heterogeneidade dos pares, SplitQuest é capaz de atingir altas taxas de sucesso e baixos tempos de resposta a custo bem mais baixo em termos do número de mensagens que a melhor solução atualmente conhecida. O protocolo SplitQuest foi avaliado em diferentes cenários de simulação. Esses cenários incluem topologias sintéticas e traços de representações de redes reais com os mais variados tamanhos e características de dinamismo. Apresenta-se também neste trabalho, uma análise matemática da solução proposta para se estabelecer um limite superior no número de pares que uma mensagem de busca pode alcançar na rede.

Palavras-Chave: *redes peer-to-peer, buscas, algoritmos distribuídos, topologia, simulação.*

Abstract

Despite numerous efforts in the past few years, efficient complex queries in large-scale P2P networks remain an open and challenging problem. Massive data and query replications are two popular techniques used to improve success rates and response times. However, proactive replication strategies may lead to large amounts of traffic and low efficiency if not handled with care. This work presents SplitQuest, a controlled and exhaustive search protocol that relies on a lightweight network structure to avoid unnecessary query replication and to speed up query propagation in unstructured P2P networks. In SplitQuest, peers are organized in replication groups, in which each peer shares its contents with all members, and queries are propagated only once to a group.

By avoiding query duplication, directing queries to disjoint groups, and exploiting peers' heterogeneity, SplitQuest is able to achieve high levels of recalls and low response times, while incurring very low overhead. SplitQuest protocol was simulated using synthetic and traces of real-world topologies and show that it outperforms the best known solution in number of messages, response time, number of hops, and success rate for query resolution, while being resilient to high churn rates. It was also derived an upper bound on query routing for SplitQuest.

Palavras-Chave:*peer-to-peer networks, search, distributed algorithms, topology, simulation.*

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A utilização de sistemas par-a-par (P2P - *peer-to-peer*) é cada vez mais visível em nosso cotidiano. Diversas são as aplicações desenvolvidas para o dia-a-dia caracterizadas por esse tipo de sistema, tais como compartilhamento de recursos (músicas,vídeos) [2, 7, 1], troca de mensagens instantâneas [8, 5], computação distribuída [3, 10], *streaming* [12, 6] etc. A rápida expansão na quantidade de aplicações desenvolvidas se deve às inúmeras vantagens oferecidas pelas redes que formam a base desses sistemas. Uma rede P2P é caracterizada não só pela troca mútua de serviços entre seus participantes, mas também por oferecer suporte à comunicação entre participantes com capacidades de transmissão distintas, autonomia parcial ou total com relação a um servidor centralizado e escalabilidade.

Uma característica chave em sistemas P2P é o estabelecimento de conexões pelos pares na formação da rede sobreposta. Essa característica deve ser levada em consideração no desenvolvimento de algoritmos e aplicações suportadas pela rede, já que a estrutura estabelecida tem influência direta em aspectos de desempenho, robustez e escalabilidade. De maneira mais específica, a estrutura da rede afeta diretamente questões como garantia na localização de dados, roteamento de mensagens e definem a classificação dos sistemas P2P em modelos estruturado e não estruturado. As redes estruturadas se caracterizam por um conjunto de computadores dispostos em uma topologia específica e previamente determinada. As topologias dessas redes seguem os mais variados modelos, tais como Hipercubo [65], Malha [57], Butterfly [45], Grafos de Bruijn [32], etc. A introdução de estrutura em uma rede torna os eventos de entrada e saída de nós críticos em razão da necessidade de manutenção da topologia previamente determinada. Em compensação, a estrutura permite que fortes garantias sejam oferecidas na localização de um recurso, caso este esteja disponível em algum participante. Uma rede P2P não estruturada, por outro lado, não utiliza topologia específica e as conexões dos nós são realizadas de forma aleatória. Esse tipo de rede é caracterizado pela grande robustez a falhas dos nós (visto que a falha de um participante compromete uma parte ínfima da rede) e também pelo baixo custo necessário na manutenção da topologia (não há necessidade de se manter uma estrutura específica da rede).

Apesar de inúmeros esforços nos últimos anos, realizar buscas complexas de maneira

eficiente em redes P2P de larga escala ainda pode ser considerado um problema em aberto e desafiador. Grande parte das pesquisas até então desenvolvidas apresenta estratégias pouco eficientes, como a utilização de técnicas de inundação da rede com mensagens de busca ou replicação em larga escala dos dados [4, 20, 44]. O protocolo de busca de Gnutella [4], por exemplo, realiza o espalhamento de mensagens por todos os seus vizinhos com o objetivo de atingir o maior número possível de participantes. Essa estratégia ocasiona um aumento considerável no número de mensagens inseridas na rede. Além disso, uma busca que utiliza inundação pode fazer com que um nó[1] receba a mesma mensagem de busca várias vezes, já que não há controle no espalhamento das mensagens e as conexões entre os nós são estabelecidas aleatoriamente.

Uma solução escalável para esse tipo de busca foi inicialmente proposta por *Ferreira et al.* em [25]. O algoritmo proposto é baseado no paradoxo do aniversário e oferece fortes garantias probabilísticas para localização de um objeto na rede. A idéia básica é que um nó instala referências para seus dados em um conjunto de outros nós aleatórios escolhidos uniformemente. A busca é realizada de modo semelhante escolhendo um conjunto de nós também aleatórios e uniformes na rede. A interseção desses dois conjuntos forma a base da solução. Esse trabalho mostra que se as cardinalidades dos conjuntos forem proporcionais a $O(\sqrt{n})$, em que n é o número de nós presentes na rede, a probabilidade de interseção tende a um. Uma desvantagem dessa proposta é o uso de caminhadas aleatórias sequenciais, pois o tempo de resposta aumenta consideravelmente quando o tamanho da rede aumenta. Em um trabalho mais recente, Terpstra *et al.* [71] melhora em vários aspectos a solução proposta em [25]. A solução proposta em [71] também utiliza caminhadas aleatórias para realizar uma busca exaustiva pela rede mas, ao contrário de [25], uma mensagem de busca da caminhada é replicada a cada passo em s (parâmetro do sistema) vizinhos distintos. Apesar dessa estratégia permitir que buscas sejam desempenhadas mais rapidamente que em [25], para um bom desempenho do algoritmo é necessário a instalação de uma grande quantidade de cópias de dados e mensagens de buscas na rede.

Este trabalho propõe SplitQuest, um protocolo de busca exaustiva e controlada que introduz uma estrutura leve na rede de modo a evitar replicações desnecessárias de mensagens e aumentar a velocidade de propagação de buscas em redes P2P não estruturadas. Em SplitQuest, os nós são organizados em grupos de replicação, em que cada nó compartilha o seu conteúdo com todos os membros do grupo e uma mensagem de busca é propagada apenas uma vez para um grupo de replicação específico. Essa estratégia de propagação, além de evitar mensagens de buscas desnecessárias, é ainda capaz de explorar a heterogeneidade dos nós dessa rede já que possibilita a propagação de mensagens de acordo com a capacidade de cada participante.

Neste trabalho, SplitQuest é avaliado por simulações com topologias sintéticas e topologias coletadas de redes reais. Os resultados obtidos mostram que o algoritmo proposto apresenta desempenho superior ao melhor algoritmo de busca exaustiva conhecido na literatura [71]. As simulações desenvolvidas avaliam os desempenhos dos algoritmos quanto a número de mensagens, tempo de resposta, número de saltos e taxa de sucesso das buscas. O trabalho ainda apresenta uma análise matemática que estabelece o limite superior da distância que uma mensagem de busca pode percorrer na rede. Os principais resultados

[1]Neste trabalho, as palavras nó, par e participante são usadas como sinônimos.

deste trabalho foram recentemente publicados em [41] e [42].

O restante do trabalho possui a seguinte organização: o Capítulo 2 apresenta os conceitos básicos de redes P2P e os fundamentos das arquiteturas mais conhecidas na literatura. Nesse capítulo, são apresentados também os conceitos do fenômeno de "Small World", cenário comum das redes P2P atuais e utilizado como modelo nas simulações realizadas neste trabalho. O Capítulo 3 apresenta SplitQuest, um protocolo de buscas para redes P2P. Esse protocolo inclui métodos para replicação de índices, propagação de mensagens de buscas e atribuição de identificadores. Esse capítulo apresenta ainda um método para controle de fluxo de mensagens e uma análise matemática, incluindo um limite superior assintótico para as mensagens de buscas propagadas em SplitQuest. O Capítulo 4 caracteriza o ambiente de simulação utilizado para avaliação do protocolo proposto e os detalhes técnicos das implementações do simulador utilizado neste trabalho. São apresentados ainda nesse capítulo os resultados obtidos nas simulações e as comparações com BubbleStorm–a melhor solução conhecida para buscas complexas em redes P2P. Já o Capítulo 5 apresenta as conclusões e propostas de trabalhos futuros relacionadas ao protocolo SplitQuest. No Apêndice A, são apresentados alguns fundamentos matemáticos utilizados na análise matemática do algoritmo proposto. O Apêndice B apresenta os detalhes de implementação do simulador desenvolvido.

Capítulo 2

Fundamentação Teórica

Uma rede é dita P2P se os seus participantes estão conectados e em operação sem a necessidade de um servidor central. Os participantes são simétricos e ora atuam como clientes (quando realizam requisições por serviços), ora como servidores (quando atendem às requisições). São inúmeras as vantagens de uma rede com essas características, como por exemplo, descentralização de tarefas, maior resistência a falhas e ataques maliciosos além da utilização conjunta dos computadores para armazenamento de dados e processamento de informações. Por outro lado, a ausência de um servidor central para controlar o seu funcionamento torna os processos de administração da rede e localização de dados complexos.

Na Seção 2.1 são apresentados fundamentos relacionados ao trabalho proposto. São conceituados durante a seção importantes definições utilizadas no restante deste trabalho como redes de sobreposição, replicação, buscas, etc. A Seção 2.2 apresenta as principais arquiteturas P2P existentes na literatura e detalha os principais trabalhos propostos para cada uma dessas arquiteturas. Fazem parte dessa seção as arquiteturas centralizada, descentralizada estruturada e descentralizada não estruturada. A Seção 2.3 apresenta alguns dos principais algoritmos, baseados em arquitetura de redes estruturadas, existentes na literatura. Já a Seção 2.4 apresenta algoritmos baseados na arquitetura de redes não-estruturadas. Na Seção 2.5 são apresentados ainda os conceitos e trabalhos relacionados ao fenômeno de *Small-World*–considerado um modelo realista de redes P2P e utilizado nas simulações realizadas neste trabalho.

2.1 Fundamentos de Redes P2P

Uma rede P2P pode ser visualizada como um grafo. Os vértices representam os nós participantes e a arestas as conexões estabelecidas por esses nós pela utilização de uma aplicação em comum. As conexões estabelecidas por esses nós levam à formação de topologias com propriedades particulares e estão normalmente associadas a técnicas específicas de replicação e busca de dados nessa rede. Nesta seção, serão introduzidos conceitos básicos de redes P2P, tais como topologia, instalação de cópia de dado, processo de busca,

além de uma breve descrição das arquiteturas existentes na literatura.

2.1.1 Redes de Sobreposição

A interligação de computadores em uma rede pode ser descrita de forma física ou lógica. A rede física consiste nas conexões reais existentes entre os computadores, sejam elas por cabos ou sem fio. Essa classificação corresponde ao desenho físico da rede. A rede lógica, ou rede de sobreposição, corresponde à interligação de alguns desses computadores da rede física por meio de conexões (*enlaces virtuais*) estabelecidas pelas aplicações em execução nesses computadores. Esse subconjunto forma uma nova rede, dita sobreposta, sobre a rede física original [54]. Nessa classificação, apenas os computadores pertencentes à rede sobreposta é que conseguem se comunicar diretamente, definindo dessa maneira o fluxo de dados na rede física. Esses computadores participantes possuem papéis semelhantes, ora funcionando como clientes, ora como servidores, fazendo com que não exista uma hierarquia ou diferenciação entre eles.

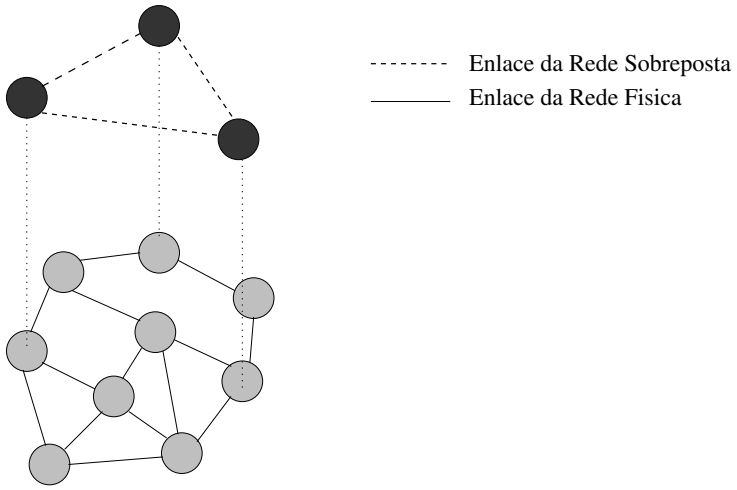


Figura 2.1: Esquema de uma rede de sobreposição

2.1.2 Busca e Replicação

De maneira genérica, o propósito de uma aplicação P2P é o compartilhamento de algum tipo de recurso entre os participantes da rede. Exemplos de recursos podem ser arquivos, ciclos de processamento, espaço de armazenamento, etc. Para requisitar um determinado dado de interesse, um nó deve executar algum **algoritmo de busca**. Isso é normalmente implementado por propagação de mensagens de busca para os demais participantes. Ao receber uma mensagem de busca, um nó, caso possua o item procurado, normalmente envia uma mensagem de resposta diretamente ao nó solicitante. A semântica de uma mensagem de busca varia de acordo com a aplicação e também de acordo com as facilidades oferecidas pela arquitetura da rede. Em redes estruturadas, por exemplo, a semântica de

uma busca é bastante restrita e é capaz de expressar diretamente apenas casamentos exatos de identificadores de objetos.

Um conjunto de réplicas corresponde a cópias de um dado que estão espalhadas pela rede. Essas cópias podem ser apenas apontadores (referências ao dado que indicam a existência desse objeto em algum nó da rede), ou até o próprio dado por completo. **Replicação** de itens é uma estratégia utilizada por desenvolvedores de aplicações para aumentar o desempenho de algoritmos de busca [20, 25, 71] com relação à taxa de sucesso e latência de resposta. Apesar de ser uma estratégia comumente utilizada, ela é geralmente criticada pela alta carga de dados inserida na rede.

2.2 Arquiteturas P2P

A definição da arquitetura do sistema P2P permite a classificação da aplicação em centralizada, descentralizada não estruturada e descentralizada estruturada. As estratégias de compartilhamento de recursos, buscas e envio de mensagens variam também de acordo com a arquitetura definida. Nesta seção, são apresentadas as características de cada arquitetura.

2.2.1 Centralizada

Um sistema com arquitetura centralizada utiliza um servidor central (ou *cluster* de servidores) para auxiliar no atendimento às requisições de serviços e nas tarefas de manutenção da infra-estrutura da rede. Apesar da semelhança com a arquitetura cliente-servidor, o funcionamento desta arquitetura é bastante diferente. Nesse modelo, um novo usuário se conecta à rede utilizando uma lista de servidores previamente disponível em algum sítio da Internet. O servidor é utilizado para receber a listagem de todo o conteúdo que um novo usuário deseja compartilhar e para disponibilizar a localização dos conteúdos de todos os outros usuários que já fazem parte da rede.

Quando um nó deseja localizar um determinado objeto, ele envia uma requisição diretamente ao servidor central. Como o servidor possui informações sobre o conteúdo de todos os nós da rede, ele responde ao solicitante se o objeto procurado existe ou não na rede. Caso o objeto exista na rede, o servidor envia uma lista com os endereços IPs de todos os nós que compartilham o objeto procurado. O nó requisitante tem liberdade para escolher um dos nós da lista e estabelecer conexão com esse para requisitar o objeto. A transferência ocorre de forma direta entre o nó requisitante e o nó escolhido da lista, sem a participação do servidor central.

A Figura 2.2 demonstra o funcionamento de uma arquitetura centralizada. Neste exemplo, existem dois nós ativos na rede e um servidor centralizado. Um novo nó deseja se conectar a essa rede e realizar o compartilhamento de recursos. No **passo** 1 o novo nó faz a requisição de conexão ao servidor. O servidor o cadastra e recebe a lista de todo o conteúdo que ele deseja compartilhar. O **passo** 2 ilustra o momento em que o novo nó realiza uma requisição por um objeto qualquer. O servidor, de posse do cadastro do

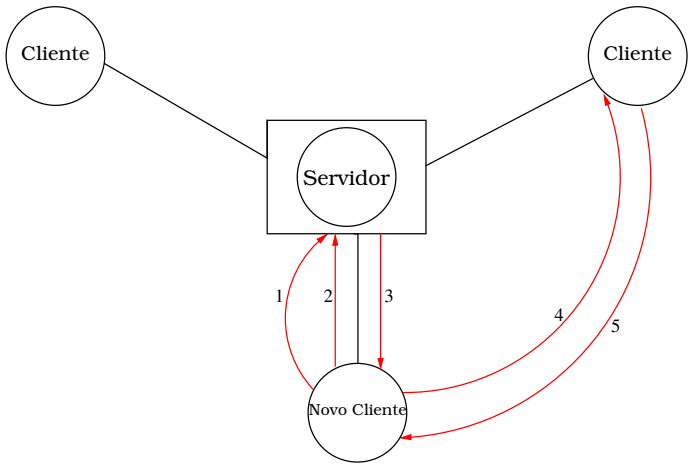


Figura 2.2: Arquitetura centralizada

conteúdo de todos os participantes conectados, verifica quais possuem o objeto requisitado e responde ao nó requisitante com seus endereços IPs (**passo** 3). O nó requisitante, sabendo quais nós da rede possuem o objeto, escolhe, com base em certas características (como por exemplo, latência, tamanho do arquivo, qualidade do arquivo, etc) de qual nó será feita a transferência. Após a escolha, ele faz contato direto com o nó escolhido e obtém o conteúdo desejado (**passos** 4 **e** 5).

Uma grande vantagem dessa arquitetura é a garantia de que, se o objeto desejado existe em um dos nós conectados à rede, esse será encontrado com uma única consulta ao servidor. Por outro lado, uma desvantagem evidente é a dependência total do sistema do servidor central. Caso haja alguma falha nesse servidor, o sistema deixa de operar. Um exemplo desse tipo de falha foi o próprio Napster [9] que, após batalha judicial pelos direitos autorais dos conteúdos compartilhados, deixou de operar seu servidor e ocasionou o fim do serviço da maneira inicialmente projetada.

Como no modelo cliente-servidor, a utilização de servidores centralizados limita a escalabilidade do sistema. Para fornecer um serviço de compartilhamento com um número muito grande de usuários, são necessários vários servidores organizados em um cluster para que não haja comprometimento e sobrecarga dos servidores disponíveis. Isso faz com que a manutenção de um serviço dessa natureza seja custosa e pouco viável financeiramente [75]. Além disso, para alguns autores, uma rede que utiliza uma arquitetura centralizada não pode ser considerada P2P, já que não é totalmente distribuída (apenas as transferências de objetos são realizadas de forma descentralizada).

2.2.2 Descentralizada Não Estruturada

Para alguns autores, os sistemas P2P devem realizar o compartilhamento de conteúdo de forma distribuída sem que exista um servidor central de armazenamento. Por isso recebem a classificação de sistemas não centralizados (ou descentralizados). Além disso as topologias definidas pelas conexões estabelecidas pelos participantes não seguem critérios

pré-definidos de conexões ou roteamentos específicos na rede e por isso esses sistemas são também denominados não estruturados. O modelo descentralizado não estruturado apresentou grande evolução nos últimos anos [71, 25, 30, 41] com o desenvolvimento de inúmeras estratégias eficientes, porém os primeiros algoritmos a realizar buscas em redes P2P desse modelo foram baseados em um modelo de inundação da rede. No modelo de inundação o controle do mecanismo de pesquisa é baseado em um parâmetro denominado tempo de vida (TTL). Quando um usuário deseja buscar por um determinado objeto na rede, ele envia a requisição para todos os seus nós vizinhos (conectados diretamente a ele) com um valor de TTL inicial embutido na mensagem. Caso um de seus vizinhos possua o objeto requisitado (ou uma referência) é enviada uma resposta para o nó requisitante com o endereço IP do nó que possui o objeto. Se o vizinho não possuir o objeto requisitado, o parâmetro TTL tem seu valor decrementado. Caso esse valor retorne zero, a busca é dada por finalizada naquele nó. Já se o TTL for maior que zero, a requisição é repassada para todos os vizinhos do nó corrente (com excessão do vizinho que o repassou a mensagem).

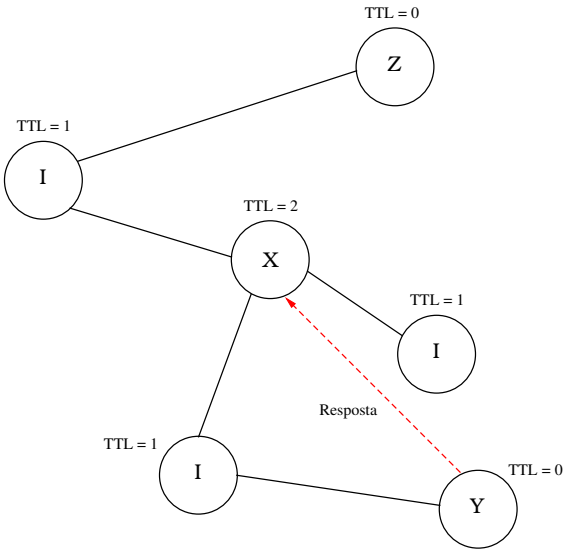


Figura 2.3: Arquitetura descentralizada não estruturada baseada em inundação

Na Figura 2.3 tem-se um exemplo de uma busca em uma rede P2P baseada em inundação. Inicialmente, o nó identificado como **X** realiza uma requisição por um determinado objeto na rede. Essa requisição possui um TTL inicial com valor dois e é enviada para todos os vizinhos imediatos identificados por **I**. Cada vizinho **I** decrementa o valor do TTL da mensagem (o valor atinge um), realiza uma busca local pelo objeto mas sem obter sucesso. Como o TTL da mensagem ainda não é zero, todos esses vizinhos **I** encaminham essa mensagem para os seus respectivos vizinhos (com excessão do vizinho que o repassou a mensagem). Nesse terceiro e último passo um nó identificado por **Y** possui o objeto requisitado pelo nó **X**. Por fim, com o endereço IP da mensagem de busca, o nó **Y** entra em contato com o nó **X** para realizar a transferência do objeto requisitado. Uma situação comum seria o nó **Y** não possuir o objeto requisitado. Neste caso, o TTL da mensagem atingiria o valor zero, a mensagem não continuaria a propagação e a busca se daria por encerrada com resultado de insucesso para o objeto requisitado. A grande

contribuição desse modelo é o seu pioneirismo na utilização de uma arquitetura totalmente descentralizada. Tal descentralização tornou o processo mais ágil e independente de grandes servidores e hardware de alto custo. Um exemplo prático dessa arquitetura é a rede Gnutella [4].

Apesar de simples e robusta, alguns dos problemas dessa arquitetura são a grande largura de banda consumida e a quantidade de processamento efetuada pelos participantes da rede. Isso porque, para fazer requisições aos participantes, a replicação da mensagem de busca é feita em todas as direções, o que ocasiona possíveis sobrecargas a participantes com menor capacidade de processamento. Outro aspecto negativo dessa técnica é a repetição de mensagens de busca para um mesmo participante. O modelo de inundação é considerado "cego" e não oferece garantia de cobertura da rede. Nesse mecanismo, um nó que apresenta um número elevado de conexões tende a receber uma maior quantidade de mensagens de seus vizinhos e, por consequência, muitas dessas mensagens podem ser repetidas. Isso faz com que a maioria dos sistemas desenvolvidos que utilizam inundação adotem mecanismos de detecção de duplicação para evitar que mensagens de buscas se repitam em um mesmo nó.

A falta de garantia de sucesso na busca por um objeto é outra desvantagem desse modelo. O TTL especificado pode ser insuficiente ou considerado grande (em função da rede possuir um valor alto de diâmetro) a ponto de tornar inviável o tempo de resposta de uma solicitação [23]. Assim, um determinado objeto pode estar presente em algum nó conectado à rede, mas por motivos de configuração inadequada do TTL da mensagem de busca, esse objeto pode nem mesmo ser encontrado (TTL insuficiente), ou o tempo para encontrá-lo é, na prática, inaceitável (TTL com valor alto).

Uma forma de reduzir a quantidade de mensagens propagadas no modelo de inundação se dá pela utilização de caminhadas aleatórias. A técnica consiste na propagação da mensagem de busca para um nó escolhido aleatoriamente a cada passo, até que o objeto seja encontrado. Para que haja uma propagação mais rápida da mensagem pelos nós da rede, alguns sistemas P2P utilizam variações da abordagem e, ao invés de realizar a propagação por um único nó aleatório, a mensagem é distribuída para um número k de vizinhos. Essas variantes serão descritas na Seção 2.4.

2.2.3 Descentralizada Estruturada

A estrutura de uma arquitetura descentralizada está relacionada com o controle de criação e manutenção da topologia. Diferentemente da arquitetura não estruturada, em que nós e dados estão distribuídos aleatoriamente pela rede, nas redes P2P estruturadas a topologia é construída de forma determinística. O procedimento determinístico mais conhecido é baseado na distribuição por tabelas de dispersão distribuídas (DHT). No modelo, tanto os dados quanto os nós recebem identificadores (únicos) de um espaço de identificadores possível. O grande desafio é construir um esquema eficiente e determinístico para mapear unicamente a chave de um item para o identificador do nó responsável pela chave.

Em uma estratégia de DHT, nós e dados possuem identificadores obtidos por uma função de dispersão. O processo de busca se inicia em um nó requisitante e a cada passo

(salto) a busca é direcionada (roteada) para o nó com identificador mais próximo do identificador do dado pesquisado. Assim que o objeto é encontrado na rede (nó responsável pelo dado é encontrado), é realizada então a transferência de conteúdo entre o nó que o solicitou e o nó que possui o conteúdo procurado, sem intervenções de nós intermediários. A estratégia de mapeamento de identificadores entre conteúdo e nó, aliada ao processo de roteamento de busca permite que o modelo estruturado apresente garantias de acesso a um dado caso esse esteja disponível na rede e ainda oferece garantias teóricas para o número máximo de nós que uma mensagem de busca percorre na rede. Para que isso seja possível, é necessário que a estrutura da rede seja mantida e as referências para conexões na rede estejam devidamente atualizadas. Estudos relacionados às características de entrada e saída de nós em uma rede P2P (*churn*) [61, 68] mostram que o tempo de permanência médio de um indivíduo em uma rede P2P é bastante pequeno. Para uma rede em grande escala, isso corresponde a um elevado número de entradas e saídas de nós em um curto período de tempo. O fator complicador é que a maioria das redes DHTs necessita de uma grande quantidade de tempo para o restabelecimento de conexões desfeitas repentinamente (sem prévio aviso), e por isso, nós com conexões limitadas (linha discada, por exemplo) se tornam alvos fáceis de sobrecarga de mensagens destinadas à manutenção da topologia da rede.

2.3 Algoritmos de Buscas em Redes Estruturadas

Para toda e qualquer busca realizada no modelo estruturado, uma função de dispersão é utilizada para gerar a identificação do nó na rede que é o responsável por armazenar o dado desejado (ou ao menos estar mais próximo que o nó atual). Uma questão importante então consiste na necessidade em saber especificar o nome dos dados procurados de forma exata. Qualquer distorção na especificação do nome do dado procurado pode levar a busca a ser direcionada para um nó que não tem relação alguma com o conteúdo. Por isso, esse processo de busca não é indicado para localização de objetos em que a identificação é apenas parcialmente conhecida. As diferentes estruturas e esquemas de roteamento definem os variados sistemas estruturados de uma rede P2P. Nas subseções seguintes, serão apresentados três exemplos de redes estruturadas que foram bem discutidos na literatura: Chord [65], Pastry [59] e CAN [57].

2.3.1 Chord

Chord [65] é um dos precursores dos sistemas estruturados para redes P2P. A estrutura proposta se baseia em um anel para criação (manutenção) da topologia e disposição tanto dos nós quanto do conteúdo por um espaço de identificadores circular de m *bits*. Cada participante é responsável por uma porção desse espaço e a distribuição é realizada de maneira uniforme pelas 2^m possíveis posições mediante a utilização de uma função de dispersão (SHA-1 [18]). Essa função é aplicada ao endereço *ip* do nó, para dispor esse em uma determinada posição do anel, e à descrição do objeto, para atribuir a referência de um objeto a um nó do anel. Um participante fica responsável por armazenar referências para

objetos que possuem identificadores maiores que o identificador do nó imediatamente a sua esquerda no círculo e menores que o seu próprio identificador. Um exemplo, seria o caso de existir nós X,Y e Z dispostos nas posições 5, 40 e 120 de um círculo, respectivamente. Nesse caso o nó Y seria o responsável pelos objetos com identificadores que estão no intervalo $[5 - 40)$, o nó Z pelos identificadores $[40 - 120)$ e, por fim, o nó X com os identificadores de $[120 - 5)$ o que completa todo o espaço de endereçamento do anel.

Ao ingressar na rede, um nó se torna responsável por armazenar as referências para os objetos do espaço de identificadores coberto por ele. Referências que estão em posse de outros nós mais antigos na rede, porém no espaço de identificadores de objetos do novo nó, devem ser transferidas para o novo nó ingressante na rede. Da mesma forma, a saída de um nó da rede também exige uma nova organização das referências, visto que as referências desse nó que saiu devem ser redistribuídas para algum nó que permaceneu na rede e tem identificador adequado para manter a organização do modelo proposto.

Em Chord, todo participante que deseja solicitar conexão à rede, primeiro obtém um identificador e requisita a algum *proxy* para estabelecer conexão com algum outro participante já inserido na rede (anel). A requisição é direcionada então por esse *proxy* para o par com identificador mais próximo ao identificador do novo integrante. Ao estabelecer conexão com seus vizinhos no anel, esse novo integrante recebe então as referências que devem ser mantidas por ele. Essas referências possuem identificadores que estão, pela definição do funcionamento do sistema, no limite do intervalo dos identificadores que devem ser mantidos pelo novo integrante (identificador). As buscas seguem um modelo de *"hot-potato"* e são sempre direcionadas a encontrar o par responsável por armazenar um determinado objeto com identificador sob sua responsabilidade. De maneira estratégica, as buscas são repassadas adiante para vizinhos com maiores identificadores na estrutura do anel até que o objeto seja encontrado ou o limite do intervalo da busca tenha se excedido.

A Figura 2.4 mostra, como exemplo, um modelo de Chord com $m = 3$ bits e um total de $2^3 = 8$ identificadores disponíveis para disposição de nós e objetos. Do espaço de identificadores, apenas os identificadores um, quatro e seis são utilizados na rede. Caso o nó um deseje publicar um novo objeto de nome "objeto1" primeiramente é feita a aplicação da função de dispersão *dispersão* ("*objeto*1") para se obter o identificador do nó que será o responsável por manter essa referência na rede. Supondo que o identificador obtido foi cinco, então o nó cinco seria o responsável por armazenar o objeto, mas como esse nó não está presente ativamente no espaço de endereços disponíveis de zero a sete no momento, o seu sucessor (seis) fica responsável por armazenar a referência e insere uma tupla com o identificador do objeto (cinco) e o endereço IP do nó um em sua tabela de informação do conteúdo disponibilizado na rede.

Em uma busca em Chord, o nó solicitante envia ao seu nó sucessor uma requisição contendo o seu endereço IP e o identificador do objeto procurado. Caso o sucessor não possua o objeto com o identificador especificado então o pedido é repassado para o seu sucessor até que o objeto seja encontrado no espaço de endereços ou a busca tenha alcançado o limite máximo do intervalo de pesquisa. Quando essa referência é encontrada em algum nó, as informações de onde encontrar o objeto são enviadas para o nó solicitante diretamente. Esse nó solicitante, ao receber as informações de qual computador na rede possui esse objeto, utiliza o endereço IP da mensagem e faz a requisição pelo dado.

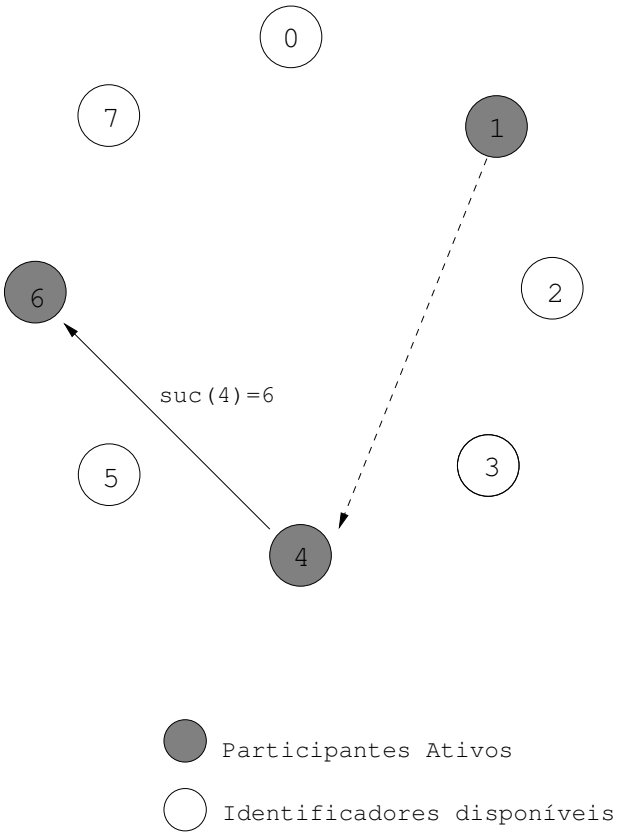


Figura 2.4: Exemplo do modelo de Chord

Um nó em Chord utiliza uma estrutura denominada tabela *finger* para manter informações adicionais de outros nós pertencentes a rede e que não estão conectados a ele. Com isso vários nós podem ser consultados ao mesmo tempo (já que vários nós estão acessíveis pela consulta a seus identificadores na tabela) e a busca ser direcionada mais rapidamente para um nó que possui identificador mais próximo do identificador do objeto procurado. Esses nós da tabela finger estão localizados em posições estratégicas do círculo (posições *hash(id)/2, hash(id)/4, hash(id)/8*, etc) e permitem que um nó faça uma busca mais eficiente pelo anel. Cada entrada da tabela consiste dos campos de ínicio e o endereço IP do sucessor, definidos da seguinte forma:

$$inicio[i] = \left(k + 2^i\right) \mod 2^m$$

$$sucessor[i] = sucessor(inicio[i])$$

em que:

k: é o identificador do nó

i: é a entrada da tabela

m: número de *bits* para endereço

A tabela *finger* do nó 1 para a configuração da Figura 2.4 é dada pela Tabela 2.1.

Tabela 2.1: Tabela *finger* para nó 1

Início	**Sucessor**
(1 + 1)%8=2	4
(1 + 2)%8=3	4
(1 + 4)%8=5	6

A busca se inicia normalmente com a aplicação da função de dispersão ao nome do objeto a ser requisitado. Se o identificador obtido pela função de dispersão estiver entre k (identificador do nó requisitante) e o sucessor de k, então o nó que pode possuir a informação sobre o objeto é o sucessor de k. Caso contrário, verifica-se na tabela *finger* a entrada, no campo início, que possui o nó com identificador mais próximo do identificador do objeto requisitado.

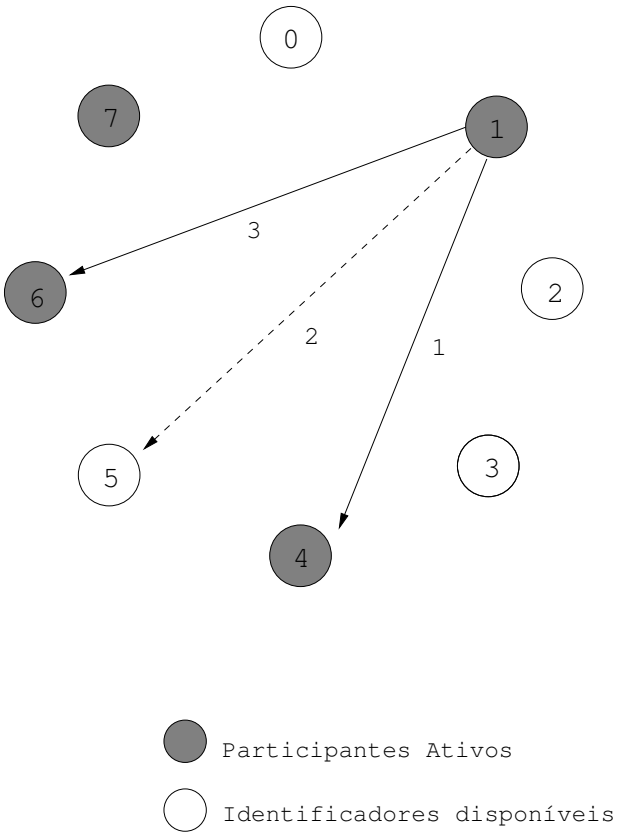


Figura 2.5: Funcionamento da tabela *finger*

A Figura 2.5 mostra o funcionamento da tabela *finger* em Chord. Nesse exemplo, os nós com identificadores um, quatro, seis e sete fazem parte da rede e o nó um procura

por um objeto que tenha o valor de dispersão igual a sete. Como esse objeto possui identificador que não está localizado entre um e o sucessor de um (suc(1)=4), faz-se necessário a consulta à tabela. Na tabela *finger*, verifica-se que o predecessor mais próximo (análise do campo início) do nó com identificador sete é o nó com *id* cinco. Porém, o nó cinco não está ativo na rede e essa requisição é repassada para o nó sucessor (seis) que usará a mesma estratégia de busca inicial do nó um. O *id* sete está entre seis e o $suc(6) = 7$ e a resposta é encontrada (sete). Assim o nó seis informa ao nó um que o objeto pode ser obtido no nó sete. O número médio de passos em uma pesquisa é $\log n$ [65]. Outra importante característica a se destacar é que Chord também pode se utilizar de caches de informações a respeito de nós que já fizeram alguma requisição por objetos no sistema. Informações dos nós que já utilizaram o nó como caminho para solução de consultas são armazenadas e podem ser utilizadas como futuros atalhos para consultas na rede.

2.3.2 CAN

O modelo *DHT* de CAN [57] é baseado em um espaço de coordenadas virtuais de n dimensões. Os participantes estão distribuídos em regiões virtuais bem definidas e que não possuem relação com o sistema físico de coordenadas. Todo objeto é mapeado neste espaço virtual por uma função de dispersão aplicada à sua descrição. Esse mapeamento fornece uma coordenada (d-tupla) do espaço virtual (d-dimensional) onde esse documento estará virtualmente presente. Como exemplo, no caso em que CAN utiliza um espaço 2-dimensional, o objeto descrito por "Cantor: Musica" poderia ser mapeado para a coordenada virtual (2,5) e um outro objeto descrito por "Cantora: Musica" ser projetado nas coordenadas (12,14). O mesmo processo é feito para distribuir os participantes pela região virtual do sistema. Enquanto um participante com IP X pode ser mapeado para as coordenadas (2,5) outro com IP Y pode ser posicionado em (9,17). Com essa técnica de distribuição de conteúdo e nós pelas regiões virtuais a política básica de CAN é resumida em subdividir recursivamente o espaço de identificadores d-dimensionais, armazenando cada objeto no nó em posse daquela região do espaço ("zona") onde o identificador do objeto é mapeado.

O nó responsável pela zona onde se encontra a coordenada gerada pela função de dispersão aplicada a um documento é quem armazena a referência desse documento. Esse nó utiliza o par (identificador do objeto,valor) para controlar o conteúdo armazenado na sua porção do espaço. O espaço virtual em posse de um nó qualquer é dinamicamente redistribuído a cada entrada ou saída de um nó da rede. A entrada de um novo nó no sistema faz então que esse divida uma região com um nó que já estava presente na rede e dessa forma, as referências de objetos presentes na antiga região virtual são redistribuídas de acordo com a nova divisão realizada desse espaço. Já na saída de um nó, as referências de objetos presentes na rede em posse do nó de saída são movidas para o nó que agora ficará responsável pelo espaço deixado vago. Isso mantém coerente a fração desse espaço destinada a cada nó e também a correta distribuição das referências no sistema.

É evidente que os custos associados com a redistribuição de dados dentro do sistema podem se tornar substanciais, principalmente nos casos em que a quantidade de dados no sistema se torna cada vez maior com o passar do tempo e se existirem muitos nós na rede

com *links* de baixa capacidade de transmissão (recepção). Outro fator de grande importância é que os nós devem ser capazes de armazenar fisicamente (espaço) as referências de objetos que são delegadas à sua zona em função da disposição imposta pela função de dispersão.

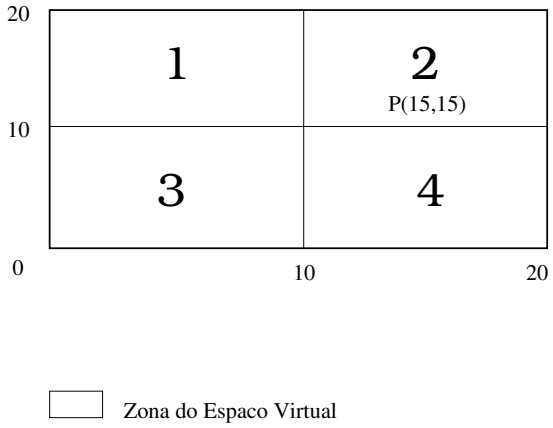


Figura 2.6: Exemplo do modelo de CAN com quatro nós

A Figura 2.6 mostra um exemplo de distribuição do espaço virtual entre quatro nós que fazem parte de uma rede CAN com duas dimensões. Os processos de busca e publicação de conteúdo neste modelo são direcionados unicamente pelas coordenadas virtuais do espaço. Para se publicar um arquivo com o nome de "objeto1" a partir do nó um, por exemplo, é feito o cálculo da função *dispersão*("objeto1") para obter uma coordenada P do espaço virtual onde o objeto deve ser publicado. O nó que possui a responsabilidade sobre a zona onde P está contido é que fica responsável por armazenar a referência para o objeto. Assim, caso o "objeto1" receba como valor de dispersão a coordenada P(15,15), então, para este exemplo, o nó dois seria o responsável por armazenar essa referência.

Uma requisição feita por um usuário a algum nó de CAN é roteada de um nó para outro que possua coordenada mais próxima à coordenada do objeto procurado (obtida pela função de dispersão aplicada à descrição). Cada nó possui uma tabela de roteamento contendo o endereço IP e as coordenadas dos seus vizinhos imediatos (vizinhos próximos segundo métrica de distância). No início do processo, a função de dispersão é utilizada para se obter a coordenada do espaço onde esse objeto deve ser procurado. Essa coordenada é de responsabilidade de algum nó da rede e a requisição é direcionada para ele utilizando as tabelas de roteamento dos vários nós intermediários. Essa requisição transita sempre em busca do vizinho que possui as coordenadas que sejam mais próximas do nó destino e o processo é repetido até que a mensagem chegue ao nó responsável pela zona de destino.

Segundo [57], para um espaço com d dimensões dividido em n zonas iguais, são necessários em média $(\frac{d}{4})\ n^{\frac{1}{d}}$ saltos para se completar uma busca, com cada nó sendo responsável por armazenar informações de $2d$ vizinhos. Como exemplo, em um espaço 2-dimensional, os caminhos terão tamanho proporcional a raiz quadrada de N. Apesar do modelo se caracterizar pela simplicidade no processo de roteamento das mensagens que seguem as

coordenadas virtuais em direção aos nós mais próximos do nó destino existem questões críticas de seu funcionamento. O usuário CAN deve ficar sempre atento a refazer atualização das suas referências a objetos em função dos nós que podem sair repentinamente da rede (seja por desconexão ou até quebra), sem aviso prévio. Além disso, outra questão bastante discutida nesse modelo é com relação a consistência do sistema mediante grandes alterações em sua estrutura (elevado índice de entrada e saída de nós).

2.3.3 Pastry

O modelo de Pastry apresentado em [59] é semelhante ao modelo adotado por Chord. Sua estrutura também se baseia em um círculo de nós com identificadores de 128 bits atribuídos aleatoriamente. A proximidade numérica de dois identificadores indica proximidade dos nós na rede de sobreposição. Cada objeto também tem associado a si um identificador com o mesmo número de bits. Além disso, um objeto com chave k está associado ao nó cujo identificador é lexicograficamente mais próximo de k.

O roteamento de mensagens pela rede de sobreposição se dá pela consulta às tabelas que armazenam referências a nós considerados próximos por alguma métrica estipulada. O conjunto de tabelas inclui a tabela de "nós folhas", tabela de "vizinhança" e a "tabela de roteamento". As informações contidas em cada tabela são resultado de descoberta dos próprios nós e também da troca de informações com outros nós da rede. A tabela de "nós folhas" é formada por L nós vizinhos lexicograficamente mais próximos. No conjunto, $\frac{|L|}{2}$ são nós com identificadores menores numericamente, chamados predecessores e $\frac{|L|}{2}$ são nós com identificadores maiores numericamente, chamados sucessores. Além dessa tabela, cada nó possui ainda uma tabela de "roteamento". A tabela é constituída de $\log_{2^b} n$ linhas e $2^b - 1$ colunas (em que b é um parâmetro do sistema) e possui vários blocos de endereços de nós com prefixos em comum a ele. Os blocos podem variar de acordo com os diferentes tamanhos de prefixos como pode ser visto na tabela de roteamento da Figura 2.7. Já a tabela de "vizinhança" representa os M nós com identificadores mais próximos de um participante em função de alguma métrica de roteamento.

O processo de busca em Pastry é semelhante aos outros modelos estruturados. A mensagem é propagada por nós distribuídos pelo círculo e a cada passo um nó que possui identificador mais próximo numericamente do identificador do objeto é que recebe a mensagem de busca. Em um primeiro momento obtém-se o identificador do objeto com uma função de dispersão. Em seguida verifica-se se o identificador desse objeto tem valor compreendido na faixa de endereços suportada (valor intermediário entre menor e maior identificadores da tabela) pela tabela de nós "folhas". Caso o identificador esteja na faixa de endereços, a mensagem de busca é transmitida ao nó cuja distância numérica (lexicográfica) em relação ao nó corrente é a menor possível dentre todos os nós pertencentes ao conjunto de nós folhas. Caso o identificador do objeto não seja compreendido pelo conjunto de nós folhas, a tabela de roteamento deve ser consultada. Na tabela de roteamento, busca-se o nó que possua um prefixo junto ao objeto que seja maior que o prefixo do nó corrente. Se existe esse nó na tabela de roteamento, a mensagem é repassada a ele. Caso contrário, a mensagem é enviada para o nó que possua a menor distância numérica, consultando as três tabelas simultaneamente. Se o nó resultante dessa consulta possui

prefixo em comum menor que o nó corrente, o procedimento de busca é finalizado (convergência do processo). Segundo [59], são utilizados em média $\log n$ passos para efetuar o roteamento de uma mensagem em uma rede com n nós ativos.

Um simples exemplo de seu funcionamento com base nas informações da Figura 2.7 seria a busca em um nó 10233102 por um objeto cujo identificador obtido pela função de dispersão é 10231425. O nó consulta a sua tabela de nós "folhas" e verifica que o identificador do objeto não está compreendido na faixa de endereços dos nós da tabela. É então feita uma consulta à tabela de roteamento em busca de um nó com identificador que tenha prefixo comum maior que o nó corrente (que possui prefixo 1023 em comum). Assim, ao consultar a tabela é encontrado o nó 10231000, cujo prefixo em comum corresponde a 10231. Dessa forma, como encontrou-se esse nó na tabela de roteamento, a mensagem de busca é direcionada a ele. O nó 10231000 em posse da mensagem de busca também fará consultas as suas tabelas para tentar encontrar um nó que esteja mais próximo do destino. Essa busca se encerra até que não seja possível propagar a mensagem de busca para um nó mais próximo do destino ou o objeto seja encontrado.

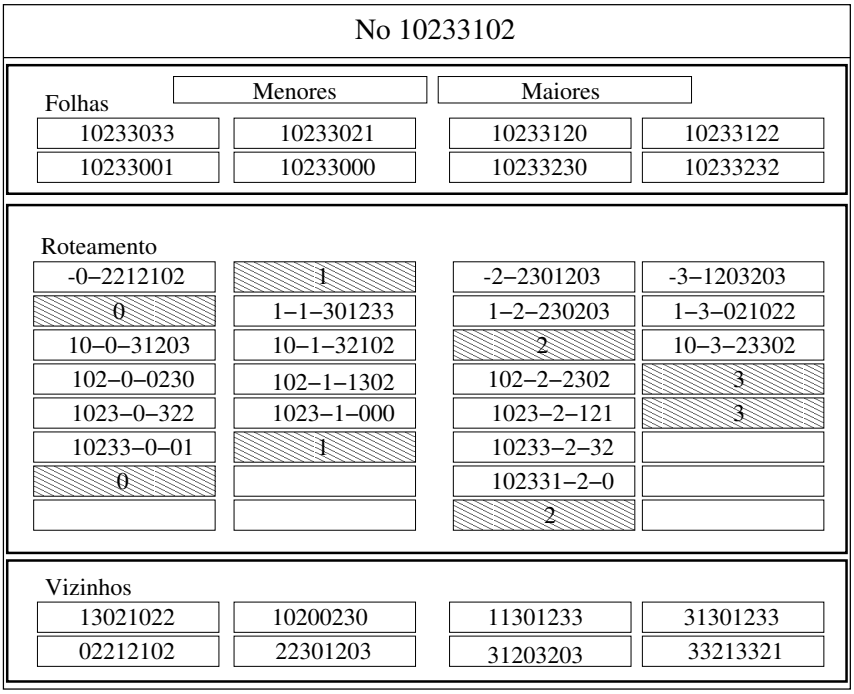


Figura 2.7: Tabelas de rotas armazenadas por um nó em Pastry

2.3.4 Symphony

Uma abordagem semelhante a Chord é apresentada em [46]. Neste modelo de rede estruturada, são adaptadas características de redes não estruturadas com a adoção de conceitos relacionados ao fenômeno de mundo pequeno (*Small World*). Em Symphony os nós estão dispostos em um anel no intervalo $[0, 1)$, em uma posição aleatória obtida no momento da conexão à rede. Além dos nós predecessor e sucessor como vizinhos imediatos, Symphony

realiza a ligação de outras conexões aleatórias por meio de uma função de distribuição de probabilidade harmônica. As réplicas são instaladas em nós escolhidos a partir de funções de dispersão distribuídas e as buscas são roteadas para o nó que possua a menor distância com relação ao identificador do objeto. Ou seja, existe um casamento exato envolvendo nome do objeto e posicionamento do nó e cada nó é responsável por uma parte do subintervalo (o tamanho do subintervalo está diretamente relacionado com as entradas de nós com indentificadores próximos ao identificador do nó em questão) do círculo. O protocolo de estimativa do tamanho da rede é baseado no arco pertencente ao nó e nos arcos pertencentes aos vizinhos dele. Pelo tamanho dos arcos é possível fazer a estimativa de quantos nós estão distribuídos pela rede. Outra novidade é a existência de links bidirecionais na rede, permitindo que as mensagens sejam roteadas em dois sentidos. Além disso, a técnica se utiliza de um esquema de *lookahead* de distância um, armazenando em cada nó, as informações dos nós sucessor, predecessor e de longo alcance dos nós imediatos a ele. Com essas informações, ele consegue direcionar a mensagem para o nó que fornece maior progressão à mensagem de busca.

2.4 Algoritmos de Buscas em Redes Não Estruturadas

Nesta seção serão apresentados alguns trabalhos diretamente relacionados ao princípio de funcionamento do protocolo SplitQuest que será proposto no Capítulo 3. Dois principais protocolos foram utilizados como fundamentação e serão mais bem detalhados nesta seção: Caminhadas Aleatórias Modificadas [25] e BubbleStorm [71].

2.4.1 Caminhadas Aleatórias Modificadas

Ferreira *et al.* [25] apresentam a primeira solução escalável (sublinear) a oferecer garantias probabilísticas em operações de busca em redes não estruturadas. O protocolo se baseia em uma variante do paradoxo do aniversário e utiliza estratégia de replicação próativa. O processo de publicação primeiramente encontra, em uma rede de tamanho n, k pares escolhidos de maneira uniforme e aleatória. Essa escolha uniforme é realizada a partir de uma caminhada aleatória modificada, em que diferentes valores probabilísticos são atribuídos as ramificações dessa caminhada, utilizando o algoritmo Metropolis-Hastings (descrito mais detalhadamente na Seção 3.4.1). Essa modificação visa garantir que todos os nós, independentemente do número de conexões, possuam a mesma probabilidade de serem selecionados na rede. Mais especificamente um par p realiza uma caminhada aleatória de tamanho $c \log n + \delta\sqrt{n}$ em que c é uma constante parametrizada pelo sistema. Na mensagem de caminhada aleatória o par p responsável pela publicação inclui as informações de um objeto O e solicita aos pares pertencentes a caminhada aleatória para que instalem as referências de O. Os primeiros $c \log n$ pares apenas repassam a mensagem de caminhada aleatória, enquanto que os $\delta\sqrt{n}$ pares restantes instalam localmente a referência para o conteúdo do par p. Neste caso, δ corresponde a um parâmetro do sistema que realiza ajustes na quantidade de pares que receberão a cópia da referência. Um maior

valor de δ implica em um número maior de pares que receberão a cópia da referência. Consequentemente, isso proporciona uma redução no tempo de respostas das mensagens de buscas e ainda uma aumento na taxa de sucesso com relação a esse objeto.

Um aspecto interessante do processo de publicação é que a instalação de referências não é feita igualmente para todos os objetos. Existe um controle, feito por um algoritmo probabilístico, no processo de instalação dessas referências para objetos. O controle visa instalar um número menor de referências para objetos ditos populares, já que a probabilidade em se encontrar um objeto deste tipo em algum par da rede é grande, e um número maior de referências para objetos raros. Esse controle é estabelecido durante o processo de estabelecimento de conexão de um par. Primeiramente, um par ao entrar na rede, realiza uma busca pelo objeto que deseja publicar. Caso a busca não tenha sucesso, o par entende esse insucesso como pouca popularidade do objeto e instala a referência em algum par da rede com probabilidade um. Se a busca retornar com sucesso e o par responsável pela referência estiver à distância (número de saltos feitos pela caminhada aleatória) l de p, então p deve instalar as referências com probabilidade $\frac{l}{c \log n + \delta\sqrt{n}}$. O principal resultado desse algoritmo consiste na redução significativa no número de referências para objetos populares com redução não significativa na probabilidade em se encontrar objetos raros na rede.

O processo de busca por objetos na rede também é realizado pelo uso de caminhada aleatória. A caminhada aleatória possui a informação da busca (objeto procurado) e um tempo de vida (TTL) configurado para $c \log n + \delta\sqrt{n}$, o mesmo valor do processo de instalação de referência. Os pares presentes durante o processo de propagação da caminhada aleatória processam a mensagem de busca localmente e analisam seu conteúdo e o conteúdo de referências instaladas por outros pares. Caso o par possua o conteúdo (ou sua referência) ele cessa a propagação da caminhada aleatória e fornece a resposta ao par origem da mensagem de busca. Já se o par não possuir a resposta, o valor do TTL é decrementado, e caso o valor não retorne zero, a mensagem é repassada para um próximo vizinho, escolhido aleatoriamente, para dar prosseguimento a caminhada aleatória. O valor zero significa o fim da caminhada aleatória.

O trabalho apresenta ainda uma estratégia de manutenção das referências instaladas. Como as redes P2P são caracterizadas pelo grande número de pares que entram e saem da rede em um curto período de tempo, a estratégia de manutenção das referências instaladas diz respeito à instalação adicional de referências visando suprir a saída de alguns pares da rede (seja essa saída programada ou devido à falha). Para uma fração f de pares que saem da rede e m o conjunto de pares que instalaram a referência para um determinado objeto, são instaladas, após um período determinado de tempo, $f * |m|$ novas referências em pares selecionados uniforme e aleatoriamente. Com essas estratégias os autores afirmam que qualquer objeto na rede pode ser encontrado, com alta probabilidade em $O(\sqrt{n \ln n})$ saltos. O principal problema da técnica proposta é o tempo de resposta das mensagens de buscas em função da latência de propagação das caminhadas aleatórias.

2.4.2 BubbleStorm

O trabalho desenvolvido em [71] se baseia na idéia proposta em [25] e apresenta BubbleStorm, uma técnica resiliente para buscas em redes P2P não estruturadas. BubbleStorm apresenta também sua fundamentação no paradoxo do aniversário para fornecer garantias probabilísticas de sucesso. O princípio básico de funcionamento consiste na existência de uma "bolha" (BubbleCast) que representa as mensagens de instalação de referências e outra "bolha" que representa as mensagens de buscas inseridas na rede. A intersecção das duas bolhas retrata a situação de uma busca ser respondida com sucesso. Para um par q (mensagem de busca) e d (dado) distribuído de forma uniforme e independente, isso ocorre com probabilidade menor que $e^{\frac{-qd}{n}}$ [71].

O funcionamento de BubbleStorm está condicionado à topologia de um multigrafo (self-loops e múltiplas conexões são permitidas). A topologia é estrategicamente modelada para que os pares não sofram variação no número de conexões (grau) durante a permanência na rede e a estrutura apresente um número menor de ocorrências de instabilidade. Para isso, os pares são posicionados em um círculo, estabelecendo um número par de conexões com dois vizinhos imediatos. Pares que possuam grau par maior que dois devem realizar o procedimento de entrada múltiplas vezes. A entrada de um novo par na rede faz com que seus vizinhos imediatos atualizem a conexão no círculo e estabeleçam uma nova conexão com o novo vizinho, desfazendo conexões com vizinhos antigos. A saída de um par na rede também faz com que seus vizinhos imediatos atualizem prontamente as conexões para que não exista a possibilidade da topologia circular ser desfeita. Para manter a topologia consistente todas as operações de entrada e saída da rede são serializadas. O processo de entrada na rede é realizado pela consulta a um *proxy* que realiza a propagação de uma mensagem de caminhada aleatória de tamanho logarítmico até enfim encontrar o local na topologia em que o novo nó estabelecerá a conexão.

BubbleCast representa a replicação das mensagens de busca e instalação de referências em pares na rede. Essa estratégia se resume a uma combinação de caminhada aleatória com inundação. Da caminhada aleatória a principal característica é a replicação controlada dessas mensagens e consequentemente do número de pares contactados durante o processo. Do modelo de inundação a principal característica herdada é a baixa latência de propagação, permitindo que um grande número de pares sejam alcançados a cada passo. Como parâmetros de entrada do processo de replicação são fornecidos w e s em que w corresponde ao número de réplicas a serem instaladas e s corresponde ao fator de replicação do sistema. O valor de s indica para quantos pares uma mensagem deve ser propagada em um único passo. Uma mensagem de busca ou publicação inicia a propagação com um valor w calculado pelo sistema. A cada par atingido, a mensagem tem o valor de w decrementado e o par que recebeu processa a réplica localmente, armazenando ou fazendo a busca. Se o valor atingido por w não for zero, a busca/publicação é distribuída entre os s vizinhos escolhidos aleatoriamente. A Figura 2.8 representa o funcionamento da propagação de uma mensagem em BubbleStorm. Nela o valor de w inicial é 11 e $s = 2$.

O valor estimado de w nas mensagens de busca ou publicação de conteúdo depende do fator de certeza c obtido a partir da equação de certeza proposta pelos autores:

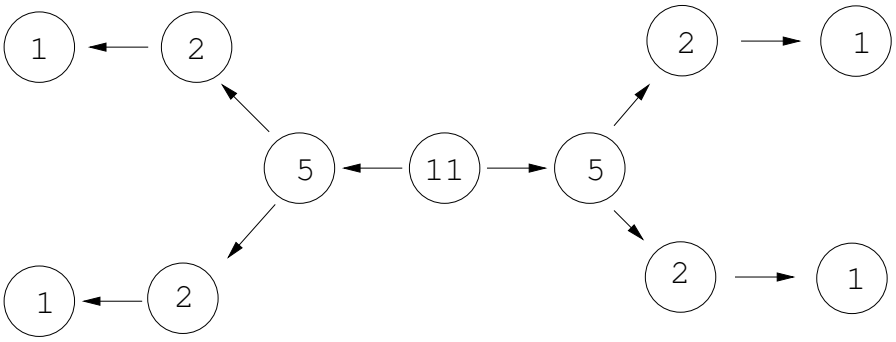


Figura 2.8: Esquema de propagação de mensagens utilizado por BubbleStorm

$$c = \sqrt{-\ln(1-r)} \tag{2.1}$$

Na especificação, r corresponde ao parâmetro de confiança. A partir da Equação 2.1 que representa o parâmetro de certeza é possível se obter

$$1 - r = e^{c^2} = e^{\frac{-qd}{n}} \implies qd = c^2 n \tag{2.2}$$

de modo que é estabelecida uma relação produto do número de réplicas/mensagens de buscas e o parâmetro de confiança desejado. A observação mais importante dessa relação é que quanto maior for o parâmetro de certeza para encontrar um objeto, maior a necessidade de inserção de mensagens nessa rede. Segundo os autores, o valor de w para mensagens de buscas e instalação de conteúdo respectivamente, podem então, ser obtidos pela equação

$$q = c\sqrt{n\frac{R_d}{R_q}} \tag{2.3}$$

$$d = c\sqrt{n\frac{R_q}{R_d}} \tag{2.4}$$

em que R_q e R_d correspondem, respectivamente, as taxas em *bytes/segundo* que as mensagens de buscas e dados são inseridas na rede. O parâmetro n ainda pode ser substituído por um limiar T, definido em função da caminhada aleatória ser influenciada pelo grau dos nós presentes na rede. Assim, é definido um limiar T como:

$$T = \frac{D_1^2}{D_2 - 2D_1} \tag{2.5}$$

em que $D_i = \sum_{v \epsilon V} deg(v)^i$, v corresponde a um par da rede, *deg(v)* corresponde ao grau do par.

O protocolo demonstra bons resultados com relação ao tempo de resposta, taxa de sucesso e robustez para diversas topologias e situações de *churn* em uma rede P2P. Porém,

as simulações desenvolvidas neste trabalho demonstram que esse protocolo, para distribuições com grande variação no grau dos nós, apresenta baixo desempenho em função do grande número de mensagens inseridas na rede.

2.4.3 Outros Algoritmos de Buscas em Redes Não Estruturadas

Com o objetivo de explorar estratégias auxiliares às buscas em redes não estruturadas, Cohen e Shenker [20] apresentam um estudo teórico de algoritmos de replicação de dados. O trabalho avalia duas estratégias de replicação, uniforme e proporcional, e mostra que ambas estratégias possuem o mesmo desempenho médio para buscas bem sucedidas. A estratégia uniforme realiza a replicação de todos os dados uniformemente, independente da popularidade do objeto. A replicação proporcional leva em consideração a proporção de requisições feitas a um dado. É apresentada também uma estratégia ótima de replicação que se situa entre as estratégias uniforme e proporcional. Essa estratégia é proporcional à raiz quadrada das taxas de frequência de acesso aos objetos. Este trabalho, apesar de fornecer um resultado teórico de grande importância, não apresenta um resultado prático de relevância, já que a realização de uma estratégia ótima de replicação obriga que os nós tenham conhecimento das frequências de acesso dos objetos, o que na realidade, é inviável (visto que as frequências de acesso dos objetos não são conhecidas em uma aplicação P2P real). Além disso, o objetivo das estratégias de replicação não é prover garantias de sucesso e sim minimizar o tempo médio de busca.

Em [27] é proposto um método de busca que combina caminhadas aleatórias com inundações localizadas. A cada passo da caminhada, uma das ramificações da mensagem de busca inunda pares vizinhos com mensagens. Outra funcionalidade apresentada é a utilização de replicação de conteúdo para vizinhos. Todos os nós da rede fazem uma replicação do índice de seu conteúdo para vizinhos imediatos (conectados diretamente). Essas funcionalidades permitem a redução no número de mensagens disseminadas em busca de conteúdo além de atingir um número maior de nós na rede. Isso proporciona um aumento na velocidade de propagação das mensagens de buscas porém, apesar dos resultados satisfatórios das simulações, esse mecanismo ainda é questionado com relação à disseminação desordenada de mensagens de buscas de conteúdo (sem direção), em função da falta de garantia de cobertura ideal da rede.

Em [33] são apresentadas duas variações do protocolo Gnutella. Na primeira e mais simples, BFS modificada, as mensagens de buscas são propagadas apenas para um subconjunto de nós. Esse subconjunto é escolhido aleatoriamente e sua cardinalidade é um parâmetro de configuração do sistema. A segunda variação corresponde ao algoritmo denominado Mecanismo de Busca Inteligente. Esse algoritmo é baseado em técnicas de recuperação de informação e um nó que deseja realizar uma busca por algum objeto sabe, com base em fatos passados, qual dos seus vizinhos tem maior capacidade em responder à tal requisição. Isso só é possível porque cada nó armazena dados estatísticos (perfis) de buscas já realizadas no passado (bem como quais nós foram responsáveis por respondê-las) para que em uma nova oportunidade, novas pesquisas sejam direcionadas a esses nós. O mecanismo utiliza técnica de ranqueamento de similaridade entre a busca atual e buscas já realizadas (e respondidas com sucesso) para criar a relação de nós aptos a receber a

nova mensagem. Resultados experimentais demonstram redução significativa do número de mensagens comparado ao modelo de inundação. Em contrapartida, o trabalho não apresenta detalhes de como a saída de nós e remoção de objetos da rede afeta a taxa de sucesso do mecanismo.

Outro estudo com características de replicação pró-ativa é apresentado em [55]. A técnica proposta é baseada em caminhada aleatória e recebe o nome de SCRW (*Supernode-Constrained Random Walk*). A idéia básica de seu funcionamento consiste em evitar a propagação de mensagens de buscas por nós comuns e enviar mensagens de buscas por supernós (nós com maior capacidade de armazenamento/processamento e maiores responsabilidades na rede). É utilizado também um mecanismo de replicação denominado de "dois saltos". Nesse mecanismo, os nós são capazes de replicar seus índices de documentos por vizinhos que se encontram em até dois saltos de distância (vizinhos dos vizinhos). Essa replicação pode ser total (replicação por todos os vizinhos dos vizinhos , replicação da raiz quadrada (feita pelos supernós e para raiz quadrada da quantidade de vizinhos supernós escolhidos aleatoriamente) e replicação constante (feita pelos supernós e para uma quantidade constante de vizinhos supernós). As simulações mostram que a técnica obtem desempenho duzentas vezes superior à caminhada aleatória comum e seiscentas vezes superior ao modelo de inundação com relação a taxa de sucesso das mensagens de buscas.

Uma abordagem baseada em interesses de conteúdo que também busca otimizar o modelo de inundação Gnutella é apresentada em [64]. Nessa abordagem, todo nó pertencente à rede possui uma tabela com um conjunto de nós considerados atalhos para as buscas efetuadas. Um nó ao se conectar à rede possui a tabela de atalhos inicialmente vazia. Qualquer mensagem de busca submetida ao sistema e que tenha sucesso trás consigo o identificador do nó responsável por compartilhar o recurso procurado. Do conjunto de nós disponíveis para fornecer esses recursos, um nó é escolhido de maneira aleatória para integrar a lista de nós atalhos do nó requisitante. Assim, buscas futuras por outros recursos utilizarão, como atalho, os nós que já responderam a alguma requisição em um passado recente. Primeiramente, o nó requisitante contata a sua tabela de nós atalhos e envia a mensagem de busca a esses. Caso a busca não tenha sucesso, a técnica de inundação é utilizada.

Essa tabela de nós atalhos é mantida de acordo com dados estatísticos de utilização. Caso um nó atalho possua participação em larga escala na resolução das buscas efetuadas na rede, esse nó é mantido na tabela por um tempo maior que aquele que apresenta uma baixa contribuição. O ponto mais relevante desse trabalho consiste em evitar a grande utilização de largura de banda da técnica de inundação, estabelecendo requisições com nós de interesses em comum. Apesar dessas vantagens o trabalho foi desenvolvido sobre condições ótimas da rede (sem transiência, por exemplo), o que não permite que se possa obter uma visão ampla da técnica sobre condições adversas. Outro fator comprometedor é que a otimização utilizando atalhos pode nem sempre ser utilizada. Isso porque, caso os resultados das buscas utilizando a tabela de atalhos não sejam satisfatórios, a técnica de inundação é utilizada como último recurso.

Mais recentemente, uma abordagem híbrida de um sistema P2P pode ser observada em [43]. A topologia do modelo consiste na junção de idéias do modelo estruturado

(DHT) com idéias do modelo não estruturado. Nós considerados estáveis (participantes que permanecem por grandes períodos de tempo na rede e normalmente possuem maior poder de computação) formam a topologia estruturada da rede e são responsáveis por armazenar informações de todos os nós participantes dessa. Qualquer nó ao se conectar à rede, deve registrar informações de seu estado (grau, tamanho de espaço disponível para armazenamento, número de arquivos compartilhados, preferência de arquivos) em algum desses nós pertecentes à topologia estruturada (subsistema). Uma busca realizada por algum nó dessa rede é primeiramente repassada a um nó desse subsistema. Esse nó é o responsável por escolher, com base em informações globais armazenadas nele, por qual nó será submetida a busca (instalação). A grande vantagem desse modelo é o aumento da probabilidade de resolução de mensagens de buscas em função da consulta prévia a nós formadores do subsistema, porém o trabalho não mostra de maneira explícita como deve ser feita essa escolha dos nós formadores do núcleo do sistema e não apresenta detalhes que evidenciem as características dos experimentos.

Com um objetivo pouco diferente do proposto neste trabalho, Raiciu *et al.* apresenta ROAR [56], um algoritmo para distribuição de conteúdo utilizando clusters de máquinas servidoras. Em ROAR as máquinas servidoras participantes estão agrupadas em clusters e possuem identificadores distribuídos no intervalo $[0, 1]$. Para fornecer maiores possibilidades no processo de busca, o conteúdo de uma máquina servidora é replicada por todas as máquinas localizadas em um mesmo cluster (intervalo). Para descobrir quais máquinas devem receber essa replicação, é consultado um servidor de *back end* responsável por armazenar informações das principais máquinas (identificação dos clusters) que deverão receber a réplica. Para agilizar o processo de pesquisa, uma mensagem de busca direcionada ao sistema é repassada para todos os clusters existentes na rede. A mensagem de busca ao chegar a um servidor de *front end* é repassada, segundo informações armazenadas localmente, a nós localizados em diferentes clusters. Todos os clusters recebem essa mensagem de busca e uma das máquinas é responsável por processar a requisição. O resultado da mensagem de busca é retornado para o servidor de *front end* novamente, e esse servidor é o responsável por fornecer o resultado final da consulta com base em todas as respostas obtidas. O artigo apresenta foco em técnicas de reconfiguração da topologia em tempo de execução para permitir que o sistema continue funcionando enquanto são realizadas modificações nos servidores e clusters da arquitetura. Não é foco do trabalho a avaliação de métricas de desempenho do processo de busca como número de mensagens inseridas na rede e tempo de resposta.

Ioannidis *et al.* [30] apresenta uma modificação do modelo Gnutella. A idéia do trabalho é encurtar a propagação das buscas por objetos que não existem na rede. Essa idéia se justifica no modelo Gnutella, mais especificamente no fato das buscas pelo método inundação serem baseadas no parâmetro de tempo de vida (TTL). Como se sabe, esse parâmetro pode promover inserção de grande quantidade de mensagens na rede (TTL alto), bem como limitar a busca (TTL baixo). Caso um objeto não exista na rede, a mensagem de busca neste modelo continua sendo difundida pela rede até que o TTL da mensagem de busca expire. Outras abordagens, como caminhadas aleatórias e suas variações, apresentam comportamento semelhante. Assim, a principal característica da técnica proposta é tentar diminuir o raio de atuação de mensagens de buscas por objetos inexistentes na rede de modo a não influenciar nos resultados das buscas por objetos que estão disponí-

veis na rede. A técnica proposta é baseada nas caminhadas aleatórias tradicionais porém com uma simples modificação. Qualquer nó que tenha feito uma busca por um objeto e a busca não tenha obtido sucesso, registra localmente a ocorrência da ausência do objeto na rede. Quando um nó qualquer realizar uma busca subsequente por aquele mesmo objeto e a mensagem de busca atingir o nó que possua o registro da ausência do objeto, o nó requisitante será informado de que o objeto não existe na rede e a busca terminará. Dessa forma, buscas por objetos que provavelmente não estão na rede possuem o raio de propagação reduzido.

2.5 Fenômeno de *Small-World*

O vínculo e a maneira com que dois indivíduos se relacionam em uma sociedade são motivos de inúmeras pesquisas em diversas áreas da ciência [62, 66, 37, 36, 35]. Desde a Psicologia até a Computação, essa relação entre os indivíduos é representada de modo formal e caracteriza o que se denomina de **redes sociais**. Uma rede social busca representar, de maneira fiel, as conexões existentes entre os indivíduos de uma sociedade e apresentar as características íntrisecas à rede como por exemplo a topologia, distância relativa entre dois indivíduos, grau de integração da sociedade, etc.

As características de uma topologia podem ser observadas pela representação computacional da sociedade. Essa representação leva em consideração, única e exclusivamente, os relacionamentos existentes entre os indivíduos. Caso um indivíduo tenha algum tipo de relação com outro, essa relação é representada por uma conexão computacional entre esses. Para realizar essa representação, são estabelecidas notações em que indivíduos são representados formalmente como vértices e as conexões existentes são representadas por arestas. Essa representação formal é interessante não só pelo aspecto representativo, mas também por permitir que novas frentes de pesquisas sejam desenvolvidas a exemplo da construção de algoritmos e protocolos que exploram as características das diversas topologias existentes.

Existem inúmeros estudos em áreas da Biologia, Redes Neurais e das próprias redes P2P que utilizam topologias conhecidas e bem especificadas como pode ser visto em [24, 71, 74, 14]. Um tipo bastante comum é a **topologia regular**. Nesse tipo de topologia todos os nós possuem um mesmo número de conexões (grau) e, em alguns casos, essas conexões são bem definidas (a formação da vizinhança é estabelecida por uma função matemática). A principal característica desse tipo de topologia é o alto índice de "agrupamento" dos participantes. Esse agrupamento consiste na formação de pequenos conjuntos de nós que são agrupados por um número elevado de conexões entre os seus formadores. Em termos práticos, isso se deve ao fato de que vizinhos de um nó qualquer dessa rede possuem alta probabilidade de serem vizinhos entre si.

Outra topologia também conhecida é a **topologia aleatória**. Nesta, os graus dos vértices são variáveis (por opção ou em função da capacidade do nó) e as conexões são realizadas por procedimentos aleatorizados. A principal consequência de uma topologia com essas características é o pequeno diâmetro da rede. Isso se justifica pela existência de caminhos curtos que interligam quaisquer dois nós e que são independentes da distância

geográfica existente na topologia. Alguns autores acreditam que essa representação é irrealista, visto que a técnica não representa o princípio de agrupamento de vizinhança observado em redes sociais reais, como constatado na topologia regular.

Apesar de serem utilizadas na representação de vários aspectos do mundo real, as topologias regular e aleatória não são consideradas representações ideais para uma rede social moderna. Isso porque, em ambas as topologias, a sociedade é caracterizada por propriedades extremas. Na topologia regular todos os vértices possuem o mesmo grau; na topologia totalmente aleatória as conexões não apresentam qualquer tipo de vínculo entre os nós da rede. Em muitos casos, faz-se necessária a utilização de uma topologia intermediária, em que não exista uma topologia bem definida, nem uma topologia totalmente aleatória. Essa topologia intermediária deve absorver tanto a característica de agrupamento da topologia regular, bem como a característica de pequeno diâmetro da rede apresentada pela topologia aleatória. São essas características de topologia que definem a existência do chamado "fenômeno de mundo pequeno" (*small-world*).

O fenômeno de *small-world* tem suas raízes na década de sessenta com um experimento de Stanley Milgram [60]. Em seu experimento Milgram tinha o objetivo de diagnosticar a distância de um indivíduo a outro com base nos estudos da rede de relacionamento a que perteciam. A experiência consistia em enviar cartas a partir de um indivíduo em uma cidade A, direcionadas para um indivíduo localizado em uma cidade B, geograficamente distante. Essas cartas possuíam poucas informações sobre o indivíduo destino e consistiam no primeiro nome, profissão e cidade. No modelo proposto, os indivíduos eram representados por vértices e as arestas representavam o fato de que dois indivíduos se conhecessem. Cada carta era repassada para o vizinho que o nó em posse da carta considerasse mais próximo do indivíduo destino, ou seja, um indivíduo que seria um potencial conhecedor do destinatário. A resposta do experimento consistia em quantificar o número de participantes que receberia a carta durante o percurso da origem para o destino.

Quando uma carta finalmente chegava ao seu destino, era possível verificar e analisar todo o caminho percorrido e os indivíduos presentes na trajetória de entrega. Milgram chegou à conclusão de que o tamanho médio do caminho existente entre dois indivíduos na rede é de aproximadamente seis[1]. Outra característica observada no experimento foi a eficiência no procedimento de transição da mensagem da origem para o destino, já que com apenas algumas informações locais (conhecimento pessoal do indivíduo em posse da carta) a maioria das cartas puderam ser remetidas ao indivíduo em seu destino.

Essas características são as responsáveis por conceituar as topologias das redes sociais existentes em "topologias de mundo pequeno". Isso porque, apesar de geograficamente distantes, dois indivíduos estão conectados por poucos passos uns dos outros. Essas características podem ser justificadas em função da existência de caminhos de longo alcance. Esses caminhos significam que os indivíduos possuem conhecimento da existência de outros indivíduos em regiões geograficamente distantes na rede. Isso faz com que exista uma redução da influência do fator geográfico na transição de mensagens pelas redes sociais, já que qualquer informação pode trafegar de maneira rápida por uma vasta extensão dessas redes.

[1]Também conhecido como "seis graus de separação".

Os primeiros estudos mais embasados para construção de topologias com características do fenômeno de *small-world* foram realizados por Watts e Strogatz em [74]. O algoritmo apresentado é dividido em duas fases. A primeira se caracteriza pela criação de uma topologia regular baseada em anel, de maneira que todos os nós possuam grau k. As k conexões estabelecidas no anel são realizadas com os $\frac{k}{2}$ vizinhos no sentido horário e com $\frac{k}{2}$ vizinhos no sentido anti-horário. Essas conexões representam o conhecimento local desses nós e também o relacionamento existente entre indivíduos que, por exemplo, moram perto, trabalham juntos ou que tenham preferências similares. Na segunda fase do processo é criada uma rede sobreposta ao anel regular. Essa sobreposição é obtida pela transformação de conexões já existentes em conexões aleatórias de longo alcance. Essas conexões de longo alcance tornam possível a existência de pequenos caminhos entre nós distantes geograficamente na rede. O processo de aleatorização das arestas pode ser conferido no Algoritmo 1.

Algoritmo 1: Algoritmo de aleatorização das arestas no modelo de Watts e Strogatz

Entrada: Anel regular com n vértices, p (probabilidade em se fazer a mudança da conexão existente) e k (número de conexões por nó).
Saída: Rede com características de *small-world*.

```
1 para cada aresta i = 1...k/2 faça
2     para cada nó v = 1...n faça
3         Com probabilidade p, substitua a aresta que conecta (v) a (v + i), por uma
          aresta de (v) para (u), em que u é selecionado de forma aleatória e uniforme
          dentre todos os nós da rede
      fim
  fim
```

É fácil perceber que a variação do valor de p, no intervalo $[0, 1]$, corresponde a significativas mudanças na topologia da rede. Para valores de p próximos a zero, a topologia se aproxima de uma topologia regular, já que as arestas criadas no anel inicial não apresentam grandes modificações. Já para valores de p próximos a um, a topologia se torna próxima à aleatória, em função da grande probabilidade de troca dos extremos das arestas.

A topologia baseada em *small-world* é caracterizada por apresentar comportamento intermediário à topologia regular e à topologia aleatória. Uma rede nessa topologia tem então como característica os agrupamentos de nós e a existência de caminhos curtos interligando dois nós quaisquer. Isso justifica as duas fases do algoritmo de Watts e Strogatz. A primeira fase é a responsável por fornecer a característica de agrupamento das redes regulares enquanto que a segunda é a responsável por estabelecer conexões aleatórias para encurtar os caminhos entre dois nós da rede. A partir dessa abordagem Watts define uma topologia como *small-world* utilizando dois índices quantificadores:

Índice de Clusterização $C(p)$: é responsável por quantificar a intensidade do agrupamento dos nós da vizinhança de um participante. Assim, para cada nó i, é mensurado o índice de agrupamento das conexões existentes entre os seus vizinhos.

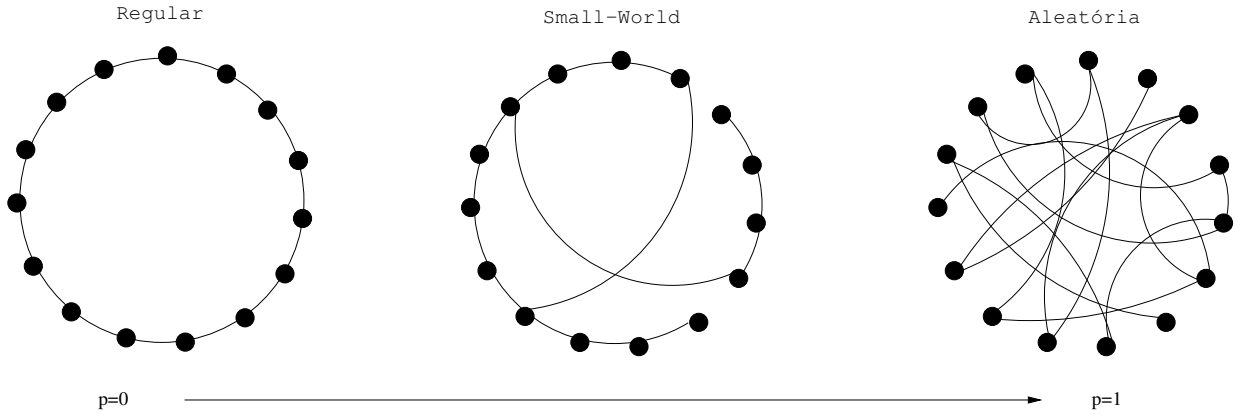


Figura 2.9: Mudanças de topologias com variação do valor de p no algoritmo de Watts e Strogatz

Seja o grafo $G(V, E)$, o representante da rede em questão, em que V corresponde ao número de vértices do grafo e E corresponde ao número de arestas. Defina o conjunto de vizinhos de um vértice v_i, como V_i=$\{v_j\}$, tal que $v_j \in V$ e $e_{ij} \in E$. O grau do vértice (número de vizinhos) pode ser definido como $g_i = |V_i|$. Logo, o número máximo de conexões existentes entre os vizinhos de um vértice v_i pode ser definido como $M_{v_i} = \frac{g_{v_i}(g_{v_i}-1)}{2}$ considerando o grafo não direcionado. Considere agora que $E_{v_i} = \{e_{jk}\}$, em que v_j e $v_k \in V$ e $e_{jk} \in E$, sejam as arestas existentes entre os vizinhos de v_i. O coeficiente de clusterização de v_i é então definido como:

$$C_{v_i} = \frac{|E_{v_i}|}{M_{v_i}} \tag{2.6}$$

Para o grafo G, o coeficiente de clusterização é definido por:

$$C = \sum_{i=1}^{n} \frac{C_{v_i}}{|V|} \tag{2.7}$$

Exemplo:

Na Figura 2.10, o nó A possui 3 vizinhos (B, C e D). O número máximo de conexões que pode existir entre esses vizinhos é então calculado como:

$$M_{v_A} = \frac{g_{v_A}(g_{v_A}-1)}{2} = \frac{3(3-1)}{2} = 3$$

Como o número de arestas existentes nos vizinhos é apenas um (conexão existente entre os nós B e C), então:

$$C_{v_A} = \frac{|E_{v_A}|}{M_{v_A}} = \frac{1}{3}$$

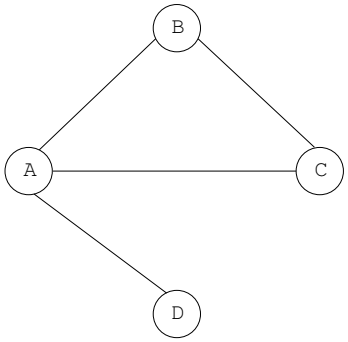


Figura 2.10: Exemplo de clusterização de vizinhança

Para calcular o índice de clusterização para **todo** o grafo, deve-se também calcular o índice de clusterização para os vértices B e C:

$$C_{v_B} = \frac{|E_{v_B}|}{M_{v_B}} = \frac{1}{1} = 1$$

$$C_{v_C} = \frac{|E_{v_C}|}{M_{v_C}} = \frac{1}{1} = 1$$

Assim:

$$C = \frac{C_{v_A}+C_{v_B}+C_{v_C}}{3}$$

$$C = \frac{\frac{1}{3}+1+1}{3} = \frac{\frac{7}{3}}{3} = \frac{7}{9}$$

Índice de Caracterização do Tamanho do Caminho $L(p)$: é responsável por quantificar o tamanho médio do caminho existente entre dois nós da rede. Em um grafo $G(V, E)$, com vértices V e arestas não direcionadas E, para todo vértice $v \in V$ deve-se calcular um caminho para todos vértices $v' \in V$ com $v' \neq v$, tal que $\sum e_{ij}$ é mínimo para o conjunto de arestas interligando um nó i a um nó j no caminho entre o vértice v e v'.

Com os conceitos de $C(p)$ e $L(p)$ definidos, é possível verificar se uma topologia possui traços de *small-world* ou não. Para isso, Watts define, de maneira empírica, os passos do Algoritmo 2 em alto nível como descrito abaixo:

Algoritmo 2: Algoritmo verificador de topologia *small-world* de Watts e Strogatz

Entrada: Rede com n vértices.
Saída: Resultado da verificação se a rede possui características de *small-world* ou não.

1 Calcule os valores de $C(p)$ e $L(p)$ para a topologia em análise

2 Utilize o mesmo número de vértices da topologia em análise, o mesmo número de conexões por vértice e construa uma topologia que seja totalmente aleatória ($p = 1$)

3 Calcule os valores de $C(1)$ e $L(1)$

4 Compare os valores de C e L de ambas as topologias. Para se caracterizar como uma topologia de *small-world*, deve-se obter os seguintes resultados:

a. Os valores de $L(p)$ e $L(1)$ devem ser próximos

b. O valor de $C(p)$ deve ser superior ao valor de $C(1)$

A condição **4a** se justifica pela utilização de arestas aleatórias de longo alcance que possibilitam a existência de pequenos caminhos entre dois nós quaisquer da rede. Já a condição **4b** é herdada da topologia regular, cujo índice de clusterização é bem superior ao da topologia aleatória, em função do alto grau de agrupamento entre os nós integrantes desse tipo de topologia.

Esse estudo oferece um embasamento empírico com a utilização de grafos de base de dados reais. Esses grafos correspondem a diferentes segmentos da ciência e apresentam as mais variadas situações. Foram utilizados grafos representantes do relacionamento de atores de cinema que participaram de um mesmo filme, da distribuição de energia nos Estados Unidos e ainda das redes neurais do vírus nematode *C. Elegans*. Todos esses grafos são caracterizados pela topologia de mundo pequeno e apresentam os índices de clusterização (C) e de tamanho do caminho médio (L) representados na Tabela 2.2.

Tabela 2.2: Representação dos índices de clusterização (C) e tamanhos de caminhos (L) das base de dados reais (top) e da topologia aleatória (ale) [74]

	L_{top}	L_{ale}	C_{top}	C_{ale}
Atores de filmes	3.65	2.99	0.79	0.00027
Estação Elétrica	18.7	12.4	0.080	0.005
Nemátode	2.65	2.25	0.28	0.05

Em [34], Kleinberg apresenta uma nova técnica para criação de topologias com características de mundo pequeno. Essa técnica se baseia no modelo de Watts e Strogatz [74] e propõe a criação da rede em um *grid* de duas dimensões na primeira fase do algoritmo. O *grid* tem tamanho definido $n \times n$, em que n é o número de nós da rede. Um nó presente neste *grid* é identificado por um par de coordenadas $\{(i, j) : i \in \{1, 2, \ldots, n\}$ e

$j \in \{1, 2, \ldots, n\}\}$. Neste *grid* a distância entre dois nós quaisquer x e y com coordenadas respectivamente (i, j) e (k, l) é definida por:

$$d((i,j)(k,l)) = |k - i| + |l - j| \tag{2.8}$$

As primeiras conexões são estabelecidas com vizinhos que estão à distância p (≥ 1) e são chamadas de conexões locais. Além dessas, devem ser feitas l conexões de longo alcance para inserção dos caminhos curtos entre dois nós da rede. Para isso, Kleinberg define a probabilidade da existência de uma conexão entre dois vértices de forma proporcional a $d(u, v)^{-r}$. O valor do expoente r indica qual a influência da distância no estabelecimento de conexão entre dois pares. Quanto maior o valor de r, menor probabilidade no estabelecimento de uma conexão. Todos os pares de nós da rede utilizam essa idéia, seguindo o Algoritmo 3.

Algoritmo 3: Algoritmo para criação de modelo *small-world* de Kleinberg

Entrada: n (número de nós), k (número máximo de conexões por nó), l (número de arestas de longo alcance) e r (expoente de *Power-Law*).
Saída: Criação da topologia de *small-world* de Kleinberg.

1 Construa um *grid* com n nós e k possíveis conexões

2 Para cada nó $v = 1, ..., n$ repita l vezes

a. Adicione uma aresta de v para u, em que u é escolhido de forma aleatória e com probabilidade proporcional a $d(u, v)^{-r}$

O passo **2** do algoritmo mostra claramente que a decisão de estabelecer conexões está vinculada à distância entre dois nós. Além disso, o parâmetro r serve como fator modificador de topologia, já que quantifica a intensidade de agrupamento existente entre os nós dessa rede. Para um valor de r igual a zero, ou seja, criar a topologia independente da distância no *grid*, o modelo se assemelha ao modelo de Watts e Strogatz com probabilidade p igual a 1. Para valores maiores de r, as conexões ficam concentradas em vizinhos da mesma região o que consequentemente pode reduzir o conhecimento global dos nós e aumentar o caminho existente entre dois nós quaisquer da rede. A Figura 2.11 apresenta um exemplo de um *grid* 6x6 criado na primeira fase do algoritmo. A Figura 2.11 b apresenta o conjunto de conexões de um nó do *grid* após o processo de aleatorização.

Em [66], é apresentada uma nova caracterização das redes de sobreposição estabelecidas pelas aplicações P2P. O trabalho desenvolvido é responsável por registrar o conjunto de hosts pertecentes à rede e a topologia de rede formada por esses hosts.

O trabalho desenvolvido utiliza um *crawler* denominado de Cruiser, responsável por averiguar quais os hosts presentes na rede e as conexões estabelecidas por esses. O *crawler* utiliza técnicas de processamento paralelo de consultas e apresenta como resultado prático inúmeros *snapshots* da rede Gnutella [4]. A principal contribuição desse trabalho é a conclusão obtida pelos autores de que os sistemas P2P modernos continuam a apresentar

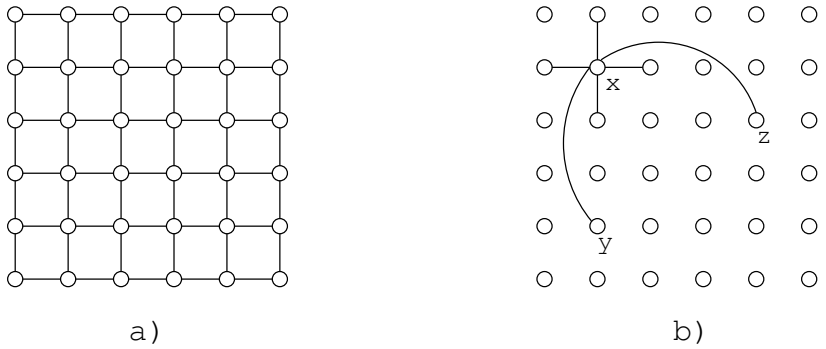


Figura 2.11: Topologia do modelo de Kleinberg

características de topologia de mundo pequeno, com alto índice de clusterização e pequeno diâmetro de rede.

Shudong *et al.* mostra em [62] razões para considerar a topologia da Internet como *small-world*. A primeira delas seria a variação existente no grau dos nós da rede. Essa afirmação se justifica pelo fato de que dois nós interligados em uma rede P2P apresentam grande probabilidade de possuir um vizinho em comum, principalmente se este nó possuir um número grande de conexões. A variação dos graus forma indiretamente as conexões denominadas "triangulares", torna as relações mais intensas entre nós e como consequência o aumento do índice de clusterização da rede.

A segunda razão apresentada seria a preferência dos nós por conexões locais (conexões com vizinhos próximos na rede de *overlay*) na rede. Segundo [62], se um nó x possui conexões com participantes y e w, então os nós y e w, com alta probabilidade, estarão conectados entre si. Essas características são representadas na Figura 2.12. Em a) é representada a variação no grau dos vértices. Em b) é representada a preferência dos nós por conexões locais.

A maior contribuição do trabalho [62] corresponde a apresentação de dois algoritmos para a construção de topologias com características de *small-world* e variação no grau dos nós. Esta última característica é uma novidade em comparação com as abordagens de Watts [74] e Kleinberg [34] e considerada mais realista pela literatura. No primeiro algoritmo os graus dos vértices são obtidos por uma distribuição bem conhecida com características *power-law*. Na fase inicial de estabelecimento de conexões, para cada vértice são criadas pd conexões locais, em que p corresponde a probabilidade de criação da conexão, enquanto d corresponde ao número máximo de conexões possível para um vértice. As arestas restantes são criadas de forma aleatória. O algoritmo a seguir mostra tal idéia.

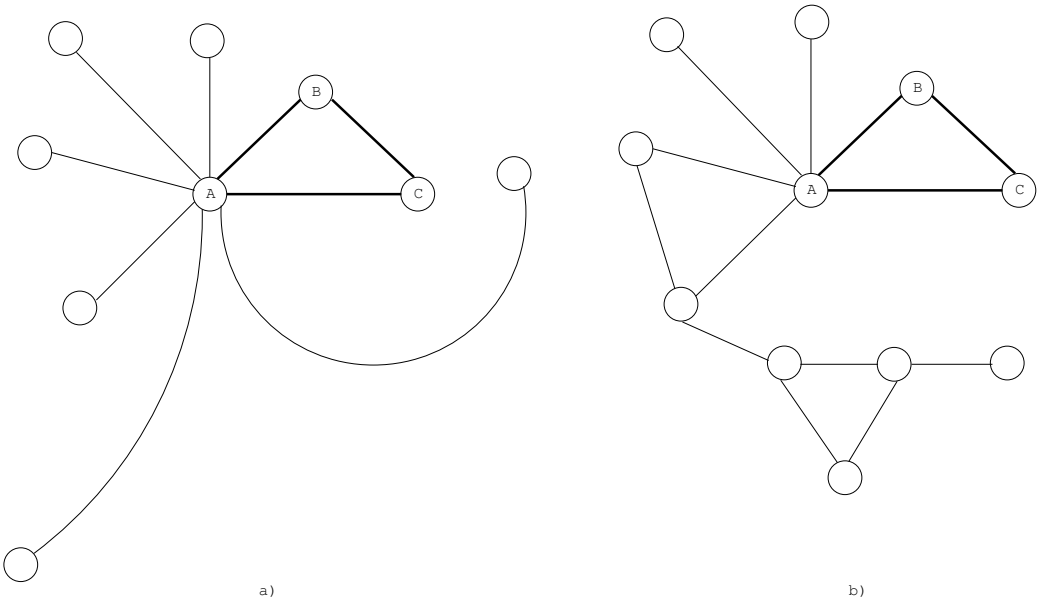


Figura 2.12: Justificativas para propriedades de *small-world* segundo Shudong *et al.* [62]

Algoritmo 4: Algoritmo para criação de modelo *Small-World* de Shudong *et al.* (modelo 1)

Entrada: n (número de nós), p (probabilidade de conexão p).
Saída: Criação da topologia de *small-world* do modelo de Shudong *et al.* (modelo 1).

1 Gere uma sequência de n valores utilizando uma distribuição *Power-Law*

2 Aleatoriamente posicione os n vértices em um plano e atribua um valor gerado no passo 1 como grau desse vértice (d)

3 Para cada um dos vértices crie conexões locais com probabilidade p para os d vértices mais próximos. Essas conexões totalizam d', $d' \leq d$.

4 Estabeleça as $(d - d')$ conexões restantes de forma aleatória (seguindo Algoritmo 5).

Algoritmo 5: Algoritmo para criação de conexões aleatórias do modelo Small-World proposto em [62].

Entrada: Rede com n nós.
Saída: Criação de conexões aleatórias na rede.

1 Crie um conjunto S contendo dv cópias distintas de cada vértice v

2 Para cada v em S, em ordem decrescente de dv faça:

a. Escolha u aleatoriamente em S de maneira que

(i) $u \neq v$ e

(ii) Não existe conexão interligando u e v

Faça $S \leftarrow S - \{u, v\}$

O segundo modelo apresenta uma adaptação ao modelo de Kleinberg [34] para tornar a avaliação mais realista. A adaptação consiste no estabelecimento probabilístico de conexões com base na distância dos participantes da rede. O algoritmo pode ser visualizado a seguir:

Algoritmo 6: Algoritmo para criação de modelo *Small-World* de Shudong *et al.* (modelo 2)

Entrada: n (número de nós), r (expoente de Power-Law).
Saída: Criação m topologia de *Small-World* do modelo de Shudong *et al.* (modelo 2).

1 Gere uma sequência de n valores com distribuição *Power-Law*

2 Aleatoriamente posicione os n vértices em um plano e atribua um dos valores gerados no passo 1 como grau desse vértice (d)

3 Para cada vértice v, a partir da ordem decrescente de grau dos vértices, repita o seguinte passo

a. Escolha um vértice u com probabilidade proporcional a $\frac{d_v}{l_{u,v}{}^r}$, em que l_{uv} é a distância euclidiana entre u e v, de forma que $u \neq v$ e não exista ainda aresta (u, v) criada

Capítulo 3

Protocolo SplitQuest

Estratégias de replicação em redes P2P, em conjunto com algoritmos de busca, devem assegurar que mensagens de pesquisa analisem todos os dados disponíveis na rede. No modelo estruturado, uma mensagem de busca é direcionada para o nó onde o objeto procurado deve estar instalado caso disponível na rede. Como é de conhecimento na literatura, a semântica neste caso é muito limitada, já que os objetos podem ser procurados somente por identificadores gerados por funções de dispersão (*hash*). Por outro lado, redes P2P não estruturadas oferecem maior riqueza semântica ao custo de grande replicação de metadados ou mensagens de buscas pela rede.

Soluções sublineares em redes não estruturadas [25, 71] replicam os metadados em um conjunto aleatório de nós e de maneira semelhante propagam as mensagens de busca por nós também escolhidos aleatoriamente. Essas soluções se baseiam em caminhadas aleatórias para oferecer garantias probabilísticas. Entretanto, a estratégia de caminhada aleatória é um método de busca "cego" que não oferece garantia de cobertura da rede com um número razoável de mensagens e que gera sobrecarga desnecessária, já que não previne que uma mensagem de busca seja enviada para o mesmo nó mais de uma vez.

SplitQuest evita a sobrecarga desnecessária dos métodos de buscas citados anteriormente impondo uma leve estrutura em uma rede não estruturada para permitir que as mensagens de buscas sejam direcionadas para partes específicas e distintas da rede. Em adição a conexões aleatórias, normalmente presentes em redes não estruturadas, SplitQuest dispõe os nós em um anel, ou seja, cada nó possui um predecessor e um sucessor, e os identifica com valores no intervalo $[0, 1]$. Além disso, os nós armazenam localmente os identificadores (IDs) de seus vizinhos. Esta idéia pode parecer semelhante às redes estruturadas, porém como será descrito nas próximas seções, as abordagens de atribuição de identificadores, entrada de nós, replicação de dados e propagação de mensagens de buscas são bem diferentes; as conexões entre os nós ainda permanecem aleatórias, não existe um roteamento bem definido e as mensagens de buscas são propagadas de forma aleatória e livre pela rede. Mais importante ainda é que SpliQuest permite pesquisas de maior riqueza semântica já que as suas buscas não estão restritas ao casamento exato de identificadores e os nós são livres para executar localmente os algoritmos que acharem mais conveniente.

O restante deste capítulo está organizado da seguinte maneira: na Seção 3.1 é apresentado o algoritmo de replicação de índices de SplitQuest. Nessa seção, é apresentada uma demonstração matemática para o tamanho ótimo de grupo para o qual o número de replicações de mensagens é mínimo. A Seção 3.2 descreve o algoritmo de busca e os artifícios utilizados por SplitQuest para garantir rapidez e cobertura da rede durante o procedimento de pesquisa. A Seção 3.3 trata do assunto de exploração da heterogeneidade dos pares e controle de fluxo das mensagens propagadas pela rede. Os métodos de entrada e atribuição de identificadores aos participantes da rede são apresentados na Seção 3.4. Nesta mesma seção, são apresentados fundamentos matemáticos auxiliares a tais métodos. A análise matemática do problema de busca em SplitQuest é modelada na Seção 3.5. Por fim, a Seção 3.6 enumera as principais distinções entre os protocolos de redes estruturadas e o protocolo proposto neste trabalho.

3.1 Replicação de Índices

Este trabalho utiliza duas hipóteses básicas para determinar o funcionamento do protocolo. As hipóteses assumidas por SplitQuest são que os nós estão uniformemente distribuídos no intervalo $[0, 1]$ (Seção 3.4 mostra como se pode obter essa distribuição) e que cada nó é capaz de estimar o tamanho da rede (n). Estimativas de tamanho da rede são problemas bem estudados e que podem ser solucionados por diferentes abordagens, como descrito em [72] e nas referências que lá estão.

A replicação de índices em SplitQuest é realizada em um conjunto aleatório de nós localizados em um subintervalo contíguo do intervalo $[0, 1]$ do anel. Quando nós são distribuídos de forma uniforme por todo intervalo, o número esperado de nós em um subintervalo contíguo é proporcional ao tamanho do subintervalo. Consequentemente, o problema básico neste caso é determinar o tamanho do subintervalo, ou mais especificamente, o tamanho do grupo de replicação que minimiza a sobrecarga de índices e mensagens de buscas propagadas. Quando o valor de n é conhecido, o tamanho do subintervalo com número esperado de nós igual a d é dado por $\frac{d}{n}$. Seja então d o tamanho do grupo e q o número de grupos na rede. O número total de mensagens de replicação e busca de um objeto, $M = q + d$, é dado por:

$$M = \frac{n}{d} + d \tag{3.1}$$

É fácil verificar que d igual a $\sqrt{n}$ minimiza M. Quando mensagens de buscas e dados possuem tamanhos distintos, por exemplo b_1 e b_2 respectivamente, o valor de d que minimiza o total de *bytes* de sobrecarga é dado por $\sqrt{\frac{b_2}{b_1}n}$ tal como observado na Equação 3.6.

$$bytes = b_1 * d + b_2 * \frac{n}{d} \tag{3.2}$$

$$bytes = \frac{b_1 * d^2 + b_2 * n}{d} \tag{3.3}$$

$$2b_1d^2 - b_1d^2 - b_2n = 0 \tag{3.4}$$

$$b_1d^2 = b_2n \tag{3.5}$$

$$d = \sqrt{\frac{b_2}{b_1}n} \tag{3.6}$$

A Figura 3.1 ilustra como os nós são organizados em grupos de replicação no intervalo $[0, 1]$.

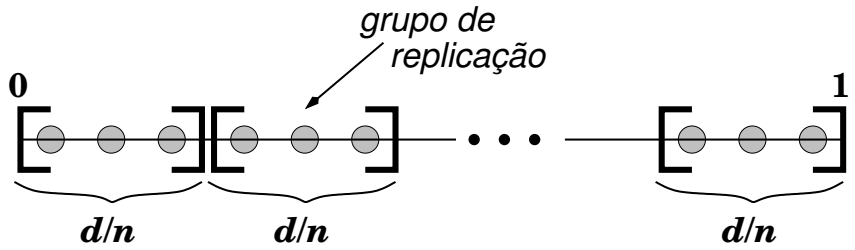


Figura 3.1: Grupos de replicação em SplitQuest

Grupos em SplitQuest são numerados de 1 a $\lceil \frac{n}{d} \rceil$ e o grupo de um nó pode ser computado pela expressão $\lceil \frac{ID \times n}{d} \rceil$. Para instalar referências para um objeto, um nó i envia uma mensagem de instalação para seus vizinhos imediatos, nós com identificadores menores e maiores que i, se eles pertencerem ao mesmo grupo. Os nós conectados diretamente a i verificam e repassam as mensagens recebidas se os seus vizinhos também estão no mesmo grupo (mensagens não são repassadas para pares já visitados). O processo é repetido até que todos os nós do grupo recebam uma cópia da referência.

Estratégias de replicação pró-ativas são bem conhecidas e utilizadas na literatura [44, 26, 25, 30, 71, 73]. Essas estratégias visam disponibilizar o conteúdo em um número maior de nós na rede e consequentemente diminuir o número de passos necessários para solucionar uma pesquisa realizada. Apesar do consumo de banda e espaço necessário para armazenamento das réplicas, este trabalho mostra que o *overhead* de SplitQuest é muito menor que a melhor solução conhecida para o problema de buscas complexas.

3.2 Algoritmo de Busca

Em SplitQuest, nós replicam seus conteúdos em outros nós que estão localizados em um mesmo grupo. Como resultado, mensagens de buscas relacionadas ao conteúdo de um grupo podem ser processadas por qualquer membro do grupo. Essa estratégia de replicação simplifica consideravelmente a propagação dessas mensagens de pesquisa, na medida em que uma busca certamente será respondida, de forma positiva ou negativa,

desde que essa mensagem atinja qualquer nó no grupo alvo. O algoritmo de busca deve garantir portanto, que cada grupo seja visitado pelo menos uma vez e de preferência não mais que uma vez. Para isso SplitQuest utiliza dois artifícios: busca por intervalo e atalhos para grupos.

Busca por intervalo significa que cada mensagem de busca contém, além das palavras-chaves ou expressão, um intervalo $[X, Y]$ que a busca deve cobrir. O algoritmo básico de busca por intervalo é apresentado a seguir. Pequenas alterações serão introduzidas na Seção 3.3 para que a heterogeneidade dos nós possa ser explorada.

Quando um nó i recebe uma mensagem com o intervalo $[X, Y]$, ele inspeciona suas conexões e determina quais vizinhos estão contidos no intervalo recebido. Esses vizinhos são possíveis candidatos para o repasse dessa mensagem de busca. Em seguida, o nó i computa os identificadores de grupos desses possíveis candidatos e mantém apenas um nó para cada grupo (caso exista mais que uma possibilidade para um mesmo grupo, a escolha é feita de maneira aleatória). Dependendo dos grupos selecionados, o nó i decompõe o intervalo inicial em subintervalos e envia mensagens de buscas contendo esses subintervalos para cada um dos nós selecionados.

A Figura 3.2 mostra um exemplo simples de propagação de uma mensagem de busca. Apenas um pequeno conjunto de nós é mostrado. A busca é iniciada em um nó **a** que está conectado aos nós **e**, **b** e **g**. O nó **a** é inicialmente responsável por cobrir todo o intervalo de $[0, 1]$. Assim, este nó cria três mensagens com intervalos $[0, x_4]$, $[x_5, x_8]$ e $[x_8, 1]$ e as envia para os nós **b**, **e** e **g** respectivamente. O nó **b** envia uma mensagem para **c** com intervalo $[0, x_2]$ e o nó **e** envia uma mensagem para **f** com intervalo $[x_6, x_8]$. Finalmente, o nó **c** envia uma mensagem para **d** com intervalo $[0, x1]$ e os nós **b** e **f** são responsáveis por realizar a cobertura dos subintervalos restantes ($[x_2, x_3]$ e $[x_6, x_7]$).

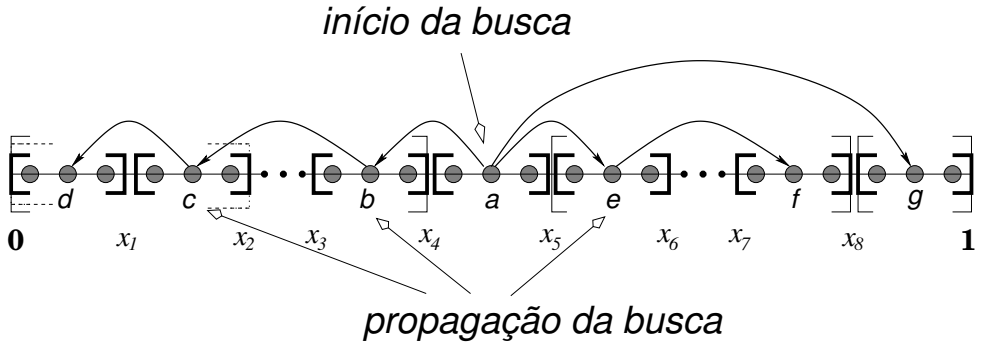


Figura 3.2: Propagação de uma mensagem de busca em SplitQuest

No exemplo acima, o nó **a** possui apenas três conexões. Se o número de conexões fosse maior, uma busca poderia ser propagada para vários outros subintervalos ao mesmo tempo e consequentemente, o tempo de resposta seria menor. É importante enfatizar que os outros nós na figura podem ter outras conexões aleatórias e que neste exemplo são representadas apenas as conexões utilizadas na propagação da mensagem de busca. As conexões omitidas podem ser para outros nós em intervalos já cobertos pela mensagem de busca corrente, para grupos não mostrados ou para nós do mesmo grupo como os

pares mostrados na figura. Além disso, o processo de divisão do intervalo de busca das mensagens sempre tenta construir os subintervalos das mensagens a serem enviadas com mesmo tamanho.

Outro artifício utilizado por SplitQuest para melhorar o desempenho das mensagens de buscas são os atalhos para grupos. Os nós constroem atalhos (conexões diretas para outros pares) para os dois grupos adjacentes aos grupos a que eles pertencem. Esses atalhos garantem que existe sempre um caminho para um grupo, ou seja, um caminho que a cada novo salto levará a mensagem de busca para cobrir um novo grupo, evitando que uma consulta seja realizada várias vezes internamente em um mesmo grupo. Um nó com identificador w pode construir atalhos para grupos adjacentes por conexões para nós com identificadores $w + \frac{1}{d}$ e $w - \frac{1}{d}$, ou para nós com identificadores mais próximos possíveis a esses valores, caso o nó com o valor exato de identificação calculado não esteja presente na rede. Um valor negativo ou um valor maior que um significa que a conexão cruza o identificador 0 e vai em direção ao outro extremo do intervalo. Nesses casos, os identificadores devem ser devidamente corrigidos.

3.3 Heterogeneidade e Controle de Fluxo

Uma característica distinta de uma rede P2P não estruturada é o alto grau de heterogeneidade dos nós formadores de sua população. De uma forma geral, poucos nós estão conectados a um grande número de outros nós, possuem grande poder de computação e largura de banda. Esses nós são normalmente chamados de supernós. A grande maioria dos nós possuem baixo número de conexões e limitações de recursos. Uma boa estratégia de busca deve ser capaz de explorar de maneira inteligente essa diversidade de conexões.

SplitQuest possui melhor desempenho em um ambiente heterogêneo, já que os nós com maior número de conexões podem propagar uma mensagem de busca para diversos grupos em apenas um passo. Entretanto, uma abordagem ingênua de envio de mensagens por um supernó para todos os seus vizinhos que estão em grupos diferentes, por exemplo, pode provocar congestionamentos em nós que possuem limitações de recursos. O motivo é que os supernós, por natureza, estão conectados a um grande número de outros nós e então recebem um grande número de mensagens de buscas. Sempre encaminhar as mensagens de buscas recebidas pode inundar os outros nós que recebem a mensagem e que possuem capacidades menores que o supernó. Para evitar essa situação, SplitQuest limita o número de mensagens de buscas que podem ser enviadas por um nó. O limite é proporcional ao número de conexões de cada nó. Desta forma, um participante que está conectado a um grande número de nós continua sendo capaz de tornar o processo de busca mais rápido, enviando a mensagem para vários outros nós a cada passo, porém com a restrição de encaminhar as mensagens recebidas para apenas uma parcela dos vizinhos.

Para explorar a heterogeneidade dos participantes, o algoritmo de busca deve ser modificado. Um nó, ao receber uma mensagem de busca, cria uma lista de vizinhos que ele conhece e que podem receber a mensagem (receber a mensagem implica estar contido no intervalo de busca) e a partir dessa lista seleciona aleatoriamente um subconjunto desses nós. A cardinalidade desse subconjunto depende do número de conexões do nó.

Por exemplo, caso se ajuste a fração máxima de envio dessas mensagens de busca para 10% e um nó possua 200 conexões, este nó será capaz de enviar uma mensagem de busca para no máximo 20 outros nós. Veremos com mais detalhes na Seção 3.5 a consequência em se utilizar diferentes valores de ramificação das mensagens.

3.4 Atribuição de Identificadores

Uma questão importante em SplitQuest é como os identificadores dos nós são gerados. Os identificadores dos nós devem ser distribuídos uniformemente no espaço de identificadores e os nós não devem depender de um protocolo de roteamento complexo para entrar na rede. Abordagens recentes de sistemas DHT se baseiam em funções de dispersão para gerar identificadores de forma uniforme e distribuída. Essa abordagem, entretanto, requer um protocolo eficiente de roteamento para a entrada de nós na rede. Além disso, cada nó deve manter conexões para locais específicos da rede a fim de garantir limites superiores no roteamento de mensagens de busca. Em Chord [65], por exemplo, os nós devem manter um conjunto de $O(\log n)$ conexões para um limite superior de $O(\log n)$ saltos no roteamento.

A atribuição de um identificador a um novo integrante em SplitQuest é realizada a partir da seleção do ponto médio entre os identificadores de dois pares adjacentes pertencentes à rede. Assim, seja o par X com identificador id_x referente a sua localização no intervalo $[0, 1)$. Seja o par Y um par adjacente ao par X com identificador id_y. Caso um novo integrante Z requisite a entrada a um dos pares, o cálculo do seu identificador na rede seria dado pela equação $\frac{id_x+id_y}{2}$. Essa estratégia é bastante simples, porém, a idéia inicial de selecionar um par (e seu vizinho adjacente) de maneira aleatória e uniforme na rede leva a uma distribuição desbalanceada de identificadores. Essa distribuição desbalanceada é caracterizada por um alto povoamento de uma região do anel em detrimento da outra. Para evitar tal desbalanceamento, SplitQuest utiliza resultados obtidos pelo algoritmo de "redução à metade" [52] e pela extrapolação do algoritmo do "poder de duas escolhas" [49] para k-escolhas. O algoritmo de "redução à metade" é descrito abaixo:

Algoritmo 7: Algoritmo de "Redução à Metade"

Algoritmo de Redução à Metade()

Passo 1: Escolha $t \log n$ pontos aleatórios no intervalo de $[0, 1]$. Neste caso, t corresponde a um parâmetro do sistema.

Passo 2: Calcule os tamanhos dos segmentos entre os identificadores dos pontos selecionados no Passo 1.

Passo 3: O identificador do novo ponto é o valor do ponto médio do maior segmento encontrado no Passo 2.

O algoritmo utilizado neste trabalho é uma variação da abordagem proposta em [52], de forma a não alterar o resultado final. Os pontos representam os pares que fazem parte da rede e possuem identificadores no intervalo $[0, 1)$. Neste caso, o primeiro passo do algoritmo é alterado de forma que um nó que deseja se conectar à rede requisita a um

proxy (um nó que já está na rede) para selecionar k pares de maneira aleatória e uniforme. Mais especificamente, ele seleciona k pares e os maiores intervalos adjacentes a esses. O identificador do nó entrante é o ponto médio do maior segmento. As simulações realizadas neste trabalho mostram que um valor de $k = 3$ apresentam resultados satisfatórios no desempenho da estratégia proposta. Na Seção 3.4.1 será discutido como um nó pode selecionar uma amostra uniforme da rede. Observe que ao selecionar um intervalo, o nó entrante recebe os dois nós que serão os seus vizinhos. Um exemplo pode ser visto na Figura 3.3. Nesse exemplo, tem-se a configuração da rede com quatro pares A,B,C e D com seus respectivos identificadores no intervalo $[0, 1)$. Caso o algoritmo de atribuição utilize como parâmetro o valor de $t = 1$, são necessários escolher dois pares da rede ($t \log 4$ subintervalos) e encontrar o ponto médio do maior segmento escolhido. Supondo, que aleatoriamente sejam escolhido dois segmentos $\overline{AB}$ ($0.65 - 0.1 = 0.55$) e $\overline{CD}$ ($0.9 - 0.75 = 0.15$), pelo cálculo obtem-se $\overline{AB}$ como maior segmento. Em seguida, é então atribuído ao novo participante o identificador que corresponde ao ponto médio do segmento $\overline{AB}$ ($\frac{0.65+0.1}{2} = 0.375$).

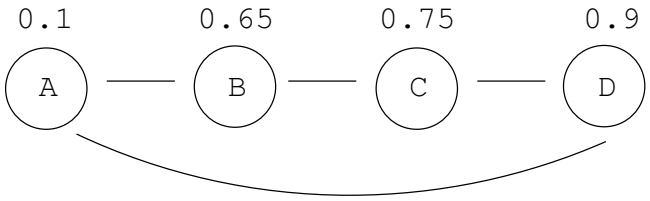


Figura 3.3: Exemplo de atribuição de identificador realizada por SplitQuest

A Figura 3.4 mostra o quão eficiente é o algoritmo proposto para atribuição de identificadores no intervalo $[0, 1]$. Ela representa o número de nós em cada grupo da rede com população de 100 mil nós e tamanho de grupos esperado de $\sqrt{n} \approx 316$. O número de pares em cada grupo é ordenado em ordem crescente nas duas situações. Como pode ser observado, o algoritmo é mais eficiente que uma simples atribuição de identificadores por uma função de dispersão. Todos os tamanhos de grupos apresentados diferem em no máximo um desvio padrão do valor esperado. Além disso, o desvio padrão é muito menor neste caso que no outro. A análise em [52] mostra que $t \log n$ amostras são necessárias para se obter uma distribuição suave. Nos experimentos, observa-se que k pode ser muito menor. Para os identificadores baseados em função de dispersão, foi utilizada a função SHA-1 em cadeias de caracteres formadas por IP:porto. Os endereços IP foram coletados pelo projeto "Route Views" da Universidade do Oregon [13] e os portos foram fixados com valores constantes (1214) usados pelo Kazaa.

3.4.1 Amostragem Uniforme

Amostragem uniforme se tornou uma importante primitiva no desenvolvimento de algoritmos distribuídos para redes P2P de larga escala. O padrão de coleta de amostras em grandes redes se dá pela utilização de caminhadas aleatórias. Em uma caminhada aleatória, todos os vizinhos de um nó i possuem a mesma probabilidade de serem escolhidos como próximo salto, com um valor de $p_{ij} = \frac{1}{d_i}$, onde d_i é o grau do nó i, $j \in \Psi(i)$ e $\Psi(i)$

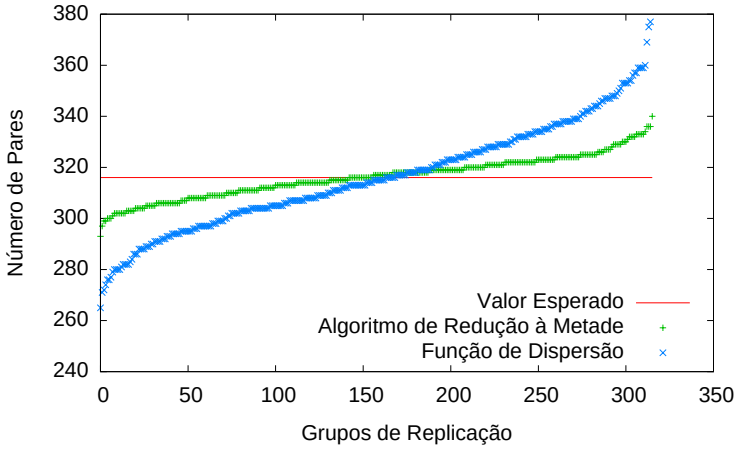


Figura 3.4: Número de nós em grupos de replicação para tamanho de rede de 1000000 de nós

é o conjunto de vizinhos diretamente conectados ao nó i. É sabido que uma caminhada aleatória simples de tamanho $\Theta(\log n)$ em um grafo não direcionado de tamanho n converge para um distribuição estacionária π^T [51]. A probabilidade de um nó ser escolhido como uma amostra da distribuição π^T é igual a $\frac{d_i}{2m}$, em que m é o número de arestas no grafo. Em redes do mundo real, o grau dos nós varia significativamente e como consequência, uma simples caminhada aleatória seleciona nós com diferentes probabilidades. Dessa forma, nós com poucas conexões, possuem menor probabilidade de serem selecionados enquanto que nós com grande número de conexões serão alcançados com maior frequência por mensagens de pesquisa.

3.4.2 Metropolis-Hastings

Awan *et al.* [15] apresentam uma adaptação do algoritmo Metropolis-Hastings [47, 29] para amostragem uniforme em redes P2P. No algoritmo distribuído, cada nó i troca informações a respeito dos graus com seus vizinhos. Quando i recebe a informação dos graus de todos seus vizinhos, ele calcula as probabilidades de transição da seguinte maneira:

$$p_{ij}^{\mathrm{MH}} = \begin{cases} 1/max(d_i, d_j) & \text{if } i \neq j \text{ e } j \in \Psi(i) \\ 1 - \sum_{j \in \Psi(i)} (p_{ij}^{\mathrm{MH}}) & \text{if } i = j \\ 0 & \text{caso contrário.} \end{cases} \tag{3.7}$$

O algoritmo, executado em todos os nós, implicitamente cria uma matriz duplamente estocástica. Uma caminhada aleatória de tamanho $\Theta(\log n)$ com transições de probabilidade calculadas segundo a equação 3.7 resulta em uma amostra uniforme, e assim, todos os nós da rede tem a mesma probabilidade de serem escolhidos $(\frac{1}{n})$.

Com a primitiva de distribuição uniforme, a atribuição de identificadores do algoritmo descrito na Seção 3.4 tem complexidade $k \log n$ no número de mensagens, o que no pior

caso é $O(\log^2 n)$. Este algoritmo permite que um nó se conecte a rede sem depender de um protocolo de roteamento específico e, além disso, livra os nós de manterem informações específicas para realizar o roteamento de mensagens na rede.

3.5 Análise Matemática

A estrutura formada pelos pares e suas conexões em SplitQuest pode ser vista como um grafo aleatório. O conjunto de nós com identificadores em um intervalo específico do anel virtual é um grupo de replicação. Em SplitQuest, grupos são visitados apenas uma vez. Se forem consideradas as conexões de grupos, montadas como conexões aleatórias durante a busca, pode-se modelar a propagação dessas mensagens de buscas para os diferentes grupos como uma difusão (*broadcast*) em uma árvore aleatória. Em cada passo do algoritmo, os grupos são particionados em função das conexões do nó e do intervalo que este deve cobrir. O Algoritmo 8 descreve a partição de diferentes grupos em cada nó, o parâmetro G é o número de grupos a serem visitados. Inicialmente, G é igual ao número total de grupos. Cada g_k representa o conjunto de grupos que cobrem intervalos não sobrepostos.

Algoritmo 8: Algoritmo de propagação de busca para análise do tempo de resposta

SplitBroadcast(G)

Se $G \leq 2$

Pare o particionamento.

Senão

Divida aleatoriamente G em $g_1, \ldots, g_R$, de maneira que $g_1, \ldots, g_R = G - 1$ e R é uma variável aleatória com distribuição fixa.

Para cada $g_k, 1 \leq k \leq R$

SplitBroadcast(g_k)

Quando uma busca se inicia em um nó i, as mensagens de busca devem visitar um conjunto G de grupos disjuntos. O conjunto é dividido em R subgrupos, que dependem das conexões de i, com distribuição dada por $P(R = r) = p_r$, em que p_r é a distribuição de probabilidade em $\{1, 2, \ldots\}$. Condicionado ao evento $\{R = r\}$, para $1 \leq k \leq r$, um grupo está no k-ésimo subconjunto com probabilidade $V_{k,r}$, em que $V_r = (V_{k,r}; 1 \leq k \leq r)$ é um vetor de probabilidade em $\{1, \ldots, r\}$. Se N_k é a cardinalidade do k-ésimo subconjunto, então, condicionado ao evento $\{R = r\}$ e nas variáveis aleatórias $V_{1,r}, \ldots, V_{r,r}$, a distribuição do vetor $(N_1, \ldots, N_r)$ é multinomial com parâmetros $G - 1$ e $(V_{1,r}, \ldots, V_{r,r})$ [50].

A análise da altura (H_G) da árvore aleatória para um distribuição geral de R é complexa e é geralmente obtida por técnicas de análise complexa. Um limite superior mais conservador pode ser obtido quando r pertence ao conjunto $\{1, 2\}$, o que significa que temos no máximo duas ramificações em cada passo da difusão. Devroye [22] mostra que, quando r pertence a $\{1, 2\}$, a altura da árvore é limitada por $O(\log G)$, mais especifica-

mente Devroye [22] mostra que

$$\lim_{G\to\infty} P\{H_G > c \log G\} = 0$$

em que c é uma constante. Um resultado mais forte é derivado em [21] e mostra que

$$H_G / \log G \to 4.31106\ldots \tag{3.8}$$

em probabilidade.

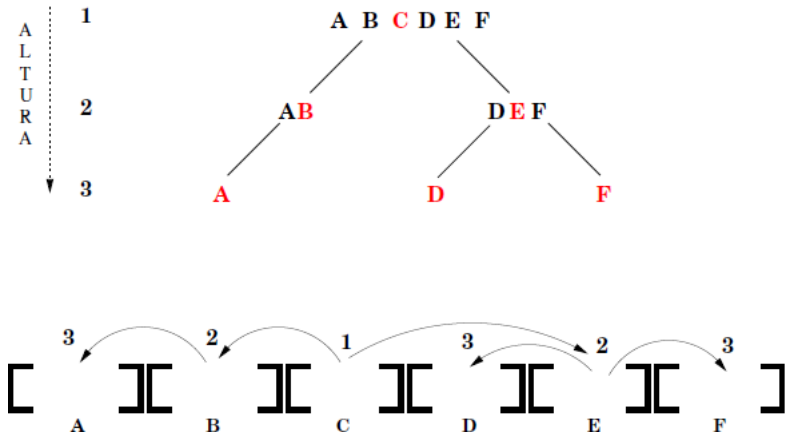


Figura 3.5: Modelagem das buscas por processo de *broadcast* em uma árvore aleatória

É fácil de se verificar que o limite superior derivado acima é também um limite superior para o maior número de saltos que uma mensagem de busca é propagada em SplitQuest. Como SplitQuest pode enviar uma mensagem de busca para mais de dois grupos, o limite superior é, de fato, bastante conservador. A Figura 3.5 apresenta um exemplo dessa propagação da mensagem de busca pelos diversos participantes em uma rede com seis grupos. Nesse exemplo, a mensagem de busca inicia a propagação pelo grupo C. Em seguida são alcançados participantes presentes nos grupos B e E. A partir do grupo B é alcançado o grupo A e do grupo E são alcançados os grupos D e F. Dessa forma em dois saltos, são cobertos todos os grupos e consequentemente, pelo funcionamento de SplitQuest, todos os dados disponibilizados na rede.

A Figura 3.6 retrata os resultados dos experimentos para avaliação do número máximo de saltos que uma busca pode se propagar em SplitQuest. Nesse experimento são utilizados diferentes valores (ramificações) relativos a quantidade de pares atingidos por uma mensagem de busca em um único passo (eixo x). O eixo y retrata o número máximo de saltos que a mensagem de busca atinge. O número de ramificações sofre variação de valores de dois a dez pares. Essa variação no número de pares que recebe uma determinada mensagem de busca a cada passo é importante já que existe um *tradeoff* da velocidade de propagação dessas mensagens e o número de mensagens inseridas na rede. Os experimentos foram desenvolvidos com uma topologia de traço real (mais detalhada no Capítulo 4) e instâncias de tamanhos cem mil e um milhão de pares. Nesse mesmo gráfico

são apresentados os limites superiores teóricos estabelecidos pela Equação 3.8. Esses experimentos confirmam a idéia de que o limite matemático apresentado para o desempenho do protocolo é ainda bastante conservador.

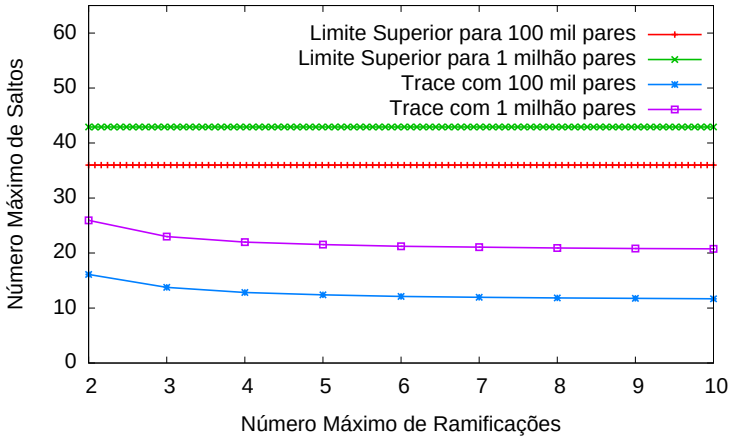


Figura 3.6: Distância máxima de propagação de uma mensagem de busca com diferentes ramificações

3.6 Distinção entre Protocolos de Redes Estruturadas e SplitQuest

Após a descrição do protocolo SplitQuest, é interessante enumerar diferenças básicas desse protocolo e um protocolo de arquitetura estruturada:

1. Os participantes da rede no protocolo SplitQuest não utilizam função de dispersão para obter a localização exata do posicionamento nessa rede. Ao invés disso, é utilizada uma função de atribuição de identificadores baseado no "poder de k escolhas";

2. Todas as conexões de longo alcance de um nó em SplitQuest são realizadas de maneira aleatória, não há roteamento específico (previamente determinado por uma função de dispersão por exemplo) dessas conexões;

3. As mensagens de buscas não seguem caminhos (nós) pré-determinados na rede. Essas mensagens são direcionadas a grupos não explorados e dependem apenas das conexões aleatórias estabelecidas pelos nós participantes;

4. Nós em SplitQuest replicam seus objetos em participantes que pertencem a um mesmo grupo. Todos os participantes do grupo devem receber uma cópia do objeto, publicado por um nó qualquer que esteja localizado no grupo;

5. Em SplitQuest, é possível se fazer buscas complexas, ou seja, buscas por qualquer sequência de caracteres existente em um documento, bem como buscas que utilizam

expressões booleanas ou tipos variados de semântica. SplitQuest pode ainda ser utilizado para aplicações em que buscas são realizadas por intervalo ou direcionadas por função de dispersão.

Capítulo 4

Avaliação Experimental

Para avaliar o desempenho de SplitQuest, foi desenvolvido um simulador na linguagem C++. O simulador utiliza o modelo de simulação de eventos discretos. Boa parte de sua funcionalidade foi inspirada no BubbleStorm [71], uma vez que um dos objetivos deste trabalho é mostrar o desempenho comparativo de SplitQuest com o do BubbleStorm. Uma simples comparação entre diferentes sistemas P2P é difícil em função das diferentes estruturas e algoritmos desenvolvidos. Redes totalmente estruturadas apresentam protocolos específicos de entrada, saída e roteamento de mensagens na rede. Além disso, ao melhor do conhecimento do autor, não apresentam abordagens para buscas complexas em redes P2P. Sistemas baseados em redes não estruturadas também nem sempre oferecem suporte a esse tipo de busca. Uma abordagem mais próxima do objetivo proposto por SplitQuest é apresentada em [71]. As seções a seguir discutem as métricas utilizadas para avaliação, os cenários simulados e os resultados preliminares obtidos.

A decisão de se desenvolver um novo simulador foi tomada pela ausência de simuladores que atendessem integralmente as necessidades do estudo. Além disso, com o simulador pode-se controlar mais precisamente os parâmetros de SplitQuest e do BubbleStorm. Durante o desenvolvimento deste simulador, informações foram trocadas com os autores do BubbleStorm sobre os vários detalhes da implementação utilizados em seus estudos. Os resultados obtidos em [71] foram reproduzidos integralmente por experimentos de forma a averiguar a corretude do simulador desenvolvido e obter uma comparação justa com relação ao desempenho dos protocolos.

A modelagem do simulador foi realizada com base em conceitos de componentes de software. Um componente da simulação corresponde a uma entidade e representa os aspectos gerais de funcionamento existentes durante o processo de avaliação experimental. Podem ser classificadas como entidades da simulação: mensagens (busca, réplica, estabelecimento de conexão, etc), hosts, conexões, escalonador de eventos, etc. Todas as entidades foram desenvolvidas por meio de orientação a objetos e seu uso associado à instanciação de um objeto da respectiva entidade. A estrutura de dados utilizada para representar a simulação da ocorrência de eventos discretos (relacionados as entidade) foi uma Heap-Min. Os eventos inseridos nessa estrutura e posteriormente executados devem obedecer a ordem temporal de acontecimentos de eventos da simulação. Para isso, cada evento da estrutura

possui um campo indicando o tempo o qual será executado determinado evento. Esse tipo de estrutura, associada ao tempo em cada evento, permite que qualquer novo evento seja removido em tempo $O(1)$. A execução de eventos durante a simulação promove a criação de novos eventos que também serão escalonados na estrutura Heap-Min e posteriormente executados. O esquema de funcionamento pode ser observado pelos passos da Figura 4.1. O passo 1 corresponde à remoção de um evento do escalonador para execução. O passo 2 retrata a geração de um novo evento a partir da execução dos eventos existentes. O passo 3 corresponde à inserção na estrutura de dados desses novos eventos criados. Detalhes adicionais das entidades e dos relacionamentos entre essas podem ser obtidos no Apêndice B.

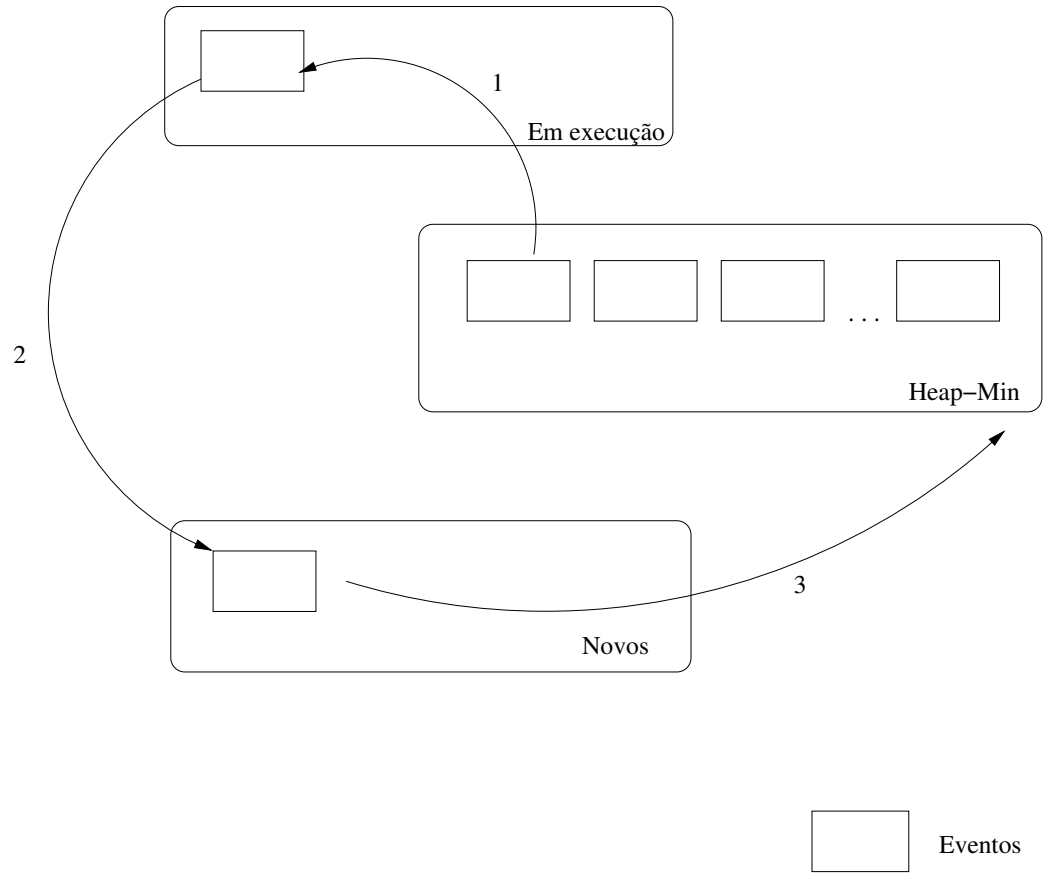


Figura 4.1: Esquema de simulação

O restante deste capítulo está organizado em três seções. A Seção 4.1 apresenta as principais métricas utilizadas para avaliação do desempenho dos protocolos tais como taxa de sucesso, latência e número de mensagens. A Seção 4.2 descreve as configurações dos experimentos realizados. Essas configurações representam as características das topologias adotadas como modelo de simulação e incluem o detalhamento de uma topologia de traço de uma rede real, uma topologia regular e por fim uma topologia de grafo aleatório com distribuição*Power-Law*. Já a Seção 4.3 apresenta os resultados das simulações. Além disso são feitas comparações envolvendo os desempenhos dos protocolos SplitQuest e BubbleStorm.

4.1 Métricas

A avaliação de todas as métricas possíveis de sistemas P2P reais é considerada uma tarefa difícil e até mesmo desnecessária. Algumas métricas são consideradas mais importantes que outras para o desempenho global do sistema. Por isso, os experimentos normalmente se concentram em três métricas de desempenho bem aceitas:

1. Taxa de Sucesso: indica a frequência com que uma mensagem de busca encontra um objeto na rede. Um dos objetivos de SplitQuest é aumentar essa taxa em relação às demais abordagens conhecidas. Como será discutido na Seção 4.3, SplitQuest apresenta taxas de sucesso superiores às taxas atingidas pelo BubbleStorm [71], melhor abordagem conhecida na literatura.

2. Latência: indica o tempo de resposta obtido por uma mensagem de busca quando o objeto procurado é encontrado na rede. Apesar de a Seção 3.5 mostrar um limite superior para o número de saltos percorridos por uma mensagem de busca em SplitQuest, os resultados de simulação mostram que o número máximo de saltos registrados é bem menor que o limite superior teórico demonstrado. A latência é calculada como o tempo de resposta da primeira ocorrência de sucesso da busca.

3. Número de Mensagens: indica a quantidade de mensagens processadas no sistema durante o tempo de simulação. Esse número inclui mensagens destinadas às instalações de réplicas e mensagens de busca. SplitQuest apresenta algoritmos tanto para replicação de objetos quanto para propagação de mensagens de busca com o objetivo de diminuir o número total de mensagens no sistema. Os resultados preliminares mostram que o número de mensagens geradas por SplitQuest é várias vezes menor que o produzido por BubbleStorm.

4.2 Configuração dos Experimentos

Como mencionado anteriormente, o cenário utilizado é semelhante ao cenário proposto em [71], mas também são avaliados novos parâmetros e diferentes cenários para distinguir claramente as duas abordagens. A rede é formada por pares posicionados em diferentes localidades, escolhidas de maneira uniforme e aleatória, na superfície da Terra. Os atrasos de propagação das mensagens são calculados como o tempo que um sinal leva, viajando a velocidade da luz, para cobrir a distância entre par origem e destino. A distância é calculada utilizando a latitude e longitude dos pares na Equação de Haversine [63]. As capacidades de *download* e *upload*, que determinam o atraso da transmissão, estão diretamente relacionadas ao grau dos pares e são ajustadas de acordo com a topologia utilizada. As capacidades serão discutidas posteriormente. Em adição à propagação e o atraso de transmissão, também é introduzido um atraso como uma variável aleatória com distribuição normal entre zero e dez *ms* em cada transmissão de mensagem para contabilizar os atrasos de processamento.

4.2.1 Topologias de Redes P2P

Um importante componente de uma rede P2P consiste em saber como os pares estão conectados entre eles para formar a topologia da rede. Não há consenso na literatura sobre qual topologia melhor representa uma rede P2P, alguns estudos afirmam ser um grafo aleatório com distribuição *power-law*, mas outros discordam [69]. Para ser o mais compreensível possível, as simulações foram feitas com três diferentes topologias e seis diferentes tamanhos de redes (10 mil, 100 mil, 250 mil, 500 mil, 750 mil e 1 milhão de pares). A primeira topologia é um grafo aleatório regular, em que cada par possui grau constante e se conecta aleatoriamente a um número fixo de outros pares, como descrito em [71]. As larguras de banda são as mesmas para todos os pares e iguais a 256 kB/s para *download* e 32 kB/s para *upload*.

Outra topologia utilizada corresponde a um traço de uma rede P2P obtida pelo rastreador de topologia Cruiser introduzido em [69]. Cruiser é um rastreador de redes P2P para Gnutella capaz de coletar informações detalhadas da rede. Sua principal característica é a capacidade de coletar informações o mais rapidamente possível para evitar distorções de configurações da rede que possam ocorrer quando o período de coleta é longo. Para a captura, Cruiser utiliza vários coletores em paralelo que são capazes de coletar um *snapshot* completo da rede em cerca de sete minutos. Essa velocidade permite que não existam mudanças significativas na estrutura da rede durante a execução do rastreador. A topologia de traço obtida por Cruiser é caracterizada pelos autores seguindo uma topologia de *Small-World*, assunto detalhado na Seção 2.5. Para informações adicionais sobre o Cruiser, veja [69, 67].

Na topologia de traço, a capacidade dos pares corresponde as mesmas apresentadas em [71]: pares com grau menor que dez possuem largura de banda de *download* de 128 kB/s e 16 kB/s de *upload*; pares com grau entre dez e vinte possuem largura de banda de *download* de 256 kB/s e de *upload* 32 kB/s; pares com grau entre vinte e oitenta possuem capacidade de *download* e *upload* de 128 kB/s e finalmente, os pares com grau maior que 80 (super pares) possuem taxa de *download* e *upload* de $1,28$ MB/s. Essas capacidades cobrem desde linhas ADSL de baixa-capacidade até altas velocidades de 10 Mbps em links dedicados. Mesmo que essas larguras de banda não representem o estado atual das linhas ADSL, elas foram utilizadas para realizar a comparação com os resultados apresentados em [71]. Além disso, larguras de banda maiores não mudarão os resultados comparativos entre SplitQuest e BubbleStorm. Isso porque no BubbleStorm, os autores caracterizam os cenários homogêneos/heterogêneos e apresentam as proporções de pares na rede com os respectivos valores de largura de banda. No protocolo SplitQuest, são utilizadas exatamente as mesmas proporções de pares com tais larguras de banda. Dessa forma, na simulação, se mudarem os valores de largura de banda, as proporções (e configurações de largura de banda) dos pares ainda serão mantidas em ambas as técnicas simuladas.

A terceira topologia é um grafo aleatório com graus dos pares seguindo uma distribuição *power-law*. Grafos *Power-Law* foram amplamente utilizados para modelar topologias de redes. Em um grafo com esse tipo de distribuição, se os pares estiverem ordenados em ordem decrescente de seus graus, então o i^{esimo} par possui grau D/i^a, em que D é

uma constante e a é um parâmetro fixado em $0,8$. Este valor de a é comumente utilizado em estudos de avaliação de redes P2P. A topologia *power-law* foi gerada com o mesmo número de pares das topologias anteriores. O grau máximo de qualquer par é limitado a oitocentos e as larguras de banda dos pares são ajustadas de acordo com os graus e seguem as mesmas regras descritas acima para a topologia traço.

4.2.2 Cenários Dinâmicos

A natureza dinâmica dos sistemas P2P é considerada um importante parâmetro. Em sistemas envolvidos com publicação de conteúdo, esse parâmetro se torna mais importante para que se possa avaliar o impacto da variação das condições. Foram simulados dois cenários diferentes no que diz respeito ao dinamismo dos pares. O primeiro e mais simples cenário não possui eventos de saídas dos pares e a população se mantém constante durante todos os experimentos (estático). Este cenário simples é utilizado principalmente para contrastar a eficácia das duas técnicas na busca por objetos na rede. O segundo cenário considera tanto entrada quanto saída dos pares durante a publicação de conteúdo e propagação das mensagens de buscas (com *churn*). Entradas e saídas de pares são escalonadas em taxas semelhantes para manter a população próxima a um valor constante. *Churn* é um evento bastante conhecido e é gerado pela chegada e partida independente dos milhares (ou até milhões) de pares existentes na rede. Alguns estudos na literatura tratam os diversos aspectos que podem estar envolvidos na análise de um cenário dinâmico [68, 61]. Durante as simulações nos cenários dinâmicos temos a entrada e saída de aproximadamente 10% dos nós da rede (de forma a manter a comparação justa com o protocolo BubbleStorm).

4.2.3 Aplicação Par a Par

Existem inúmeras aplicações P2P que podem se beneficiar de um substrato que oferece buscas complexas tal como SplitQuest. Uma comparação correta entre diferentes protocolos implica na utilização de um mesmo cenário de simulação. Assim, a aplicação utilizada neste trabalho se baseia nas descrições apresentadas em [71]. Foi simulada uma aplicação de Wiki em que pares publicadores injetam periodicamente novos artigos e outros pares realizam buscas por esses artigos. Nesta aplicação, artigos são representados por um tipo de dado de 2060 *bytes* (2 kB para metadados mais 60 *bytes* do cabeçalho TCP/IP). Mensagens de buscas são representadas por tipos de dados de 160 *bytes* (100*bytes* para os dados da busca mais 60 *bytes* do cabeçalho TCP/IP). A cada segundo, são inseridos cem artigos em pares escolhidos de maneira aleatória na rede. Esses pares seguem os seus algoritmos de replicação e iniciam o processo de instalação de réplicas. No caso de SplitQuest, os pares têm que replicar os seus próprios conteúdos em seus grupos. No BubbleStorm, os pares disponibilizam seus conteúdos em pares no caminho de uma caminhada aleatória com múltiplas ramificações. Por questões de economia de memória durante as simulações, cada par representa seus próprios dados por vetores binários, em que cada valor binário representa a presença ou ausência de um dado específico.

A fase inicial de simulação consiste na entrada dos participantes na rede até que uma quantidade pré-definida de pares seja atingida. Após a rede atingir o equilíbrio (o tamanho da rede permanece próximo ao tamanho pré-definido), se inicia a fase de replicação de conteúdo. Um determinado par que deseja publicar um dado na rede dá início ao processo de replicação para seus vizinhos imediatos com objetivo de realizar a cobertura de todo o grupo (como descrito na Seção 3.1). Após um período pré-definido de espera para que esse dado seja replicado, é iniciada, em algum nó aleatório da rede, a propagação da mensagem de busca para esse conteúdo publicado. A mensagem de busca é propagada pelos pares da rede (e consequentemente diferentes grupos) por um determinado período e a verificação de sucesso ou insucesso do procedimento de busca é feita após 140 segundos.

Todas as simulações foram executadas em um servidor com dois processadores AMD *Quad-Core Opteron* de 2GHz e 16GB de memória principal rodando Linux.

4.3 Resultados Experimentais

Nesta seção, são apresentados os resultados numéricos para comparações entre SplitQuest e BubbleStorm. São exibidos os resultados para as métricas definidas na Seção 4.1 sob três diferentes topologias e cenários. Em todas as figuras abaixo, a abscissa x representa diferentes tamanhos de redes e a ordenada y representa uma medida específica computada como média de onze execuções independentes. Redes com diferentes tamanhos e topologias permitem comparações das duas abordagens com relação à escalabilidade e exploração da heterogeneidade dos pares. Salvo se explicitado, os parâmetros para SplitQuest são os mesmos descritos no Capítulo 3. Para todos os gráficos abaixo, foram calculados os intervalos de confiança para um grau de confiança de 95%. Entretanto, em situações em que os intervalos ficaram extremamente pequenos e de difícil visualização, eles não foram plotados.

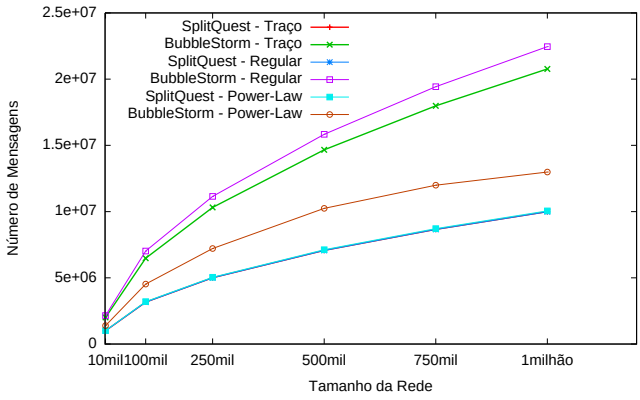


Figura 4.2: Número de mensagens para um cenário dinâmico de simulação

Na Figura 4.2, são apresentados os resultados para o número de mensagens quando 5000 artigos são inseridos na rede. Para cada artigo, uma busca é iniciada cem segundos depois de que o artigo é inserido. Este tempo é assim ajustado para permitir a instalação

das réplicas na rede. Os resultados são analisados 140 segundos depois que se iniciam as buscas. Mensagens de buscas com sucesso que levam mais que 140 segundos não são contabilizadas e são consideradas como insucesso. Esse intervalo é importante para medir o tempo de resposta dos protocolos e mostrar o quão rápido eles são capazes de encontrar os objetos. Os resultados discutidos abaixo são para o cenário dinâmico (o cenário estático apresenta resultados semelhantes).

O número total de mensagens na Figura 4.2 corresponde a soma de mensagens de dados e mensagens de buscas durante todo o período de simulação. Pode-se perceber que, em todas as três topologias e tamanhos de redes, o número de mensagens para SplitQuest é quase o mesmo. Percebe-se ainda as três linhas como uma única na parte inferior da figura. Este comportamento é fácil de ser compreendido, já que SplitQuest utiliza o tamanho da rede como parâmetro para determinar o número de réplicas e mensagens de buscas. Uma importante conclusão que pode ser tirada da figura é que o número de mensagens que SplitQuest gera é independente da topologia e de parâmetros particulares dos pares, como a distribuição dos graus. Pode-se perceber também, que o algoritmo de atribuição de identificadores, discutido na Seção 3.4, é capaz de distribuir os pares uniformemente pelo anel virtual, já que o número de mensagens para as três diferentes topologias permanece o mesmo.

A observação mais importante da Figura 4.2 é que SplitQuest supera o BubbleStorm em todas as três diferentes topologias. Os ganhos variam de 50% até 52% em uma topologia regular, de 53.5% até 55.5% na topologia de traço e de 22.5% até 30.5% em uma topologia *power-law*. Os maiores ganhos foram observados para redes maiores, o que torna os resultados mais significativos já que redes P2P possuem tamanho da ordem de 1 milhão de pares. Os menores ganhos, obtidos na topologia *power-law*, podem ser explicados de acordo com a forma que BubbleStorm calcula o número de réplicas (protocolo próprio $qd = c^2n$, em que q e d correspondem às mensagens de buscas e réplicas de dados estimadas pela rede, c um parâmetro de certeza e n o tamanho estimado da rede - tal protocolo é apresentado com detalhes na Seção 2.4.2), o que resulta em um menor número de réplicas dependendo da distribuição do grau dos pares.

Um cenário estático é simplista e irrealista, mas permite que as idéias chaves dos algoritmos sejam avaliadas. A Figura 4.3 mostra que, em condições perfeitas de funcionamento, SplitQuest sempre alcança 100% de taxa de sucesso em todas as topologias e tamanhos de rede. BubbleStorm, que utiliza um método "cego" de busca e não oferece garantias de cobertura da rede, atinge 98% de taxa de sucesso. A taxa de sucesso do BubbleStorm pode ser incrementada pela mudança do parâmetro de certeza c. Entretanto, o número de mensagens aumentará consideravelmente, já que os valores de q e d aumentarão proporcionalmente a c^2.

A taxa de sucesso em um cenário dinâmico e mais realista (com *churn*) pode ser observada na Figura 4.4. É possível também verificar que, para SplitQuest, a taxa de sucesso decai aproximadamente 0.17% em comparação ao cenário estático, atingindo taxas de sucesso acima de 99.8%. Este decaimento pode ser explicado pelos pares que deixam a rede durante o momento de processamento dos dados e mensagens de buscas. Um par que recebe uma mensagem de replicação de dado deve distribuir a mensagem para outros pares no mesmo grupo. Caso esse par saia, pares ainda não cobertos não receberão a

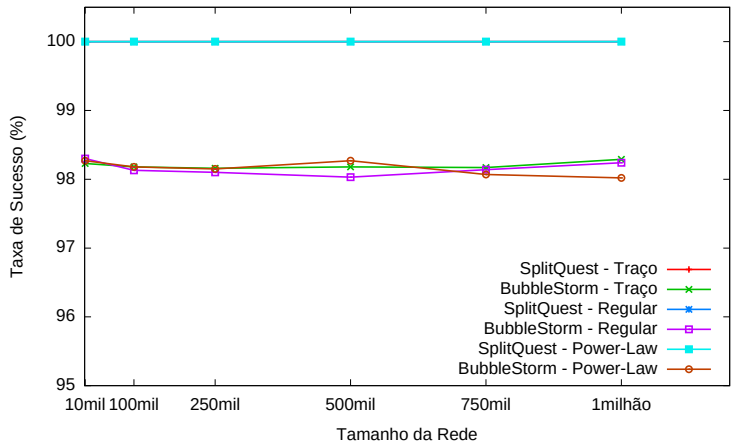


Figura 4.3: Taxa de sucesso com uma população estática de pares

mensagem. Além disso, um par que deixa a rede pode não repassar uma mensagem de busca para outros pares no intervalo que essa deve cobrir. Embora BubbleStorm replique mais dados e mensagens de buscas que SplitQuest, ele não atinge taxa de sucesso superior a 98.5%. Este resultado se deve principalmente ao método de busca "cego" que não explora todas as possibilidades de pares. Os resultados mostram ainda que SplitQuest é resiliente a eventos de entradas e saídas dos pares e mantém altas taxas de sucesso mesmo sob condições de *churn*.

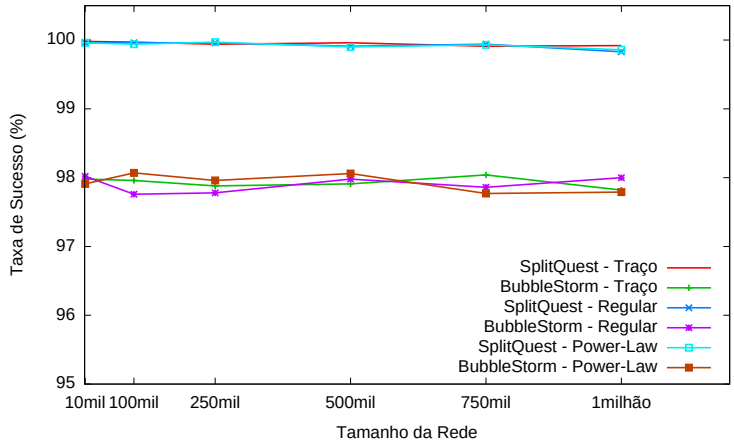
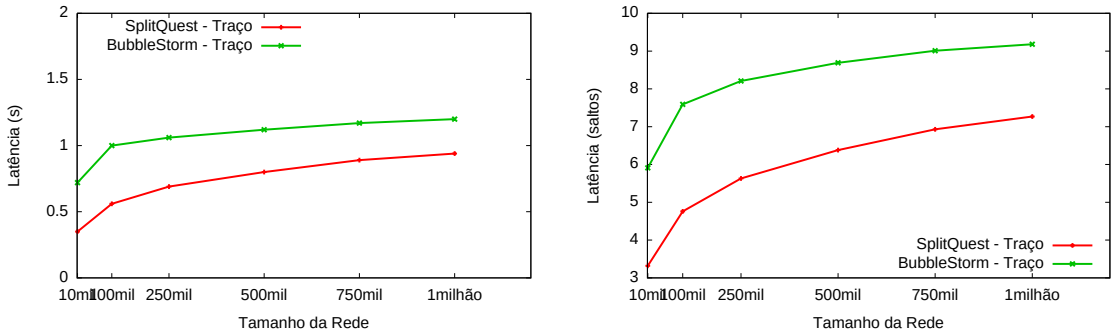


Figura 4.4: Taxa de sucesso com uma população dinâmica de pares

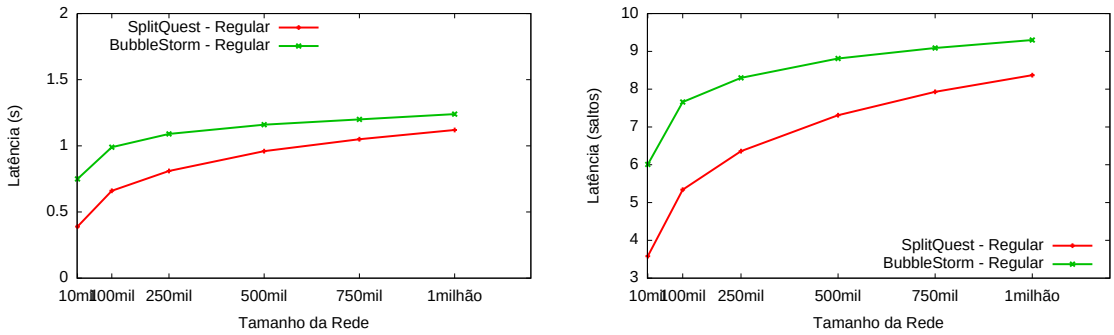
Uma das principais características do BubbleStorm é a baixa latência na resolução de mensagens de buscas. O seu desempenho reduziu significativamente o tempo de resposta em comparação à primeira abordagem baseada em caminhadas aleatórias [25]. SplitQuest também é capaz de atingir tempos baixos de resolução, como pode ser visto na Figura 4.5. Esta redução na latência é possível porque SplitQuest explora agressivamente a heterogeneidade dos pares e espalha as mensagens mais rapidamente que BubbleStorm. Pares com grande número de conexões são capazes de enviar mensagens de buscas para diversos

grupos diferentes em um simples passo. Além disso, as buscas direcionadas evitam *loops* e cobrem rapidamente todos os grupos da rede. A capacidade em explorar a heterogeneidade dos pares é mais evidente quando se compara as latências da topologia regular com a topologia *power-law*. Visto que a topologia *power-law* possui pares com mais que oitocentas conexões, SplitQuest resolve uma mensagem de busca com quatro saltos a menos que na topologia regular.



(a) Latência em segundos (b) Latência em saltos

Figura 4.5: Latência da primeira resposta de sucesso para uma mensagem de busca em cenário dinâmico e topologia traço

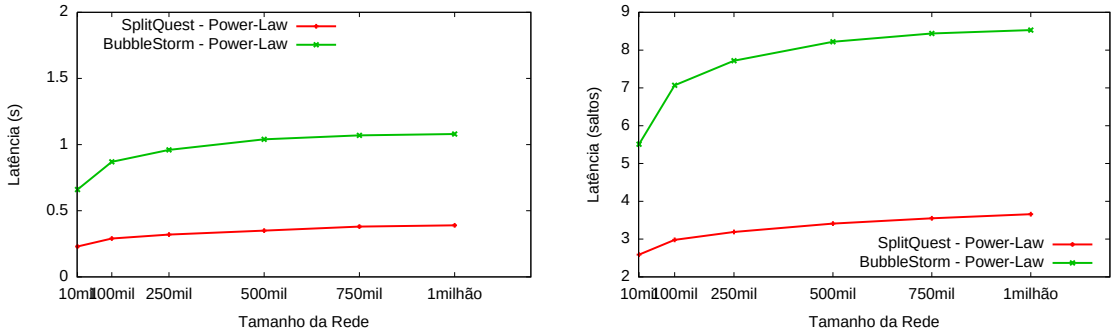


(a) Latência em segundos (b) Latência em saltos

Figura 4.6: Latência da primeira resposta de sucesso para uma mensagem de busca em cenário dinâmico e topologia regular

Para todas as topologias e tamanhos de redes, SplitQuest apresenta resultados melhores em termos de tempo e número de saltos que o BubbleStorm. O ganho mais considerável é observado na topologia *power-law*, em que a latência é reduzida em até 59%. O pior dos casos acontece na topologia regular, porém SplitQuest continua superando BubbleStorm em aproximadamente 11%.

A Seção 3.3 trata do assunto de heterogeneidade e controle de fluxo em SplitQuest. Como se sabe, nós em uma rede P2P possuem diferentes configurações e capacidades. Alguns nós são mais capacitados a processar e enviar mensagens que outros, seja por



(a) Latência em segundos (b) Latência em saltos

Figura 4.7: Latência da primeira resposta de sucesso para uma mensagem de busca em cenário dinâmico e topologia *power-law*

questões de desempenho de processamento ou pela qualidade da largura de banda. Um problema a ser gerenciado nesse caso é a distribuição das mensagens a partir de um nó com uma mensagem de busca. Replicar essa mensagem em todos os nós vizinhos pode, em algumas situações, provocar a sobrecarga em um nó com pouca capacidade de recepção dessas mensagens. SplitQuest gerencia esse envio para um número percentual máximo de vizinhos.

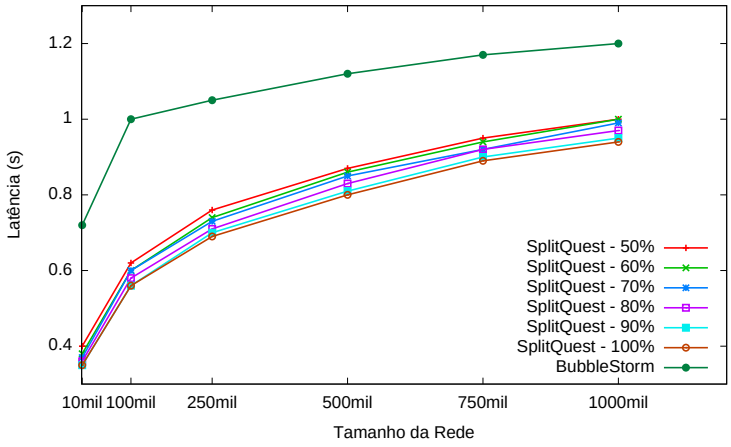


Figura 4.8: Variação da latência da primeira resposta de sucesso em cenário dinâmico e topologia traço para diferentes capacidades dos pares

O desempenho dessas avaliações pode ser verificado nas Figuras 4.8, 4.9 e 4.10. Nesses experimentos, um nó, ao receber uma mensagem de busca, faz a selecão dos nós vizinhos localizados em grupos disjuntos a serem cobertos pela mensagem de busca. Porém, ao invés de enviar para todos os possíveis nós desses grupos o nó fica responsável por escolher aleatoriamente uma porcentagem dos vizinhos que receberão essa mensagem de busca. Essas porcentagens nos experimentos varia de 50% a 100% dos vizinhos. BubbleStorm não aborda tal distribuição por porcentagem pois utiliza um fator denominado *splitfactor*

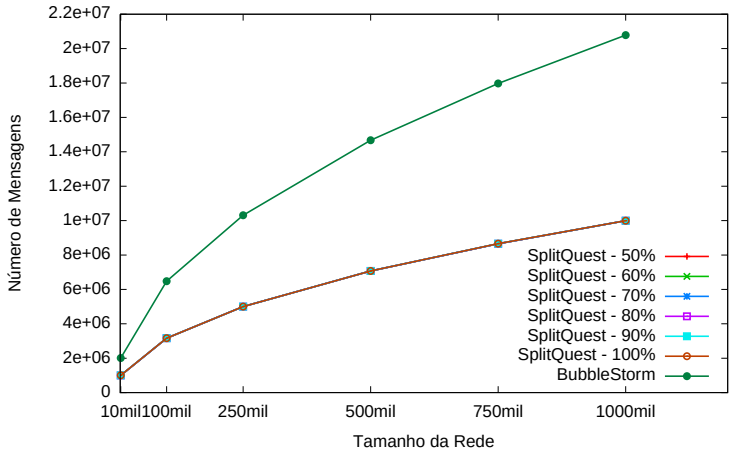


Figura 4.9: Variação do número de mensagens em cenário dinâmico e topologia traço para diferentes capacidades dos pares

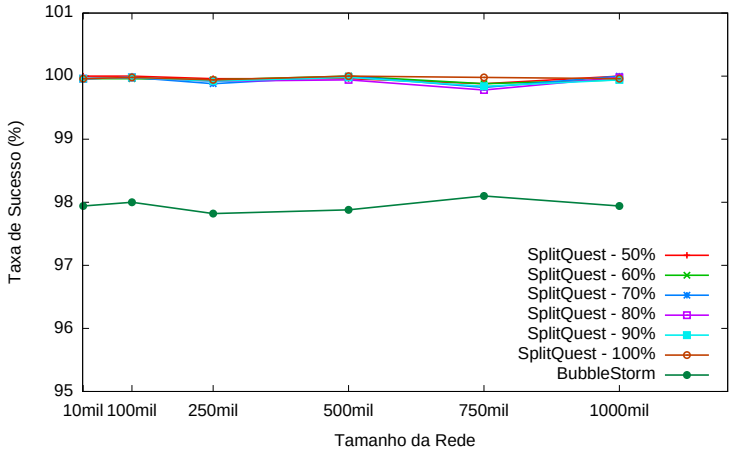


Figura 4.10: Variação da taxa de sucesso em cenário dinâmico e topologia traço para diferentes capacidades dos pares

(s) para distribuir a mensagem de busca pelos s vizinhos imediatos. Na Figura 4.8 podemos observar uma redução na latência conforme se aumenta a porcentagem de vizinhos a receberem a mensagem de busca em um determinado passo. Isso é facilmente compreendido, visto que quanto maior for o número de vizinhos a receber a mensagem de busca, mais rapidamente essa busca cobre os grupos existentes na rede e consequentemente mais rapidamente a busca é solucionada. Uma outra observação importante a ser feita é que, independente da porcentagem de vizinhos utilizada para distribuir a mensagem de busca, o desempenho de SplitQuest é sempre superior ao desempenho de BubbleStorm. Além disso, a decisão de utilizar apenas 50% dos vizinhos possíveis, reduz a eficiência da técnica em aproximadamente $12,5\%$ com relação a utilização de todos os vizinhos.

A Figura 4.9 representa o desempenho dos protocolos BubbleStorm e SplitQuest com relação ao número de mensagens inseridas na rede durante o processo de simulação. A novidade desse gráfico com relação aos experimentos apresentados na Figura 4.2 é a variação, em SplitQuest, do percentual de vizinhos a receber a mensagem de busca a cada passo de propagação. Apesar dos experimentos utilizarem diferentes percentuais de vizinhos, a quantidade de mensagens geradas por SplitQuest é semelhante independente desse percentual (fato que justifica as linhas não visíveis devido a sobreposição dos valores). Esse comportamento é esperado e pode ser justificado pelo funcionamento básico de propagação das mensagens de buscas e publicação de conteúdo. Como em SplitQuest cada par representa o conteúdo de todos os participantes do grupo e a cobertura dos grupos significa encontrar qualquer conteúdo que venha a ser publicado por um par em toda a rede, as mensagens então são direcionadas com o intuito de cobrir esses grupos. Isso faz com que as mensagens de buscas tenham quantidade diretamente relacionadas a quantidade de grupos na rede e não com o percentual de pares que receberão a mensagem a cada passo da busca. Já quantidade de mensagens de publicação de conteúdo está relacionada ao tamanho dos grupos apenas. Isso faz com que a quantidade total de mensagens (publicação e busca) inseridas na rede durante a simulação permaneça independente do percentual de vizinhos utilizados para propagar uma mensagem.

Outro aspecto analisado, em virtude da variação percentual de vizinhos, é a taxa de sucesso. A Figura 4.10 representa essa avaliação e torna evidente que a taxa de sucessos também é independente desse percentual. As taxas de sucessos atingidas por SplitQuest estão quantitativamente relacionadas com a cobertura aos grupos da rede, ou seja, com que frequência um grupo com um determinado conteúdo procurado, está sendo alcançado. Assim, independente do número de grupos atingidos a cada passo da propagação de uma mensagem de busca, o que importa, com relação ao cálculo da taxa de sucesso atingida por SplitQuest, é alcançar um nó do grupo publicador para que a busca seja dada como sucesso, e não o quão rápido essa busca é propagada.

Capítulo 5

Conclusão e Trabalhos Futuros

O objetivo inicial deste trabalho não consistia em única e exclusivamente fazer comparações com outra técnica de replicação conhecida na literatura e sim agregar, em um único protocolo, as melhores características de dois mundos de arquiteturas distintas (descentraliza estruturada e não-estruturada) criadas com o propósito de solucionar um mesmo problema: buscas complexas em redes P2P.

A estrutura híbrida para se evitar replicação desnecessária de mensagens de buscas e para se garantir a cobertura da rede são pontos fundamentais do protocolo. Com esse esquema de funcionamento baseado em divisão dos pares em grupos e buscas por intervalo, SplitQuest faz com que uma mensagem de busca seja comparada diante de todos os dados de um par que receber uma mensagem de busca, deixa os pares livres para escolher o algoritmo local de busca que desejar e garante que cada grupo da rede seja visitado uma única vez. Resultados de simulação demonstram, sob variadas topologias, tamanhos de rede e condições de dinamismo, que SplitQuest é capaz de alcançar taxas de sucesso altas, baixas latências nas mensagens de buscas e um pequeno número de mensagens em comparação com outros algoritmos de replicação pró-ativos discutidos na literatura. Este trabalho mostra que o protocolo desenvolvido pode ser utilizado para aplicações com paradigma P2P, mais em particular, para aplicações que geram pequenas quantidades de dados, tais como wikis, microblogs e twitters distribuídos. A replicação do conteúdo em todos os pares de um mesmo grupo sugere que apenas informações pequenas (metadados) dos arquivos sejam replicadas, evitando o problema de espaço de armazenamento físico nos pares pertencentes aquele grupo.

Esse trabalho apresenta ainda vários seguimentos do seu projeto que podem ser incrementados ou até mesmo modificados para exploração total de sua arquitetura. Os resultados aqui apresentados são baseados em simplificações dos protocolos implementados no simulador e na hipótese de que todos os nós possuem informações de parâmetros globais da rede, como, por exemplo, o número de nós participantes. O tamanho da rede pode ser obtido por algoritmos de estimação como o desenvolvido em [72]. Esses algoritmos devem ser incorporados a qualquer ambiente de simulação de redes P2P não-estruturadas para um avaliação mais realista do ambiente de simulação. Como sequência deste trabalho, é interessante adaptar o simulador para utilizar algum algoritmo da literatura para

estimação de tamanho da rede.

Outra questão de bastante interesse é estudar e detalhar de forma mais específica a análise matemática de SplitQuest. Este trabalho utiliza resultados matemáticos de limite superior para árvores aleatórias mostrados por Devroye [22]. Esse limite é muito conservador se comparado ao número máximo de saltos de SplitQuest obtidos por simulação.

Com base no estudo mais detalhado da análise matemática do problema, novas estratégias em SplitQuest podem ser implementadas visando aumentar a taxa de sucesso e reduzir o tempo de resposta das buscas. Algumas possibilidades são: o envio de mensagens de pesquisa para mais de um nó do mesmo grupo, gerando assim múltiplos caminhos de busca; e replicação de dados em mais de um grupo, disponibilizando um número maior de referências em pares da rede. A consequência imediata dessas alterações seria tornar o protocolo mais robusto à transiência e falhas dos nós e ainda mais eficiente com relação ao tempo de resposta das buscas realizadas.

Essas implementações de novas estratégias em SplitQuest levantam ainda questões a respeito da replicação baseada em popularidade do objeto. É certo que objetos populares não devem ser replicados a uma mesma proporção que objetos não-populares, visto que não existe a necessidade de tornar mais popular um objeto já bem conhecido na rede na mesma proporção que objetos raros ou inexistentes. Para fazer tal alteração, novas abordagens devem ser investigadas em SplitQuest, inclusive com relação à análise matemática do problema proposto. Todas essas alterações feitas no funcionamento do protocolo irão, com certeza, afetar as métricas analisadas neste trabalho já que maior número de réplicas instaladas significa maiores possibilidades de espalhamento das mensagens de buscas e como consequência, mudança na avaliação dos quesitos latência, taxa de sucesso e número de mensagens.

Referências Bibliográficas

[1] Bittorrent. http://www.bittorrent.com/. [acesso em 16/08/2010].

[2] Emule. http://www.emule.com/. [acesso em 16/08/2010].

[3] Folding@home. http://folding.stanford.edu. [acesso em 16/08/2010].

[4] Gnutella. http://gnutella.wego.com/. [acesso em 16/08/2010].

[5] Icq. http://www.icq.com/. [acesso em 16/08/2010].

[6] Justintv. http://www.justin.tv/. [acesso em 16/08/2010].

[7] Kazaa. http://www.kazaa.com/. [acesso em 16/08/2010].

[8] Msn. http://www.msn.com/. [acesso em 16/08/2010].

[9] Napster. http://www.napster.com. [acesso em 16/08/2010].

[10] Seti@home. http://setiathome.berkeley.edu/. [acesso em 16/08/2010].

[11] Skype. http://www.skype.com/. [acesso em 16/08/2010].

[12] Sopcast. http://www.sopcast.com/. [acesso em 16/08/2010].

[13] University of Oregon Route Views Project. http://www.routeviews.org. [acesso em 16/08/2010].

[14] L. Adamic, R. M. Lukose, A. R. Puniyani, e A. B. Huberman. Search in Power-Law Networks. *Physical Review E*, 64, 2001.

[15] A. Awan, R. A. Ferreira, S. Jagannathan, e A. Grama. Distributed Uniform Sampling in Unstructured Peer-to-Peer Networks. In *Proceedings of The 39th IEEE Hawaii International Conference on Systems Sciences (HICSS'06)*. Kauai, HI, Janeiro 2006.

[16] M. W. Berry, Z. Drmac, e E. R. Jessup. Matrices, Vector Spaces and Information Retrieval. *SIAM Rev.*, 41:335–362, 1999.

[17] M. W. Berry, S. T. Dumais, e G. W. O'brien. Using Linear Algebra For Intelligent Information Retrieval. *SIAM Review*, 37:573–595, 1995.

[18] A. Bosselaers, R. Govaerts, e J. Vandewalle. SHA: A Design for Parallel Architectures. In *Advances in Cryptology, Proceedings Eurocrypt'97, LNCS 1233*, páginas 348–362. Springer-Verlag, 1997.

[19] Y. Chawathe, S. Ratnasamy, L. Breslau, N., e S. Shenker. Making gnutella-like P2P systems scalable. In *Proceedings of the 2003 ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'03)*, páginas 407–418. Karlsruhe, Alemanha, 2003.

[20] E. Cohen e S. Shenker. Replication Strategies in Unstructured Peer-to-Peer Networks. In *Proceedings of the 2002 ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'02)*, páginas 177–190. Pittsburgh, PA, 2002.

[21] Luc Devroye. A Note on the Height of Binary Search Trees. *Journal of ACM*, 33(3):489–498, 1986.

[22] Luc Devroye. Universal Limit Laws for Depths in Random Trees. *SIAM Journal on Computing*, 28(2):409–432, 1998.

[23] D. Doval e D. O'Mahony. Overlay Networks: A Scalable Alternative for P2P. páginas 79–82. Piscataway, NJ, EUA, 2003.

[24] M. Faloutsos, P. Faloutsos, e C. Faloutsos. On power-law relationships of the internet topology. In *Proceedings of the 1999 ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'99)*, páginas 251–262. Massachusetts, EUA, 1999.

[25] R. A. Ferreira, M. K. Ramanathan, A. Awan, A. Grama, e S. Jagannathan. Search with probabilistic guarantees in unstructured peer-to-peer networks. In *Proceedings of The 5th IEEE International Conference on Peer-to-Peer Computing (P2P'05)*, páginas 165–172. Konstanz, Alemanha, 2005.

[26] C. Gkantsidis, M. Mihail, e A. Saberi. Random Walks in Peer-to-Peer Networks. In *Proceedings of IEEE INFOCOM 2004, the Twenty-Third Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'04)*, páginas 120–130. Hong Kong, China, 2004.

[27] C. Gkantsidis, M. Mihail, e A. Saberi. Hybrid Search Schemes for Unstructured Peer-to-Peer Networks. In *Proceedings of IEEE INFOCOM 2005, the Twenty-Fourth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'05)*, páginas 1526–1537. Miami, FL, Março 2005.

[28] G. H. Golub e C. F. V. Loan. *Matrix Computations*. The Johns Hopkins University Press, 1996.

[29] W. Hastings. Monte carlo sampling methods using Markov chains and their applications. *Biometrika*, 57:97–109, 1970.

[30] S. Ioannidis e P. Marbach. Absence of Evidence as Evidence of Absence: A Simple Mechanism for Scalable P2P Search. In *Proceedings of IEEE INFOCOM 2009, the Twenty-Eight Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'09)*. Rio de Janeiro, Brasil, 2009.

[31] S. Iyer, A. Rowstron, e P. Druschel. S. SQUIRREL: A Decentralized, Peer-to-Peer Web Cache. In *Proceedings of the Twenty-First ACM Symposium on Principles of Distributed Computing (PODC'02)*, páginas 213–222. Monterey, CA, EUA, 2002.

[32] F. Kaashoek e D. R. Karger. Koorde: A Simple Degree-Optimal Hash Table. In *Proceedings of the Second International Workshop on Peer-to-Peer Systems (IPTPS'03)*, páginas 98–107. Berkeley, CA, Fevereiro 2003.

[33] V. Kalogeraki, D. Gunopulos, e D. Zeinalipour-Yazti. A Local Search Mechanism for Peer-to-Peer Networks. In *Proceedings of the Eleventh International Conference on Information and Knowledge Management (CIKM'02)*, páginas 300–307. Nova Iorque, NY, EUA, 2002.

[34] J. Kleinberg. The Small-World Phenomenon: An Algorithm Perspective. In *Proceedings of the 32nd annual ACM symposium on Theory of computing (STOC'00)*, páginas 163–170. Nova Iorque, NY, EUA, 2000.

[35] J. Kleinberg. Small-World Phenomena and the Dynamics of Information. In *In Advances in Neural Information Processing Systems (NIPS'01)*. MIT Press, Vancouver, Canada, 2001.

[36] J. Kleinberg. The Small-World Phenomenon and Decentralized Search. *Math Awareness Month*, 37, 2004.

[37] J. Kleinberg. Complex Networks and Decentralized Search Algorithms. In *Proceedings of the International Congress of Mathematicians (ICM'06)*. Madrid, Espanha, 2006.

[38] T. A. Letsche e M. W. Berry. Large-Scale Information Retrieval With Latent Semantic Indexing. *Inf. Sci.*, 100:105–137, 1997.

[39] Jinyang Li, Boon Thau Loo, Joseph M. Hellerstein, M. Frans Kaashoek, David R. Karger, e Robert Morris. On the Feasibility of Peer-to-Peer Web Indexing and Search. In *Proceedings of the Second International Workshop on Peer-to-Peer Systems (IPTPS'03)*, páginas 207–215. 2003.

[40] C. Lin, Y. Chang, S. Tsai, e C. Chou. Distributed Social-based Overlay Adaptation for Unstructured P2P Networks. In *Proceedings of IEEE INFOCOM 2007, The 26th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'07)*. Anchorage, AK,EUA, 2007.

[41] P. Lopes e R. A. Ferreira. SplitQuest: Controlled and Exhaustive Search in Peer-to-Peer Networks. In *Proceedings of the Seventh International Workshop on Peer-to-Peer Systems (IPTPS'10)*. San Jose, CA, EUA, Abril 2010.

[42] P. Lopes e R. A. Ferreira. Uma Arquitetura Híbrida para Buscas Complexas em Redes P2P. In *Proceedings of the 28th Brazilian Symposium on Computer Networks and Distributed Systems (SBRC'10)*. Gramado, Brasil, Maio 2010.

[43] X. Luo, Z. Qin, J. Han, e H. Chen. DHT-Assisted Probabilistic Exhaustive Search In Unstructured P2P Networks. In *Proceedings of the 22nd IEEE International Parallel and Distributed Processing Symposium (IPDPS'08)*. Miami,FL,EUA, 2008.

[44] Q. Lv, P. Cao, E. Cohen, K. Li, e S. Shenker. Search and Replication in Unstructured Peer-to-Peer Networks. In *Proceedings of the 16th international conference on Supercomputing (ICS'02)*, páginas 84–95. Nova Iorque, NY, EUA, 2002.

[45] D. Malkhi, M. Naor, e D. Ratajczak. Viceroy: A Scalable and Dynamic Emulation of the Butterfly. In *Proceedings of the Twenty-First ACM Symposium on Principles of Distributed Computing (PODC'02)*. Monterey, CA, Julho 2002.

[46] G. S. Manku, M. Bawa, P. Raghavan, e V. Inc. Symphony: Distributed Hashing in a Small World. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS'03)*, páginas 127–140. Seattle, WA, EUA, 2003.

[47] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. H. Teller, e E. Teller. Equations of State Calculations by Fast Computing Machines. *Journal of Chemical Physics*, 21:1087–1091, 1953.

[48] M. Mitzenmacher e E. Upfal. *Probabilistic Analysis and Randomized Algorithms*. Cambridge University Press, 2004.

[49] Michael Mitzenmacher. The Power of Two Choices in Randomized Load Balancing. *IEEE Trans. Parallel Distrib. Syst.*, 12(10):1094–1104, 2001.

[50] H. Mohamed e P. Robert. A Probabilistic Analysis of Some Tree Algorithms. *The Annals of Applied Probability*, 15(4):2445–2471, 2005.

[51] R. Motwani e P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[52] Moni Naor e Udi Wieder. Novel Architectures for P2P Applications: The Continuous-Discrete Approach. *ACM Transactions on Algorithms (*Eletronic Edition*)*, 3(3), Agosto 2007.

[53] C. H. Papadimitriou, H. Tamaki, P. Raghavan, e S. Vempala. Latent Semantic Indexing: a Probabilistic Analysis. In *Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'98)*. Seattle, WA, EUA, 1998.

[54] L. Peterson e B. Davie. *Computer Networks*. Elsevier Science and Technology Books, 2003.

[55] K. P. N. Puttaswamy, A. Sala, e B. Y. Zhao. Searching for Rare Objects using Index Replication. In *Proceedings of IEEE INFOCOM 2008, The 27th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'08)*. Phoenix, AZ, EUA, 2008.

[56] C. Raiciu, F. Huici, M. Handley, e D. Rosenblum. ROAR: increasing the flexibility and performance of distributed search. In *Proceedings of the 2009 ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'09)*, páginas 291–302. Nova Iorque, NY, EUA, 2009.

[57] S. Ratnasamy, P. Francis, M. Handley, R. Karp, e S. Schenker. A Scalable Content-Addressable Network. In *Proceedings of the 2001 ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'01)*, páginas 247–254. San Diego, CA, 2001.

[58] S. Ratnasamy, I. Stoica, e S. Shenker. Routing Algorithms for DHTs: Some Open Questions. In *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS'02)*, páginas 45–52. Cambridge, MA, EUA, 2002.

[59] A. Rowstron e P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg (MIDDLEWARE'01)*, páginas 329–350. Londres, Inglaterra, 2001.

[60] S. Milgram. Small-World Problem. *Psychology Today*, 1:61–67, 1967.

[61] S. Saroiu, P. Krishna Gummadi, e S. D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proceedings of the Multimedia Computing and Networking (MMCN'02)*. San Jose, CA, EUA, 2002.

[62] J. Shudong e A. Bestavros. Small-world Characteristics of Internet Topologies and Implications on Multicast Scaling. *The Computer Networks Journal (COMNET): The International Journal of Computer and Telecommunications Networking*, 50:648–666, 2006.

[63] R. W. Sinnott. Virtues of the Haversine. *Sky and Telescope*, 68:159, 1984.

[64] K. Sripanidkulchai, B. Maggs, e H. Zhang. Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems. In *Proceedings of IEEE INFOCOM 2003, The 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'03)*. San Franciso,CA,EUA, 2003.

[65] I. Stoica, R. Morris, D. Karger, F. Kaashoek, e H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Proceedings of the 2001 ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'01)*, páginas 149–160. San Diego, CA, 2001.

[66] D. Stutzbach e R. Rejaie. Characterizing Unstructured Overlay Topologies in Modern P2P File-sharing Systems. In *In Internet Measurement Conference (IMC'05)*, páginas 49–62. 2005.

[67] D. Stutzbach e R. Rejaie. Capturing Accurate Snapshots of the Gnutella Network. In *Proceedings of IEEE INFOCOM 2006, the 25th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'06)*. Barcelona, Espanha, Abril 2006.

[68] D. Stutzbach e R. Rejaie. Understanding Churn in Peer-to-Peer Networks. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement (IMC'06)*, páginas 189–202. Rio de Janeiro, RJ, Brasil, 2006.

[69] D. Stutzbach, R. Rejaie, e S. Sen. Characterizing Unstructured Overlay Topologies in Modern P2P File-Sharing Systems. *IEEE/ACM Transactions on Networks (*Eletronic Edition*)*, 16(2), Abril 2008.

[70] C. Tang, Z. Xu, e S. Dwarkadas. Peer-to-peer Information Retrieval Using Self-Organizing Semantic Overlay Networks. In *Proceedings of the 2003 ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'03)*, páginas 175–186. Karlsruhe,Alemanha, 2003.

[71] W. W. Terpstra, J. Kangasharju, C. Leng, e A. P. Buchmann. BubbleStorm: Resilient, Probabilistic, and Exhaustive Peer-to-Peer Search. In *Proceedings of the 2007 ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'07)*. Tóquio, Japão, 2007.

[72] W. W. Terpstra, C. Leng, e A. P. Buchmann. Brief Announcement: Practical Summation via Gossip. In *Proceedings of the 2007 Symposium on Principles of Distributed Computing (PODC' 07)*, páginas 390–391. Portland, OR, Agosto 2007.

[73] S. Tewari e L. Kleinrock. Proportional Replication in Peer-to-Peer Networks. In *Proceedings of IEEE INFOCOM 2006, the 25th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'06)*. Barcelona, Espanha, Abril 2006.

[74] D. J. Watts e S. H. Strogatz. Collective Dynamics of Small-World Networks. *Nature*, 393:440–442, 1998.

[75] Wikimedia Foundation Reports. http://wikimediafoundation.org/wiki/, 2010.

[76] B. Y. Zhao, J. D. Kubiatowicz, e A. D. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Relatório técnico, UC Berkeley, 2001.

Apêndice A

Fundamentos Matemáticos

A.1 Distribuição Binomial

A Distribuição Binomial é uma distribuição de probabilidade discreta que se refere ao número de sucessos em uma sequência de tentativas independentes. Em cada uma das tentativas realizadas existe apenas duas possibilidades de resultados, sucesso ou fracasso e cada possibilidade possui probabilidade constante de acontecimento.

A.1.1 Função de Probabilidade

Seja X a variável aleatória que indica o número de tentativas que resultam em sucesso e possui uma distribuição binomial com parâmetros n e p tal que $X \sim B(n, p)$. A probabilidade de que existam exatamente k sucessos é dada pela função de probabilidade:

$$f(k; n, p) = \binom{n}{k} p^k (1-p)^{n-k} \tag{A.1}$$

em que,

$k = 0, 1, 2, ..., n$ corresponde ao número de sucessos
n = número total de ensaios/experimentos
p = probabilidade de sucesso
$(1 - p)$ = probabilidade de fracasso (Distribuição de *Bernoulli*)

A.2 Distribuição Multinomial

A Distribuição Multinomial é uma extensão (generalização) da Distribuição Binomial. Neste caso um evento não possui como espaço amostral apenas duas possibilidades, sucesso ou fracasso, e pode assumir valores variados. Basicamente, a Distribuição Multi-

nomial responde a seguinte questão Qual a probabilidade de que um evento com espaço amostral de 1 a k tenha uma certa configuração (quantidade de vezes que um evento ocorre) mediante n experimentos realizados?

A.2.1 Função de Probabilidade

Seja um experimento que permite ter k resultados diferentes $(A_1, ..., A_k,)$ formando o espaço amostral. Considere n o número de tentativas do experimento com p_i, $i = 1, ..., k$ associados aos k resultados permanecendo constante durante a repetição dos experimentos, com $\sum_{i=1}^{k} = p_i = 1$. Sejam as variáveis $X_i, ..., X_k$ as variáveis aleatórias que representam os números de ocorrências associados aos k resultados possíveis com função de probabilidade:

$$P(X_1 = N_1, X_2 = N_2, ..., X_k = N_k) = \frac{n!}{n_1!n_2!n_k!} p_1^{n_1} p_2^{n_2} p_3^{n_3} ... p_k^{n_k} \tag{A.2}$$

A análise do protocolo SplitQuest pode ser resumida a um problema de distribuição multinomial. Se forem consideradas as conexões de grupos, montadas como conexões aleatórias durante a busca, pode-se modelar a propagação dessas mensagens de buscas para os diferentes grupos como uma difusão (*broadcast*) em uma árvore aleatória. Em cada passo do algoritmo, os grupos são particionados em função das conexões do nó e do intervalo que este deve cobrir.

Na raiz da árvore de difusão tem-se então o conjunto de todos os grupos que devem ser alcançados pela mensagem de busca. A mensagem de busca inicia a propagação para nós que representam alguns dos grupos ainda não cobertos pela mensagem de busca. Essa mensagem contém subintervalos desses grupos. Isso significa que o nó inicial da árvore, contendo todos os grupos subdivide-se em l subgrupos com probabilidade $P(G = l) = p_l$ em $\{2, 3\}$... . Condicionado a esse evento de divisão de grupos, a descida na árvore promove o alcance aos grupos formadores dessa rede. Existe então uma determinada probabilidade em se atingir cada um desses filhos (nós da árvore).

Se definir N_i como o tamanho do i-ésimo grupo e condicioná-lo ao evento de $G = l$ com probabilidades de descida na árvore para atingir cada filho $V_{1,l}, V_{2,l}, ..., V_{l,l}$ pode-se concluir que o problema consiste em uma distribuição multinomial. Na Figura A.1 tem-se um exemplo com um conjunto inicial de grupos A, B, C, D, E e F.

A partir da definição da distribuição de probabilidade multinomial

$$P(X_1 = N_1, X_2 = N_2, ..., X_k = N_k) = \frac{n!}{n_1!n_2!n_k!} p_1^{n_1} p_2^{n_2} p_3^{n_3} ... p_k^{n_k} \tag{A.3}$$

e substituindo as variáveis que interessam:

$$P(N_1 = m_1, N_2 = m_2, ..., N_l = m_l) = \frac{n!}{m_1!m_2!m_k!} p_1^{m_1} p_2^{m_2} p_3^{m_3} ... p_l^{m_l} \tag{A.4}$$

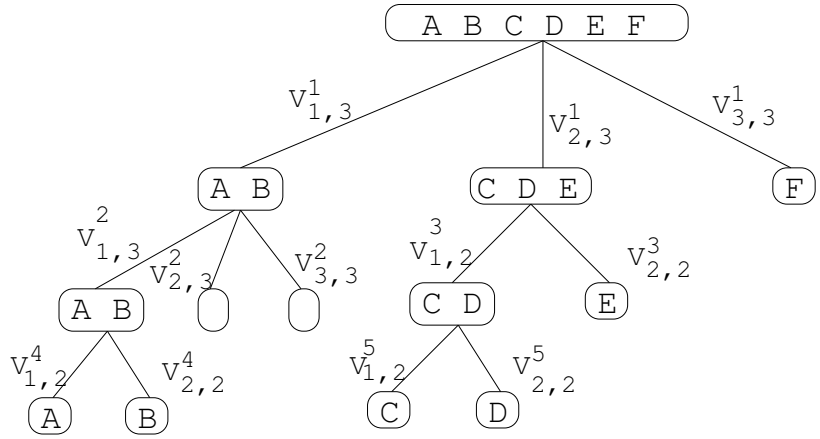


Figura A.1: Árvore de probabilidades

$$P(N_1 = m_1, N_2 = m_2, ..., N_l = m_l) = \frac{n!}{m_1!m_2!m_k!} V_{1,l}^{m_1} V_{2,l}^{m_2} V_{3,l}^{m_3} ... V_{l,l}^{m_l} \tag{A.5}$$

$$P(N_1 = m_1, N_2 = m_2, ..., N_l = m_l) = \frac{n!}{m_1!m_2!m_k!} \Pi_{i=1}^{k} V_{k,l}^{m_k} \tag{A.6}$$

com $n = m_1 + m_2 + ... + m_k$.

Apêndice B

P2Pns: Simulador de Redes P2P

Para realizar a simulação dos experimentos e quantificar o funcionamento dos protocolos SplitQuest e BubbleStorm foi desenvolvido uma ferramenta na linguagem C++ com suas devidas simplificações de ambientes simulados. As características lógicas da simulação dos experimentos já foram descritas no Capítulo 4 e não serão abordadas neste capítulo. O simulador P2Pns apresenta uma arquitetura capaz de suportar múltiplos algoritmos de buscas em redes P2P. A sua estrutura básica possui capacidade de execução dos procedimentos de criação da rede, estabelecimento de conexões de novos pares, instalação de referências para objetos e propagação de mensagens de buscas pela rede. Neste capítulo, são descritas as classes mais representativas do simulador desenvolvido, incluindo métodos, atributos e o relacionamento entre os objetos da arquitetura.

B.1 Classe *Network*

Descrição: A classe *Network* representa a rede de sobreposição par-a-par da aplicação. Essa classe é a principal instância do simulador **P2Pns**. Ela define a sequência de instruções executadas pela aplicação e todo o relacionamento entre os objetos envolvidos na simulação. Outras atividades que competem a esta classe são: inicializar os atributos das métricas de avaliação, construir a rede (incluindo estabelecer conexões dos pares no anel e conexões aleatórias de longo alcance), escalonar entrada e saída de pares, instalar réplicas, inserir mensagens de buscas na rede e processar todos eventos escalonados pelo simulador. Uma observação importante é que todos os eventos (com exceção dos eventos de saída de pares) a serem processados na rede são armazenados em uma estrutura min-heap.

Atributos

- **generalScheduler***: Escalonador responsável por fazer o tratamento de eventos gerais da rede. Eventos gerais correspondem a entrada de pares, instalação de réplicas e processamento de mensagens de buscas.
- **departureScheduler***: Escalonador responsável por fazer exclusivamente o tratamento de eventos de saída de pares da rede. A decisão de implementação de um

escalonador separado dos eventos gerais da rede se justifica pela utilização de uma árvore min-heap para armazenar tais eventos. Um número menor de eventos implica em um desempenho mais eficiente da estrutura.

Funções Membros

- **void run(double)**: Responsável por invocar para execução o próximo evento do escalonador. Esse método deve decidir qual dos dois escalonadores (general ou departure) possui o evento mais próximo do tempo de execução do relógio global do sistema. O parâmetro double define até que tempo na ordem cronológica do relógio é que se pode executar os eventos armazenados nos escalonadores.

- **void simulation(char**)**: Método que fornece a sequência lógica de execução da simulação. Possui a sequência de execução dos procedimentos relacionados à simulação. Fazem parte dessa sequência desde a abertura e interpretação dos arquivos relativos a topologia da rede a ser simulada, até a invocação do procedimento de início da avaliação das métricas. O parâmetro char** refere ao arquivo de configuração passado como parâmetro à simulação.

- **void insertHost(host*)**: Responsável por inserir um novo nó na rede. O nó inserido na rede é o parâmetro Host* do método.

- **void removeHost(host*)**: Responsável por remover um nó da rede. O nó a ser removido da rede é o parâmetro Host* do método.

- **void process(scheduler*, message*)**: Método responsável por processar mensagens referentes às modificações estabelecidas sob atributos da rede. Essas mensagens são eventos passados como parâmetro message*. Qualquer evento que venha necessitar a ser escalonado durante a execução deste método é inserido no escalonador scheduler * passado como parâmetro.

- **void createBubblecast(double, int*)**: Realiza o escalonamento de eventos de replicação de conteúdo e propagação de mensagens de buscas pela rede. O parâmetro double se refere ao tempo corrente de execução do simulador enquanto que o int* se refere ao identificador do último objeto replicado na rede.

- **host* scheduleJoin(double, double)**: Esse método é o responsável por escalonar um evento de entrada de um nó na rede. O primeiro parâmetro double corresponde ao IP do nó a entrar. O segundo parâmetro se refere ao tempo na ordem cronológica de execução que o host vai entrar na rede. Esse método retorna o apontador para o host com entrada escalonada.

- **void scheduleDeparture(host*, double)**: Esse método é o responsável por escalonar um evento de saída de um nó na rede. O parâmetro host* se refere ao nó que terá saída escalonada, enquanto que o valor double se refere ao tempo em que será realizada a saída da rede.

- **void printInfo()**: Método de impressão das informações relativas aos nós pertecentes à rede.

- **void doLongRangeConnections(double)**: Esse método é o responsável por estabelecer as conexões aleatórias de longo alcance dos nós. Ele também é o responsável por estabelecer as conexões com grupos adjacentes, uma conexão com um nó do grupo sucessor e uma conexão com um nó do grupo predecessor. O parâmetro double se refere ao tempo em que o nó estabelecerá as conexões.

- **void mapZones()**: Função auxiliar para realizar o mapeamento dos grupos para os nós que iniciam a rede. Essa função tem como objetivo calcular a que grupo um nó pertence.

- **host* scheduleJoinChurn(double, double)**: Realiza o escalonamento de entrada de nós durante períodos de *churn* na rede. O primeiro parâmetro double corresponde ao IP, enquanto o segundo parâmetro se refere ao tempo de entrada do nó.

- **void readConfigFile()**: Esse método é o responsável por fazer a leitura dos parâmetros de configuração da simulação. Esses parâmetros incluem características como arquivo de entrada, tamanho da rede, porcentagem de pares a receber as mensagens (tratamento de fluxo), arquivo de saída, etc.

- **void updateResultFile()**: Esse método é responsável por armazenar os resultados da simulação corrente. O nome do arquivo de simulação gerado é criado a partir dos parâmetros utilizados no método readConfigFile.

- **void updateQueueSize()**: Método utilizado para iniciar os valores do tamanho das filas. Esse método auxilia na estimativa do tamanho médio das filas de mensagens em cada nó.

Atributos Externos

- ***hosts[MAXHOSTS]**: Estrutura de dados responsável por armazenar os apontadores para pares da rede. O tamanho da estrutura é definido pelo valor de MAXHOSTS.

- ***hosts_long[MAXHOSTS]**: Estrutura de dados responsável por armazenar os apontadores para os pares de rede que já estabeleceram conexões de longo alcance. O tamanho da estrutura é definido pelo valor de MAXHOSTS.

- **currentSize**: Indica o tamanho atual da rede.

- **long_size**: Indica o número de nós que já estabeleceram todas as suas conexões.

- **vector <Host*> markZones**: Estrutura de dados auxiliar que realiza a marcação dos limites dos grupos. Cada posição dessa estrutura indica o nó que marca o início de um grupo da rede.

- **leftPeers**: Indica o número de pares que já saíram da rede durante o período de *churn*.

- **joinChurn**: Indica o número de pares que entraram na rede após o início do período de *churn*.

Funções auxiliares

- **double urand()**: Método que gera uma variável aleatória uniforme.
- **double getExpoRandomVar(double)**: Método que gera um valor seguindo uma distribuição exponencial. O parâmetro double corresponde ao valor de lambda da distribuição.

B.2 Classe *Host*

Descrição:A classe *Host* representa os pares da rede de sobreposição da aplicação. Esses pares devem ser instanciados durante a criação de um objeto da classe *Network* pelo estabelecimento de conexões por parte dos participantes.

Atributos

- **int id**: Identificador do nó.
- **int queueSize**: Registra o tamanho da fila de mensagens do nó.
- **int zoneMarker**: Indica se o par é responsável por marcar um grupo ou não. Marcar significa delimitar um determinado grupo no intervalo $[0, 1)$ estabelecido pelo anel. Caso seja, ele armazena o grupo que ele é reponsável por marcar.
- **double lat**: Indica a posição geográfica do nó com relação à latitude. Esse valor é gerado aleatoriamente no intervalo -90 a $+90$.
- **double longt**: Indica a posição geográfica do nó com relação à longitude. Esse valor é gerado aleatoriamente no intervalo -180 a 180.
- **double arrivalTime**: Indica o momento da entrada do nó na rede.
- **double departureTime**: Indica o momento em que o nó deve sair da rede.
- **int uploadCapacity**: Indica a capacidade de *upload* do nó.
- **int downloadCapacity**: Indica a capacidade de *download* do nó.
- **int state**: Indica o estado atual do nó. Esse estado pode variar em ocupado (*busy*) e ocioso (*idle*). Um nó estará ocupado caso esteja processando alguma mensagem e ocioso caso contrário.
- **int position**: Indica a posição de armazenamento das informações do nó na estrutura de dado do simulador.

- **int positionLong**: Indice a posição de armazenamento das informações do nó na estrutura de dado referente as conexões de longo alcance.
- **int zone**: Indica a que grupo o nó pertence.
- **int con**: Indica o número de conexões estabelecidas pelo nó com base em seu grau.
- **vector<Connection*> connectList**: Estrutura de dados responsável por armazenar as conexões do nó relativas a estrutura.
- **vector<Connection*> longRangeList**: Estrutura de dados responsável por armazenar as conexões de longo alcance.
- **Host *shorcutAnt**: Armazena apontador para nó predecessor ao nó corrente.
- **Host *shortcutSuc**: Armazena apontador para nó sucessor ao nó corrente.
- **vector <Host*> shortsAnt**: Armazena apontador para o nó do grupo predecessor ao seu.
- **vector <Host*> shortsSuc**: Armazena apontador para o nó do grupo sucessor ao seu.
- **list<Message*> messageList**: Estrutura de dados responsável por armazenar as mensagens destinadas ao nó.
- **int qsize**: Indica o número de mensagens de busca e instalação de referências que o nó recebeu.
- **bitset<MAXDATA> data**: Armazena os dados que um nó possui. $MAXDATA$ corresponde ao número máximo de dados possível.

Funções Membros

- **bool searchData(int)**: Procura localmente por um objeto com identificador especificado como parâmetro.
- **void connectTo(connection*)**: Estabelece uma conexão do nó com o anel. A conexão é o parâmetro connection* do método.
- **void connectToLong(connection*)**: Estabelece uma conexão de longo alcance do nó. A conexão é o parâmetro connection* do método.
- **int getZone()**: Retorna o valor do grupo a que o nó pertence.
- **void checkInterval(int, int, vector<int>, host*)**: Verifica se um nó está localizado em um grupo que esteja no intervalo limitado pelos grupos passados como parâmetro (inteiros). O parâmetro vector<int> corresponde ao endereço dos nós que já estão inclusos no grupo escolhido para receber uma mensagem de busca. Host* corresponde ao apontador para o nó que terá o identificador do grupo analisado.

- **void sendQueryMessage(host*, queryMessage*, scheduler***: Método responsável por enviar uma mensagem de busca para um host. O host e a mensagem são passados como parâmetros. Outro parâmetro é o escalonador que será utilizado para atribuir a mensagem para execução.
- **void handleQueryMessage(scheduler*, querymessage*)**: Esse método é responsável por fazer o tratamento de uma mensagem de busca (querymessage*) em um nó. Esse tratamento consiste na manipulação da mensagem para decidir para quais grupos a mensagem deve ser enviada. O paramêtro scheduler* se refere ao escalonador que receberá os eventos (nesse caso, as mensagens que irão ser inseridas no conjunto de eventos futuros da simulação).
- **void getShortcut(int)**: Atribui um determinado atalho para um nó. O parâmetro int corresponde qual dos atalhos deverão ser atribuídos (as opções são predecessor (0) e sucessor (1)). O nó ao invocar o método, procura analisar quais são as opções disponíveis para tornar atalho na configuração corrente da rede.
- **void process(scheduler*, message*)**: Procedimento responsável por realizar o processamento de uma mensagem em um nó. Essa mensagem e o escalonador de mensagens são passados como parâmetro.

Atributos Externos

- **double queriesTimeMatch**[**MAXDATA**]: Atributo que mantém as informações de tempo de quando um determinado objeto foi encontrado. Cada posição dessa estrutura referencia o tempo que um objeto com determinado identificador foi encontrado. Esse identificador está relacionado à posição do vetor utilizada para marcar a existência da referência encontrada.
- **int queriesHops**[**MAXDATA**]: Atributo que mantém as informações do número de saltos necessários para encontrar um determinado objeto. Cada posição dessa estrutura diz respeito ao número de saltos utilizados para que um objeto (com identificador relacionado à posição do vetor) fosse encontrado.
- **double queriesTimeStart**[**MAXDATA**]: Atributo que mantém as informações do tempo de início da busca por um objeto. Cada posição dessa estrutura diz respeito ao tempo exato que a busca se iniciou para o objeto com identificador relativo à sua posição na estrutura.
- **double queriesTimeFinish**[**MAXDATA**]: Atributo que mantém as informações do tempo de término da busca por um objeto. Cada posição dessa estrutura diz respeito ao tempo exato que a busca finalizou para o objeto com identificador relativo à sua posição na estrutura. Esse tempo corresponde ao último registro para a *querie* com identificador especificado.
- **double articlesTimeStart**[**MAXDATA**]: Atributo que mantém as informações do tempo de início da instalação de referências de um objeto. Cada posição dessa estrutura diz respeito ao tempo exato que a mensagem de réplica se iniciou. O identificador do objeto corresponde à sua posição na estrutura.

- **double articlesTimeFinish[MAXDATA]**: Atributo que mantém as informações do tempo de término da instalação das referências de um objeto. Cada posição dessa estrutura diz respeito ao tempo exato que a mensagem de réplica finalizou para o objeto com identificador relativo à sua posição na estrutura.
- **vector<double> zones**: Estrutura que mantém os grupos existentes na rede devidamente registrados.
- **unsigned int nmessagesQ**: Armazena o número total de mensagens de buscas realizadas na simulação.
- **unsigned int nmessagesR**: Armazena o número total de mensagens de replicação realizadas na simulação.
- **int minNodes**: Atributo utilizado para controle de fluxo que tem a função definir o número mínimo de nós que podem receber uma mensagem de busca.
- **int maxNodes**: Atributo utilizado para controle de fluxo que tem a função definir o número máximo de nós que podem receber uma mensagem de busca.
- **double percNodes**: Define o percentual máximo de nós que receberão uma determinada mensagem de busca.

Funções auxiliares

- **double toRad(double)**: Transforma o ângulo de graus para radianos. Utilizado no cálculo do atraso de propagação da mensagem. O ângulo (em graus) é passado no parâmetro double.
- **double getDistanceDelay(host*, host*)**: Fornece o atraso no tempo de propagação da mensagem na transição de um nó para outro. Esses nós são parâmetros da função.
- **bool isZoneIn(vector<int>, vector<hosts*>, host*)**: Verifica se já existe um nó, dentre os já escolhidos, que cubra um determinado grupo alvo da mensagem de busca. O primeiro vetor corresponde ao conjunto de grupos que serão cobertos, o segundo vetor são nós já escolhidos e o terceiro parâmetro corresponde ao nó que poderá vir a receber a mensagem.

B.3 Classe *Message*

Descrição: A classe *Message* representa genericamente as mensagens existentes na rede durante a simulação. Os tipos de mensagens variam de acordo com a funcionalidade desejada. Para isso utiliza-se o conceito de herança para estabeceler as classes de mensagens com aplicação específica. É visto mais a frente tipos diferentes de mensagens existentes no simulador.

Atributos

- **Host* to**: Representa o nó, vizinho imediato do nó atual, que deverá receber a mensagem a ser propagada.

Funções Membros

- **virtual unsigned int getType() = 0**: Método responsável por informar o tipo da mensagem a ser processada pelo nó corrente. Esse método é declarado como virtual puro para que as classes derivadas de *Message* implementem as definições que satisfaçam cada tipo de classe.

B.4 Classe *ChangeStateMessage*

Descrição: A classe *ChangeStateMessage* derivada da classe *Message* é responsável por alterar o estado corrente de um nó da rede. Esse estado pode variar nas opções BUSY e IDLE. O estado BUSY se refere ao estado ocupado do nó. Já o estado IDLE se refere ao estado inativo do nó.

Atributos

- **Host* from**: Esse atributo referencia o nó que deverá realizar a mudança de estado.

Funções Membros

- **unsigned int getType()**: Método responsável por informar que o tipo da mensagem é CHANGE_STATE_MESSAGE.

B.5 Classe *JoinMessage*

Descrição: A classe *JoinMessage* derivada da classe *Message* é responsável pela requisição de entrada de um nó na rede. Todo nó que deseja entrar na rede deve enviar uma mensagem deste tipo a um nó já conhecido e pertencente à rede.

Atributos

- **Host* from**: Esse atributo referencia o nó que enviou a mensagem de entrada na rede.

- **int type**: Indica o tipo da mensagem de entrada. O tipo apresenta basicamente três possíveis valores: RING_TYPE, RING_TYPE_CHURN e QUERY_TYPE. O tipo RING_TYPE se refere a entrada do nó no anel estruturado da rede. O tipo RING_TYPE_CHURN se refere a entrada do nó na estrutura de anel da rede em períodos de *churn*. Já o tipo QUERY_TYPE se refere à entrada do nó perante a estrutura de conexões de longo alcance da rede.

Funções Membros

- **unsigned int getType()**: Método responsável por informar que o tipo da mensagem é JOIN_MESSAGE.

B.6 Classe *LeaveMessage*

Descrição: A classe *LeaveMessage*, derivada da classe *Message*, é a responsável por tratar as mensagens referentes a saída de um nó da rede. Ao sair da rede um nó deve estabeler a conexão entre nós vizinhos, de modo a preservar a estrutura do anel.

Atributos

- **Host* from**: Esse atributo referencia o nó que irá fazer a retirada da rede.

Funções Membros

- **unsigned int getType()**: Método responsável por informar que o tipo da mensagem é LEAVE_MESSAGE.

B.7 Classe *QueryMessage*

Descrição: A classe *QueryMessage*, derivada da classe *Message*, é a responsável por tratar as mensagens de buscas propagadas na rede.

Atributos

- **Host* source**: Responsável por armazenar a informação de onde partiu a mensagem de busca (origem da propagação).
- **Host* from**: Responsável por armazenar de que vizinho imediato ao nó corrente é que se originou a mensagem.
- **int data**: Indica o dado procurado pela mensagem de busca.
- **int begin**: Indica o menor indentificador de grupo possível para que a mensagem de busca seja propagada.

- **int end**: Indica o maior indentificador de grupo possível para que a mensagem de busca seja propagada.
- **int id**: Identificador da mensagem de busca.

Funções Membros

- **unsigned int getType()**: Método responsável por informar que o tipo da mensagem é QUERY_MESSAGE.

B.8 Classe *ReferenceMessage*

Descrição: A classe *ReferenceMessage*, derivada da classe *Message*, é a responsável por tratar as mensagens de instalação de réplicas na rede.

Atributos

- **Host* source**: Responsável por armazenar a informação de onde partiu a mensagem de instalaçao de réplica (origem da publicação).
- **Host* from**: Responsável por armazenar de que vizinho imediato ao nó corrente se originou a mensagem de instalação de réplica.
- **int value**: Armazena o valor do dado a ser publicado.

Funções Membros

- **unsigned int getType()**: Método responsável por informar que o tipo da mensagem é REFERENCE_MESSAGE.

B.9 Classe *QueryResultMessage*

Descrição: A classe *QueryResultMessage*, derivada da classe *Message*, é a responsável por averiguar o desempenho do sistema. Esse evento é escalonado para que seja feita a verificação do funcionamento global do sistema, com a avaliação dos dados encontrados durante a busca, número de mensagens inseridas no sistema e o tempo gasto no desempenho das pesquisas.

Atributos

- **int item**: Identifica a mensagem de busca que será avaliada durante o processo de checagem de desempenho.

Funções Membros

- **unsigned int getType()**: Método responsável por informar que o tipo da mensagem é QUERY_RESULT_MESSAGE.

B.10 Classe *UpdateShortcutMessage*

Descrição: A classe *UpdateShortcutMessage*, derivada da classe *Message*, é a responsável por realizar a atualização dos atalhos dos nós presentes na rede. Essa atualização é necessária quando um determinado nó sair da rede e os atalhos ficarem inconsistentes.

Funções Membros

- **unsigned int getType()**: Método responsável por informar que o tipo da mensagem é UPDATE_SHORTCUT_MESSAGE.

B.11 Classe *Connection*

Descrição: A classe *Connection* representa uma conexão existente entre dois *Hosts* na rede.

Atributos

- **Host *owner**: Identifica o nó que possui a conexão.
- **Connection *CW**: Representa a conexão que sucede a conexão corrente.
- **Connection *CCW**: Representa a coxexão que precede a conexão corrente.

B.12 Classe *Event*

Descrição:A classe *Event* representa os eventos do sistema. Qualquer acão a ser executada na rede deve ser alocada no simulador como um evento a ser processado, seja ela entrada, saída, replicação, etc.

Atributos

- **double time**: Tempo global de execução do evento.
- **Message *message**: Mensagem referente ao evento a ser processado. Os tipos de mensagens associadas a esses eventos são descritas na classe *Message*.

Funções Membros

- **bool isNetworkEvent()**: Responsável por verificar se o evento a ser processado faz parte do funcionamento da rede. Se fizer, o evento deve ser processado pela rede, caso contrário, o evento deve ser processado por um *Host*. Essa distinção é necessária para que a rede tenha disponível funcionalidades próprias para manutenção da topologia e obtenção de resultados das simulações.

B.13 Classe *Scheduler*

Descrição: A classe *Scheduler* representa o escalonador de eventos da simulação. Neste simulador são armazenadas todos as ações que serão executadas na rede.

Atributos

- **double clock**: Relógio global de execução. Todos os eventos escalonados utilizam esse relógio como referência para futuros escalonamentos.
- **MinHeap* eventList**: Estrutura de dados responsável por armazenar os eventos de acordo com o tempo de escalonamento fornecido. Essa estrutura funciona como uma Min Heap, em que o próximo evento a ser processado pode ser obtido em $O(1)$ passos.

Funções Membros

- **Event* getNext()**: Retorna uma referência para o próximo evento a ser executado pelo escalonador de eventos.
- **bool hasNext()**: Método que verifica a existência de algum evento na estrutura de dados do escalonador.

B.14 Classe *MinHeap*

Descrição: A classe *MinHeap* representa a estrutura de dados utilizada pelo escalonador de eventos da simulação. Ela representa uma árvore Min Heap.

Atributos

- **int currentSize**: Indica o número corrente de eventos na Heap.
- **int maxSize**: Indica o número máximo de eventos suportado pela estrutura.
- **Event** Heap**: Referência para a estrutura de dados.

Funções Membros

- **Event* insert(Event*)**: Procedimento responsável por inserir um evento na estrutura. O evento a ser inserido é representado pelo parâmetro Event*. O valor de retorno se refere a uma referência para o próximo evento a ser executado.
- **Event* getMin()**: Retorna uma referência para o próximo evento a ser executado.
- **Event* deleteMin()**: Exclui o elemento recém processado.