UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

Uma Arquitetura P2P Baseada na Hierarquia do Endereçamento IP com Roteamento Unificado

Marcos César Madruga Alves Pinheiro

Profa. Thais Vasconcelos Batista, Dr.
Prof. Luiz Affonso Henderson Guedes de Oliveira, Dr.
Orientadores

Natal, RN – Brasil
Janeiro de 2006

UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

Uma Arquitetura P2P Baseada na Hierarquia do Endereçamento IP com Roteamento Unificado

Marcos César Madruga Alves Pinheiro
Orientadores: Profa. Thais Vasconcelos Batista, Dr
Prof. Luiz Affonso Henderson Guedes de Oliveira, Dr

Tese de Doutorado apresentada ao Programa de pós-graduação em Engenharia Elétrica da Universidade Federal do Rio Grande do Norte como parte dos requisitos para a obtenção do titulo de Doutor em Engenharia Elétrica.

Natal, RN – Brasil
Janeiro de 2006

Universidade Federal do Rio Grande do Norte

Banca examinadora

Profa. Thais Vasconcelos Batista, Dr. – DIMAp/UFRN

Prof. Luiz Affonso Henderson Guedes de Oliveira, Dr. – DCA/UFRN

Prof. Ivan Saraiva Silva, Dr. – DIMAp/UFRN

Prof. Bruno Schulze, Dr. - LNCC/RJ

Prof. Francisco Vilar Brasileiro, Dr. – DSC/UFCG

Para minha esposa Renata e meu filho Matheus

Agradecimentos

A minha esposa Renata, pelo companheirismo, incentivo, apoio, e por ter dividido comigo todos os momentos dessa longa caminhada.

Ao meu filho Matheus, pelos seus beijos, sorrisos e brincadeiras que tornam os meus dias tão felizes.

Aos meus pais por sempre terem me permitido escolher o caminho a seguir.

A minha orientadora Thais, pelo incentivo, confiança, atenção, disponibilidade e pelo otimismo constante.

Ao meu co-orientador Affonso, por ter estado sempre presente e me apoiado em todas as etapas desse trabalho.

Ao amigo Rodrigo, pela valiosa contribuição nas implementações contidas nesse trabalho.

Aos amigos Amador, Cláudio, Kalil e Vergara, pelo incentivo e apoio no ambiente de trabalho.

A Universidade Federal do Rio Grande do Norte, por proporcionar mais essa etapa na minha formação.

A Deus, por sempre guiar os meus passos para a realização dos meus sonhos.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

Entre as diversas abordagens para se aproveitar os recursos computacionais ociosos existentes nas folhas da Internet, ou seja, nas máquinas dos usuários, as redes *peer-to-peer* (P2P) vêm ganhando destaque especial nos últimos anos devido principalmente à sua escalabilidade, desempenho e tolerância à falhas.

As arquiteturas P2P atuais, entretanto, ainda apresentam alguns problemas como a sobrecarga nos nós devido à realização do roteamento de mensagens, o número elevado de nós reconfigurados devido à mudanças de topologia da rede, a existência de tráfego de roteamento dentro das redes das organizações que não é destinado a nenhuma de suas máquinas e à ausência de relação entre a proximidade dos nós na rede P2P e a proximidade desses nós na rede IP. Embora algumas arquiteturas considerem essas distâncias na rede IP, o fazem através de métodos que requerem a troca de informações constantemente.

Nesse trabalho nós propomos uma arquitetura P2P para resolver os problemas citados. Essa arquitetura é composta por três partes. A primeira parte consiste em uma arquitetura P2P básica, chamada SGrid, que mantém a relação dos nós na rede P2P com suas posições na rede IP e atribui regiões de chaves adjacentes para nós de uma mesma organização. A segunda parte consiste em um protocolo chamado NATal (*Routing and **NAT a**pplication **l**ayer)* que estende a arquitetura básica para retirar dos nós a função de roteamento de mensagens. A terceira parte consiste de um tipo especial de nó, chamado LSP (*Lightware Super-Peer*), que é o responsável pela manutenção das tabelas de roteamento P2P.

Além da descrição da arquitetura proposta e da especificação dos protocolos SGrid e NATal, esse trabalho apresenta o simulador desenvolvido para validar a arquitetura e um módulo para ser utilizado em roteadores Linux que implementa o protocolo NATal.

Palavras-chave: Sistemas Distribuídos, Redes *Peer-to-Peer*, *Super-peers* e redes *overlay*.

Abstract

There are some approaches that take advantage of unused computational resources in the Internet nodes – users´ machines. In the last years , the peer-to-peer networks (P2P) have gaining a momentum mainly due to its support for scalability and fault tolerance.

However, current P2P architectures present some problems such as nodes overhead due to messages routing, a great amount of nodes reconfigurations when the network topology changes, routing traffic inside a specific network even when the traffic is not directed to a machine of this network, and the lack of a proximity relationship among the P2P nodes and the proximity of these nodes in the IP network. Although some architectures use the information about the nodes distance in the IP network, they use methods that require dynamic information.

In this work we propose a P2P architecture to fix the problems afore mentioned. It is composed of three parts. The first part consists of a basic P2P architecture, called SGrid, which maintains a relationship of nodes in the P2P network with their position in the IP network. Its assigns adjacent key regions to nodes of a same organization. The second part is a protocol called NATal (*Routing and **NAT** **a**pplication **l**ayer*) that extends the basic architecture in order to remove from the nodes the responsibility of routing messages. The third part consists of a special kind of node, called **LSP** (*Lightware Super-Peer*), which is responsible for maintaining the P2P routing table.

In addition, this work also presents a simulator that validates the architecture and a module of the NATal protocol to be used in Linux routers.

Keywords: *Distributed Systems*, *Peer-to-Peers Networks*, *Super-peers*, and *Overlay Networks*.

1 Introdução

O modelo cliente-servidor tradicional, onde uma única máquina, o servidor, é encarregada de receber requisições de diversas outras (os clientes), realizar as operações solicitadas e fornecer os resultados, não é adequado ao novo cenário da computação, pois os clientes rompem as fronteiras da empresa e não estão restritos nem em número nem em localização ao ambiente físico da mesma.

Entretanto, os avanços da informática tanto na área de hardware como de telecomunicações não trouxeram mudanças apenas para os fornecedores de serviços, revolucionaram também o ambiente do usuário doméstico. Nesse novo mundo, as máquinas *desktop* possuem não só espaço de armazenamento disponível, como também uma alta capacidade de processamento, que é subutilizada na maior parte do tempo [1]. Além disso, as conexões à Internet, que até pouco tempo eram, em sua maioria, conexões temporárias, realizadas através de linhas telefônicas discadas, estão se tornando conexões permanentes, através da utilização de tecnologias como redes sem fio e "cabo". Somado a esse universo de máquinas domésticas, existem também, as máquinas das empresas que se enquadram na mesma situação, mostrando que existem recursos computacionais disponíveis e conexões de rede para acessá-los. Porém, para que esses recursos possam ser efetivamente utilizados, faz-se necessário um software que gerencie todo o processo.

Esse novo cenário tem impulsionado fortemente as pesquisas em sistemas distribuídos, o que acabou levando ao surgimento de novas tecnologias que, embora possam ser agrupadas sob o termo comum de "sistemas distribuídos", uma vez que envolvem a utilização e gerência de recursos em diversas máquinas, têm objetivos e características relativamente distintos. Podemos classificar esses sistemas como *Clusters* [2], Grades Computacionais [3] [4] e redes *Peer-to-Peer (P2P)* [9]. Embora os dois primeiros tenham como objetivo principal dar suporte à execução de programas complexos que requerem muito poder de processamento, os *clusters* são tipicamente constituídos por máquinas de poder computacional semelhante e que estão conectadas por uma rede de alta velocidade. Além disso, normalmente utilizam o mesmo ambiente operacional e estão restritas a um ambiente físico limitado, como o de uma empresa, por exemplo. As Grades, por outro lado, não estão restritas a um mesmo ambiente físico

nem a máquinas de configuração semelhantes, sendo compostas de máquinas dispersas geograficamente e que utilizam qualquer ambiente computacional. Um dos projetos recentes em Grades Computacionais é o SETI@HOME [5] que tem por objetivo utilizar processamento de máquinas de usuários espalhados pela Internet para analisar dados recebidos do telescópio de rádio de Arecibo, localizado em Porto Rico, em busca de evidências de transmissões de rádio de inteligência extraterrestre, e conta com mais de 5 milhões de participantes. Entre os vários outros projetos nessa área, podemos citar o Globus [6], o OurGrid [7], e o Grid2003 [8].

Embora a diferença entre *Cluster* e Computação em Grade seja clara, a linha que separa esse último das recentes redes *peer-to-peer* é mais tênue. Essas redes são compostas de máquinas *desktop* espalhadas pela Internet, onde não existe um controle centralizado e todas desempenham aproximadamente as mesmas tarefas, permitindo que cada máquina seja tanto um cliente quanto um servidor. Existe também um dinamismo muito grande nos componentes da rede, ou seja, os nós entram e saem da rede constantemente. Uma das maiores diferenças em relação às grades computacionais é que, enquanto essas últimas têm como objetivo resolver problemas complexos, que normalmente envolvem muito poder de processamento, as redes *peer-to-peer* visam principalmente resolver problemas de complexidade mais simples, porém que sejam solicitados por um número muito grande de usuários, como é o caso das redes *peer-to-peer* para compartilhamento de arquivos.

As redes *peer-to-peer* são também chamadas de redes *overlay* [10], uma vez que constituem uma nova rede lógica sobre uma rede existente. Essa característica facilita a criação e disseminação de diferentes redes *peer-to-peer* uma vez que essas redes são compostas apenas por software instalado nas máquinas dos usuários, sem necessitar de equipamentos ou configurações especiais na infra-estrutura de rede. Dessa forma é possível que máquinas em um mesmo ambiente (ou até mesmo, uma mesma máquina) participem de diferentes redes *peer-to-peer*.

Apesar de existirem inúmeras redes *peer-to-peer* diferentes, a maior parte delas compartilha uma série de características em comum, entre as quais podemos citar:

- Balanceamento de carga: os dados a serem armazenados na rede e/ou os processos a serem executados, devem ser distribuídos de forma razoavelmente homogênea entre todos os nós da rede. Ou seja, cada nó deve receber aproximadamente a mesma quantidade de dados para armazenar e/ou processamento para executar.

- Escalabilidade: A complexidade com que as tarefas relativas à operação (pesquisas, atualização de nós, etc..) são executadas deve crescer apenas logaritmicamente com o número de nós na rede, de modo que a mesma suporte a adição de um número potencialmente muito grande de novos nós. Além disso, nenhuma intervenção manual de configuração deve ser requerida para que a rede se ajuste a um número elevado de nós. Esse processo deve ocorrer de forma automática e transparente.
- Descentralização: Todos os nós devem desempenhar tarefas semelhantes. Existe, entretanto, um tipo de rede *peer-to-peer* onde alguns nós, chamados *super-peers* [11], desempenham tarefas específicas. Embora a existência desses nós implique em uma certa centralização, a rede como um todo permanece descentralizada pois, normalmente, existe um número grande de *super-peers*.
- Tolerância a falhas: Em redes compostas por um número muito elevado de máquinas é natural que algumas delas possam falhar ou deixar a rede de forma inesperada. Portanto, as redes devem possuir um mecanismo de recuperação de nós em caso de falha e conseguir realizar suas operações normalmente, ainda que isso implique numa perda de desempenho temporária.

Diversas aplicações distribuídas como, por exemplo, sistemas de armazenamento redundante [45], sistemas para garantir o anonimato do usuário [38], sistemas de pesquisa [68], sistemas de processamento distribuído [43], entre outras, podem ser construídas utilizando como base uma rede *peer-to-peer*. Entre as operações fornecidas por uma rede *peer-to-peer*, uma em especial tem um papel fundamental para essas aplicações de alto nível: a operação de busca, onde dada uma chave é retornado o nó (*peer*) responsável por ela. Isso se deve ao fato que todas elas precisam identificar os nós da rede associados a algum dado em particular.

O primeiro sistema a conseguir mostrar o poder das redes *peer-to-peer* foi o Napster [12] [13], em 1999, que permitia aos usuários compartilharem arquivos de música no formato *mp3*. Nesse sistema a operação de localização dos dados, no caso os computadores que possuíam as músicas desejadas, era feita através da consulta a um servidor central que mantinha um índice das músicas disponíveis nos computadores de cada usuário. Como todos os usuários registravam-se nesse servidor central, pesquisas complexas, como, por exemplo, por palavras-chave, podiam ser realizadas facilmente.

Assim sendo, apesar do acesso aos dados, no caso, o *download* das músicas, ser feito de forma distribuída, a localização do dado baseava-se em uma estrutura centralizada. A despeito dos problemas legais sofridos pelo Napster, relativos a direitos autorais, uma solução centralizada não é adequada para sistemas com a abrangência da Internet. Dessa forma, as redes *peer-to-peer* que surgiram posteriormente buscaram atingir um nível completo de distribuição tanto em relação ao acesso aos dados quanto à localização dos mesmos.

Após a criação de algumas novas tecnologias de redes *peer-to-peer* que se seguiram ao Napster e que se baseavam em propagar as consultas de forma indiscriminada por vários nós da rede, tentando atingir o nó onde os dados se encontravam, como é o caso do Gnutella [14], uma técnica recente, chamada DHT (*Distributed Hash Table*) [15], vem sendo bastante utilizada e deu origem a uma geração de redes *peer-to-peer* onde pode-se incluir as redes Chord [16], CAN [17], Pastry [18] e Tapestry [19], por exemplo. Em redes DHT, identificadores, normalmente calculados utilizando funções *hash*, são associados tanto aos nós quanto aos dados a serem armazenados na rede. Cada nó mantém apontadores para um pequeno número de outros nós da rede. Os dados são armazenados no nó que possui o identificador mais próximo do identificador do dado. Quando uma consulta por um dado vai ser realizada, primeiro o identificador associado ao dado é calculado e a consulta é passada de nó em nó. A cada passo, o nó que recebe a consulta tem um identificador mais próximo ao identificador pesquisado do que o nó anterior. A consulta é finalizada quando atinge-se o nó da rede que possui o identificador mais próximo ao identificador pesquisado.

1.1 Motivação

Como as redes *peer-to-peer* são utilizadas no contexto da Internet, e são constituídas, conseqüentemente, de milhares, ou até mesmo milhões de máquinas, devem existir parâmetros que permitam a qualquer novo modelo de rede mensurar a sua escalabilidade e desempenho. Vários trabalhos ressaltam diferentes aspectos a serem analisados [20] [21] [22], entre os quais podemos destacar três. O primeiro se refere ao número de nós consultados para a realização de uma busca, que deve ser no máximo O(Log N), onde N é o número de nós na rede. O segundo diz respeito à quantidade de informações sobre outros nós da rede que deve ser mantida em cada nó. Tipicamente cada nó deve manter apontadores para, no máximo, O(Log N) outros nós. E, finalmente,

o terceiro aspecto refere-se ao número de nós que precisam ser reconfigurados quando um novo nó entra na rede ou algum nó sai do sistema. Um valor aceitável para esse parâmetro seria que o mesmo fosse mantido abaixo de $O(\text{Log}^2 N)$.

Existe, entretanto, uma característica que pode mascarar problemas de desempenho, mesmo sendo mantidos os parâmetros citados no parágrafo anterior e que as redes mais recentes têm tentado incorporar, que diz respeito à distribuição ou não dos nós na rede *peer–to-peer* levando-se em conta as distâncias de rede física existentes entre os nós. Ou seja, como as redes *peer-to-peer* são redes *overlay*, o fato de dois nós serem vizinhos na rede *peer-to-peer* não garante que os mesmos estejam próximos em termos de distâncias de rede física. Dessa forma, uma busca que passe por apenas *n* nós numa rede *peer-to-peer* pode na verdade, passar por *m* nós na rede física (com *m* muito maior que *n*). Essa característica está relacionada ao tema que é comumente referenciado na literatura como "ciência da localização" [55].

Embora soluções distribuídas, em geral, melhorem a tolerância a falhas e a confiabilidade dos sistemas, diferentes tarefas de um sistema podem precisar de níveis de distribuição diferentes. As redes *overlay* possuem duas principais tarefas bem definidas, que poderiam ser tratadas com níveis diferentes de distribuição. A primeira dessas tarefas consiste em fazer com que cada nó responsabilize-se por um conjunto de chaves e responda às pesquisas endereçadas a cada uma dessas chaves. Como essa tarefa está relacionada diretamente com as aplicações, pois determina, por exemplo, onde os dados da aplicação são armazenados, é interessante que seja utilizado um modelo altamente distribuído para fazer com que todos os nós participem da rede diretamente, aumentando, assim, a quantidade de recursos disponíveis. A segunda tarefa de uma rede *overlay* consiste em fazer com que cada mensagem chegue ao nó de destino, ou seja, a rede deve prover um mecanismo para roteamento de mensagens. Como essa tarefa é de vital importância tanto para o desempenho quanto para a confiabilidade da rede, um modelo com um nível menor de distribuição em relação ao da tarefa anterior é mais adequado, onde um número menor de nós participa da tarefa de roteamento, tipicamente nós mais robustos e estáveis.

As arquiteturas atuais, no entanto, não fazem a separação entre o nível de distribuição utilizado nessas duas tarefas, aplicando uma única abordagem para ambas e, conseqüentemente, prejudicando a realização de uma delas. Desse modo, as arquiteturas que reduzem o nível de distribuição, limitando o número de máquinas que participam efetivamente da rede, fazem com que máquinas com bons recursos

computacionais deixem de ser aproveitadas. Por outro lado, a abordagem que é utilizada pela maioria das arquiteturas emprega um alto nível de distribuição, onde todos os nós participam da rede e realizam as mesmas tarefas, acarretando problemas ainda maiores.

O primeiro problema refere-se ao fato de que as redes P2P são formadas por nós heterogêneos, onde todos são encarregados de realizar roteamento de mensagens, mesmo máquinas com capacidades limitadas de processamento e banda de rede que degradam o desempenho da rede. Além disso, muitas dessas maquinas podem deixar a rede de forma abrupta seja por falta de energia, travamento ou perda de conectividade. Embora as arquiteturas possuam mecanismos para recuperação dessas situações, tais mecanismos geram tráfego adicional, requerem processamento extra e não evitam que parte da rede fique instável por um certo período, ainda que um período limitado.

O segundo problema está relacionado com tolerância a sabotagem. As situações discutidas no parágrafo anterior são involuntárias, decorrentes da própria utilização da rede. O modelo de roteamento atual, entretanto, permite que usuários mal intencionados realizem o roteamento de mensagens de forma incorreta, fazendo com que as mesmas demorem mais para atingir o seu destino, e propagem informações de roteamento também incorretas, comprometendo as tabelas de rotas de outros nós e prejudicando o funcionamento da rede.

O terceiro problema está associado ao mecanismo de roteamento de mensagens nas redes P2P. Como a maioria dos nós que constituem as redes P2P está localizada nas redes internas das organizações, para que esses nós possam realizar o roteamento, as mensagens devem entrar nessas redes e posteriormente retornarem à Internet. Esse tráfego de roteamento circulando por dentro das redes das organizações, mas que não é destinado a nenhuma de suas máquinas, gera uma utilização desnecessária dos *links* de comunicação de cada organização.

O quarto problema também está relacionado com o mecanismo de roteamento, mais precisamente com a manutenção das tabelas de roteamento. Uma vez que cada nó em uma rede P2P mantém uma tabela de roteamento com apontadores para outros nós na rede, sempre que um novo nó entra ou algum nó deixa a rede, diversos outros nós precisam ter suas tabelas modificadas. Portanto, utilizar uma arquitetura onde todos os nós realizam roteamento e, conseqüentemente, precisam estar com suas tabelas de roteamento atualizadas, gera um aumento considerável no número de nós que precisam ser reconfigurados após mudanças de topologia na rede. Isso é especialmente importante

para as redes P2P uma vez que, são compostas por milhares, ou até mesmo milhões de nós, e têm sua topologia modificada constantemente.

1.2 Objetivos do Trabalho

Diante da relevância do tema das redes P2P, esse trabalho tem como objetivo principal propor uma nova arquitetura de rede P2P que procura resolver os problemas discutidos anteriormente relacionados às arquiteturas atuais.

A arquitetura especificada nesse trabalho, chamada SGrid (*Symmetric Grid*), propõe um método, baseado na hierarquia do endereçamento IP, para a definição da topologia da rede P2P de uma forma a considerar a distância entre os nós na rede física. Desse modo, nós que estão próximos na rede P2P também estão próximos na rede física. Além disso, o método utilizado para a definição da topologia não requer a troca de informações entre os nós.

Como a maior parte dos problemas identificados nas redes P2P são decorrentes da adoção do mesmo nível de distribuição para todas as tarefas de uma rede *overlay*, nesse trabalho é apresentada uma proposta baseada em dois protocolos: (1) um protocolo para manter a infra-estrutura de roteamento da rede *overlay*, chamado NATal (*Routing and **NAT** **a**pplication **l**ayer*); (2) outro protocolo, chamado SGrid, mais direcionado para a realização das tarefas básicas da rede *overlay*, como, por exemplo, alocação das chaves aos nós da rede. Cada um desses protocolos utiliza um nível de distribuição diferente.

Enquanto o protocolo SGrid deve permitir ao sistema desempenhar todas as tarefas de uma rede P2P, o protocolo NATal deve liberar os nós convencionais da realização de roteamento de mensagens, criando assim dois grupos de nós diferenciados, e permitindo que a rede lide melhor com o problema da heterogeneidade dos recursos computacionais das máquinas. Os nós do primeiro grupo são responsáveis por manterem a estrutura da rede P2P e, portanto, realizarem o roteamento das mensagens. O segundo grupo é composto por nós que são encarregados apenas de realizarem tarefas relacionadas à utilização da estrutura da rede P2P, como por exemplo, responderem as pesquisas endereçadas às chaves pelas quais estão responsáveis. Os nós que realizam roteamento devem resolver o problema de engenharia de tráfego citado anteriormente, repassando para dentro das redes apenas as mensagens destinadas a algum de seus nós.

O protocolo NATal deve ainda permitir que todos os nós de uma mesma rede (de uma empresa, por exemplo) sejam vistos na rede P2P global como um único nó, de modo a reduzir consideravelmente o número de mudanças de topologia, e conseqüentemente de nós reconfigurados, decorrentes da entrada ou saída de nós na rede. Para isso é definido um tipo especial de nó, baseado no conceito de *super-peers*, chamado LSP (*Ligthware Super-peer*), que é o único nó de cada rede particular conhecido na rede P2P global. Embora esse nó apareça para os outros nós da rede global como sendo responsável pelas chaves de todos os nós de uma determinada rede a qual pertence, o LSP apenas recebe as mensagens referentes às chaves pelas quais realmente é responsável (por isso o termo *lightware*).

Além da especificação dos protocolos SGrid e NATal o presente trabalho apresenta a implementação de um simulador da arquitetura incluindo o protocolo SGrid e a implementação do protocolo NATal como um módulo para o sistema operacional Linux, que permite a análise dessa arquitetura em uma rede real.

Em suma, os principais objetivos desse trabalho são:

- Proposição de uma arquitetura de rede P2P que se diferencia das arquiteturas existentes nos seguintes aspectos:
    - Utilização de um mecanismo simples e eficiente, que não requeira troca de informações dinâmicas, para considerar as distâncias entre dois nós nas redes IP de modo que esses nós estejam próximos na rede P2P.
    - Redução significativa das mudanças de topologia na rede P2P.
    - Eliminação do tráfego de dados que passa por dentro das redes das organizações, mas que não é destinado a nenhuma de suas máquinas.
    - Redução significativa do processamento realizado por cada nó da rede, e, conseqüentemente, das exigências necessárias em termos de recursos computacionais que um nó precisa atender para participar eficientemente da rede P2P.
    - Elevação do nível de confiabilidade da rede, uma vez que nós comuns não podem (ou dificilmente conseguirão) prejudicar o desempenho da rede como um todo.
    - Utilização de uma estrutura em grade com níveis hierárquicos, que proporciona um ambiente propício à construção de

protocolos de busca utilizando técnicas que podem se beneficiar dessa estrutura, como, por exemplo, *bloom filters* [53] e árvores *quadtree* [62].

- Especificação dos protocolos NATal e SGrid.
- Desenvolvimento do protótipo da arquitetura e do protocolo SGrid que permita simulações para análise do comportamento do modelo proposto.
- Desenvolvimento de um módulo para o kernel do Linux, utilizando a arquitetura do NetFilter, que implementa o protocolo NATal e permite, portanto, que máquinas com esse sistema operacional sejam utilizadas como roteadores NATal.

1.3 Estrutura do Trabalho

O trabalho está estruturado da seguinte forma:

- No Capítulo 2 é apresentada uma discussão sobre os modelos de redes P2P existentes e as características comuns à maior parte delas. Esse capítulo também apresenta as principais representantes das redes P2P.
- No Capítulo 3 é apresentado o modelo básico da rede P2P proposta, sem a utilização de *super-peers*, sendo analisadas várias questões relacionadas às medidas de desempenho da rede.
- O Capítulo 4 apresenta o modelo completo incluindo a descrição do protocolo NATal e a discussão de como os *super-peers* e os roteadores NATal são incorporados ao modelo básico.
- O Capítulo 5 apresenta a especificação dos protocolos SGrid e NATal, onde são descritos todos os formatos das mensagens.
- O Capítulo 6 apresenta os protótipos desenvolvidos, onde o comportamento da rede P2P pode ser analisado. São apresentados o simulador, desenvolvido na linguagem Java, onde é possível criar redes com um número elevado de nós, e um módulo para o kernel do Linux, desenvolvido em C, que permite a análise da arquitetura em um ambiente real.
- O Capítulo 7 contém uma análise comparativa do presente trabalho com trabalhos relacionados que foram descritos no Capitulo 2.
- O Capítulo 8 apresenta as principais conclusões desse trabalho e possíveis trabalhos futuros.

2 Redes *Peer-to-Peer*

Diante do grande interesse surgido a respeito das redes *peer-to-peer* nos últimos anos, as pesquisas sobre esse tema têm integrado profissionais de diversas áreas da computação, como banco de dados, redes, sistemas distribuídos, teoria dos grafos, sistemas de agentes, entre outros. Essa integração tem proporcionado uma evolução significativa nas tecnologias relacionadas às redes P2P, bem como, acarretado o surgimento de diversas novas arquiteturas e propostas de utilização desses sistemas. Porém, como efeito colateral desse processo ainda existe uma certa inconsistência nas nomenclaturas e modelos utilizados que dificultam a compreensão e comparação entre essas tecnologias.

Esse capítulo, apresenta uma definição de redes P2P juntamente com as características comuns às diversas redes existentes e um esquema de classificação das mesmas baseado no modelo proposto em [27].

2.1 Definição

Entre as muitas definições para redes *peer-to-peer* (P2P) encontradas na literatura, Oram [9] caracteriza uma rede P2P como uma classe de aplicações que tiram vantagem do armazenamento de recursos, ciclos de CPU, conteúdo e presença humana disponíveis nas "folhas" da Internet. Ressaltando ainda que: como o acesso a esses recursos descentralizados significa operar em um ambiente de conectividade instável e endereços IPs imprevisíveis, os nós P2P devem operar fora do domínio do DNS [23] e ter significante, ou total autonomia de servidores centrais.

Como complemento da definição acima, várias outras características são necessárias para que um sistema possa ser classificado como P2P [24]:

- A rede deve ser composta por nós folha;
- Os nós entram e saem da rede constantemente;
- Os endereços dos nós podem variar, uma vez que saiam da rede;
- Os nós da rede podem estar conectados com taxas de transmissão diferentes;
- Os nós devem ter autonomia parcial ou total em relação a um servidor centralizado;

- Os nós devem tanto consumir recursos da rede quanto disponibilizar outros recursos;
- A rede deve ser altamente escalável;
- Os nós podem comunicar-se diretamente uns com os outros sem a intervenção de um servidor central.

Os nós de uma rede P2P podem desempenhar funções de cliente, de servidor, ou ambas, embora o comportamento esperado seja que todos os nós funcionem como cliente e servidor. Além disso, cada nó tem autonomia tanto para decidir quais recursos compartilhar quanto para impor restrições de acesso aos mesmos. Entre os recursos que podem ser compartilhados podemos citar: arquivos, banda de rede, espaço de armazenamento e ciclos de processador.

Mesmo com as restrições citadas anteriormente, existe um número grande de sistemas que podem ser classificados como P2P. Porém, um tipo especial de sistema P2P, chamado *overlay network*, tem sido bastante utilizada no suporte a localização de serviços e vem ganhando destaque especial na comunidade científica. Uma rede *overlay* [25] é uma rede virtual criada sobre uma rede existente, de modo que a própria Internet, com o protocolo IP [26], se enquadra nessa classificação, uma vez que esconde as topologias reais das redes físicas (Ethernet, ATM, Frame Relay, entre outras) sobre as quais é criada. O objetivo de uma rede *overlay* é criar uma estrutura de nível mais elevado que seja mais adequada à solução de determinados problemas do que a estrutura fornecida pela manipulação direta dos componentes da rede física existente. Como aplicações diferentes precisam de requisitos diferentes, qualquer que seja a configuração da rede física, irá beneficiar um conjunto de aplicações em detrimento de outras. As redes *overlay* amenizam esse problema permitindo que cada aplicação tenha a sua própria visão da rede, ou seja, crie a estrutura de rede que lhe é mais adequada. Dessa forma, uma rede P2P pode ser considerada uma rede *overlay*.

2.2 Classificação dos Sistemas P2P

Os sistemas P2P podem ser divididos em três categorias [27], conforme ilustrado pela figura 2.1, onde cada um deles possui objetivos e características semelhantes.

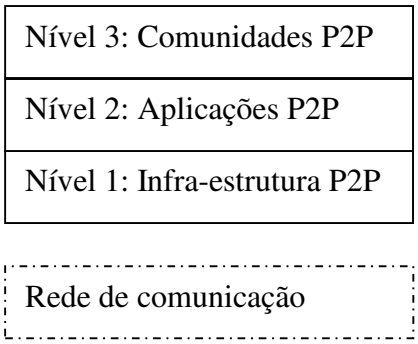


Figura 2-1. Classificação dos sistemas P2P.

2.2.1 Nível 1: Infra-estrutura P2P

Esse nível está posicionado imediatamente acima da estrutura de rede existente e fornece os serviços básicos para a construção dos níveis superiores. Entre os serviços oferecidos estão a identificação, localização e comunicação com os outros nós (*peers)* da rede. Além de suportar, em geral, mecanismos de segurança com autorização e autenticação. Portanto, é nessa categoria que se enquadram as redes *overlay*. A principal função dessa camada é fornecer interoperabilidade entre os diversos sistemas P2P existentes e para isso são necessárias APIs padronizadas que possam ser utilizadas por qualquer aplicação. Entre as iniciativas existentes nesse sentido, podemos citar:

- *JXTA* [28]: Uma plataforma aberta que define um conjunto de protocolos padronizados, baseados em XML [30], através dos quais quaisquer dispositivos conectados à rede podem se comunicar entre si. É definido por um modelo de três camadas (kernel, serviços e aplicações) e inclui um protocolo específico para localização de serviços, o JXTA *Search* [29].
- Em [31] é apresentada uma API, desenvolvida em C, específica para as redes Chord, CAN, Pastry e Tapestry. O ponto desfavorável desse trabalho é possuir uma aplicabilidade restrita apenas a um subconjunto específico de redes *peer-to-peer*.
- *Web Services* [32]: Embora não se tratem especificamente de redes *peer-to-peer* podem ser utilizados como base para a construção das mesmas. As tecnologias que compõem os *web services* são:

- XML (*Extensible Markup Language*) – linguagem utilizada para a definição dos formatos de documentos.
- SOAP (*Simple Object Access Protocol*) - protocolo de comunicação.
- *WSDL (Web Services Description Language)* – Linguagem utilizada para a descrição dos serviços.
- WSFL (*Web Services Flow Language*) – linguagem utilizada para a descrição dos fluxos de trabalho.
- UDDI (*Universal Description, Discovery and Integration*) – diretório utilizado para a publicação e localização dos serviços.

- *Groove* [33]: Plataforma que fornece os serviços necessários (como armazenamento, sincronização e segurança) para a construção de aplicações P2P, onde podemos incluir, como sendo desenvolvida usando essa plataforma, o *Groupware Groove Workspace* [34].

2.2.2 Nível 2: Aplicações P2P

Este nível é formado por aplicações utilizadas diretamente pelos usuários, entre as quais podemos citar: mensagem instantânea, compartilhamento de arquivos, computação distribuída e trabalho colaborativo.

- Mensagem instantânea: Estão atualmente entre as aplicações mais utilizadas na Internet, devido à sua característica de permitir que as pessoas se comuniquem em tempo real. Diferentemente do e-mail, onde as mensagens são armazenadas para serem entregues posteriormente ao receptor, uma vez que não existe importância no fato deste estar ou não conectado, nas aplicações de mensagem instantânea é possível verificar se um usuário está conectado em um dado instante e iniciar uma comunicação interativa com ele. Entre as aplicações mais utilizadas nessa categoria, podemos citar: AOL *Instant Messenger* [35], MSN *Messenger* [36] e o *Yahoo! Messenger* [37].

- Compartilhamento de arquivos: Essas aplicações foram uma das grandes responsáveis pelo sucesso atual das redes P2P, uma vez que permitem a troca de arquivos de forma anônima e eficiente entre qualquer usuário conectado na Internet. Caracterizam-se por manterem os arquivos distribuídos entre diversas máquinas da rede, viabilizando, assim, a criação de áreas de armazenamento potencialmente ilimitadas, como é o caso do Freenet [38]. Como, normalmente, utiliza-se replicação dos dados através do armazenando de cada arquivo em diversos locais, para garantir tolerância a falhas, inerentemente agrega-se uma melhora de desempenho em termos de velocidade de acesso, uma vez que os *downloads* podem ser realizados de locais diferentes. Além do FreeNet, citado anteriormente, outros sistemas de compartilhamento de arquivos amplamente utilizados são o Napster [13], o Gnutella [14] e o KaZaa [39] [40].
- Computação distribuída. Utilizar o poder computacional ocioso disponível em máquinas espalhadas pela Internet é um objetivo antigo que vem ganhando novas perspectivas com a melhorar das conexões de rede existentes. Algumas das iniciativas mais relevantes nessa área são o projeto Beowulf [41] da NASA, o distributed.net [42], que quebrou o algoritmo RSA, o OurGrid [7] e o SETI@home [43] que auxilia na busca por inteligência extra-terrestre.
- Trabalhos colaborativos: Embora essa categoria tenha surgido nos ambientes de redes locais com o nome de *groupware*, após o surgimento das redes P2P esses sistemas expandiram suas capacidades permitindo a integração de usuários independentemente de sua localização. O objetivo desses sistemas é permitir a comunicação e colaboração, como a edição compartilhada de documentos, por exemplo, entre os usuários do sistema. Para isso integra diversas aplicações, como e-mail, calendário, listas de discussão, sistemas de gerenciamento de documentos, vídeo-conferência, entre outras. Um dos membros mais conhecidos dessa categoria é o Groove [33].

Evidentemente há outros tipos de aplicações, além dos quatro tipos citados anteriormente, como jogos em rede, por exemplo, que utilizam a tecnologia P2P.

Embora esse modelo de classificação das aplicações P2P seja muito utilizado na literatura, evidentemente existem aplicações que podem se enquadrar em mais de uma categoria. Dessa forma, alguns autores [27] preferem analisar as aplicações P2P sobre os recursos que são compartilhados, tipicamente: informações, arquivos, banda de rede, espaço de armazenamento e ciclos de processador.

2.2.3 Nível 3: Comunidades P2P

Uma comunidade, ou comunidade virtual, é um termo comumente utilizado para representar indivíduos, que podem estar geograficamente dispersos, mas possuem interesses comuns e utilizam a tecnologia como ferramenta de integração. As redes P2P, devido à sua natureza distribuída, são bastante adequadas para a criação de comunidades, onde podemos destacar as comunidades formadas para troca de arquivos, que utilizam como base as redes *overlay* e protocolos existentes, como o Gnutella [14]. Um outro exemplo de comunidades P2P são as redes @HOME, como seti@home [44], que procura inteligência extraterrestre, e fightaids@home [46], que tem como objetivo ajudar na descoberta da cura da AIDS. Vemos, portanto, que o termo comunidade, refere-se mais a uma forma de usar a tecnologia do que a criação de uma nova tecnologia.

2.3 Redes Overlay

As principais aplicações P2P utilizam redes *overlay* para fornecerem seus serviços. Essas redes fornecem uma estrutura para armazenamento de dados de forma distribuída e mecanismos de busca para a localização das informações desejadas, mantendo o princípio das redes P2P onde os nós entram e deixam a rede constantemente. A operação de uma rede *overlay* pode ser dividida em quatro etapas: entrada, consulta, acesso aos dados e saída. Ao entrar na rede um nó deve se comunicar com outros nós existentes para que tomem conhecimento do novo nó e, após isso, deve publicar as informações que irá disponibilizar aos outros usuários. Essa descoberta de outros nós existentes na rede pode ser feita através de mecanismos como *multicast* ou através de algum serviço de nomes, por exemplo. Embora a exata finalidade para a qual a rede será utilizada dependa da aplicação, essa tarefa de alto nível precisará identificar o nó da rede responsável por um determinado dado. Portanto, após entrar na rede, um nó

normalmente emite uma consulta para algum outro nó da rede. Essa consulta é, então, propagada por vários nós através de um protocolo de roteamento, até chegar ao seu destino. Quando a consulta é bem sucedida, o valor retornado será alguma identificação do nó onde a consulta foi satisfeita (normalmente o seu endereço IP). Nesse ponto, o nó solicitante pode obter os dados diretamente do nó destino. Finalmente, quando um nó não desejar mais utilizar a rede, ele deverá anunciar a sua retirada. Entretanto, como alguns nós podem deixar a rede de forma abrupta, como em casos de falta de energia, perda de conectividade de rede, travamento de máquina, entre outros, as redes P2P precisam de mecanismos que detectem a ausência desses nós e reconfigurem suas estruturas de modo a refletir a nova topologia.

2.3.1 Arquitetura

Diante da diversidade de redes P2P existentes, é importante identificar quais características são comuns a todas elas e quais as opções de projeto são mais comumente adotadas. Desse modo, a existência de um modelo de referência com essa finalidade, conforme proposto em [10], e que servirá como base para a discussão apresentada nessa seção, facilita a compreensão e a comparação entre diferentes arquiteturas de redes.

Uma rede *overlay* é formada por um conjunto de nós que fornecem acesso a um conjunto de recursos. Para determinar qual nó será encarregado de qual recurso, ambos, os nós e os recursos, são mapeados para um espaço de identificadores, sendo que esses mapeamentos podem ser realizados com a mesma função ou através de duas funções diferentes. É definida, então, alguma métrica para associar os identificadores dos nós com os identificadores dos recursos, normalmente fazendo com que os recursos fiquem sob a responsabilidade do nó que possui o identificador mais próximo ao seu. Para permitir que um nó encontre os recursos de outro nó, os identificadores são organizados de acordo com alguma estrutura lógica, como grafos, árvores, círculos, grades, hiper-cubos, entre outros. Essas estruturas lógicas, juntamente com os mecanismos necessários para permitirem a propagação das mensagens através dos seus pontos, são os principais componentes de uma rede *overlay*. A figura 2.2 ilustra esse modelo, onde as linhas pontilhadas dentro do espaço de identificadores representam as diversas possibilidades de associação entre os identificadores dos nós e os identificadores dos recursos.

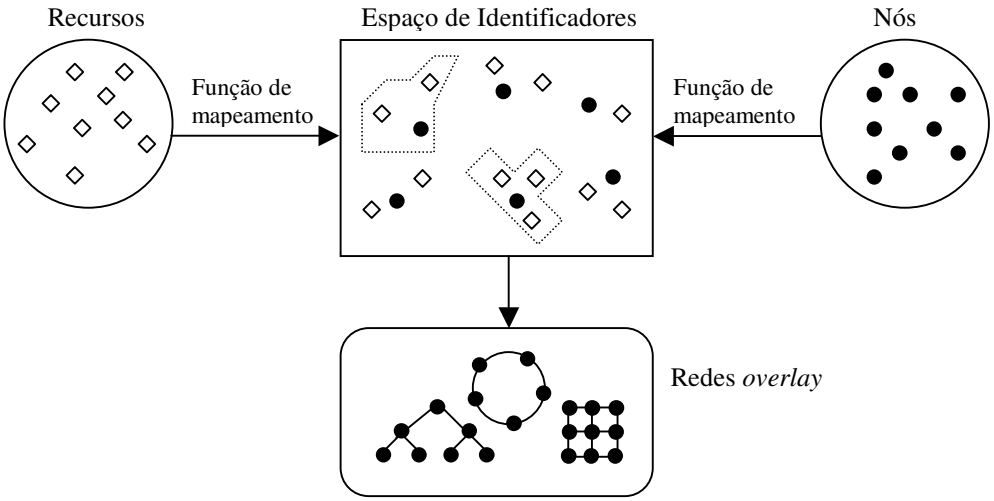


Figura 2-2. Associação entre nós e recursos na composição das redes *overlay*.

Podemos identificar seis aspectos principais que são comuns a qualquer rede *overlay* e que são os responsáveis pela diversidade de redes existentes, na medida em que cada uma resolve essas questões de uma forma diferente. São eles:

1. Escolha do espaço de identificadores
2. Mapeamento dos recursos e nós para o espaço de identificadores
3. Gerenciamento do espaço de identificadores pelos nós
4. Estrutura lógica da rede
5. Estratégia de roteamento
6. Estratégia de manutenção

As decisões de projeto a respeito das questões acima devem atender aos requisitos característicos das redes *overlay*, que são: eficiência, escalabilidade, auto-organização, tolerância a falhas e cooperação.

2.3.1.1 Escolha do Espaço de Identificadores

O espaço de identificadores tem o papel de funcionar como uma espécie de endereço dos recursos e dos nós, o que em algumas redes leva a uma independência da localização física, fazendo com que os nós possam ser movidos sem que isso implique obrigatoriamente na troca de seus identificadores. Além disso, por serem identificadores

virtuais, podem ser definidos de uma maneira específica da aplicação de modo a preservar informações semânticas importantes relativas à mesma.

O espaço de identificadores deve possuir suporte ao cálculo de uma métrica *d* que represente a distância entre dois identificadores. Essa métrica é normalmente utilizada na atribuição dos recursos aos nós e nos protocolos de roteamento de mensagens.

As redes CAN [17], por exemplo, usam um espaço euclidiano com os identificadores sendo as coordenadas nesse espaço. Nas redes Chord [16], o espaço de identificadores é composto por um subconjunto dos números naturais de tamanho N, e supondo que *x* e *y* são identificadores desse espaço a métrica *d* é calculada pela fórmula:

$$d(x,y) = (y - x) \bmod N.$$

2.3.1.2 Mapeamento dos Recursos e Nós para o Espaço de Identificadores

O mapeamento dos nós para o espaço de identificadores é feito através de uma função de mapeamento que toma como entrada alguma identificação do nó, tipicamente o seu endereço IP, e fornece como resultado um valor do espaço de identificadores, ou seja, um identificador.

O mapeamento dos recursos para o espaço de identificadores segue a mesma regra do mapeamento dos nós, ou seja, uma função de mapeamento é aplicada aos dados e fornece como resultado um identificador. Essa função tanto pode ser a mesma utilizada para o mapeamento dos nós quanto outra função diferente.

O espaço de identificadores deve ser grande o suficiente para que a probabilidade de que dois nós, ou recursos, diferentes, resultem em um mesmo identificador seja muito pequena. Evidentemente deve ser escolhida uma função de mapeamento que possua essa característica.

Dois comportamentos das funções de mapeamento devem ser especialmente considerados. O primeiro refere-se à capacidade de realizar o mapeamento de modo a beneficiar alguma característica específica da aplicação. Se existe, por exemplo, um grupo de recursos que normalmente são acessados em conjunto, a função de mapeamento poderia mapeá-los de modo a fazer com que a distância *d* entre seus identificadores seja pequena. Esse comportamento, entretanto, compromete o balanceamento de carga, fazendo com que alguns nós recebam mais recursos e solicitações que outros. Portanto, esses comportamentos são opostos e quando o

balanceamento de carga for uma característica desejada, deve ser utilizada uma função de mapeamento que resulte em uma distribuição homogênea no espaço de identificadores.

As redes Chord, por exemplo, utilizam funções *hash* para mapear tanto os nós quanto os recursos, o que acarreta uma distribuição homogênea de ambos no espaço de identificadores e garante o balanceamento de carga. Por outro lado, operações de pesquisas mais complexas, como buscas por faixas de valores, são bem mais difíceis de serem suportadas. As redes P-Grid [47], por outro lado, permitem gerar identificadores mais próximos de acordo com os dados mapeados, porém requer uma estratégia de balanceamento de carga.

2.3.1.3 Gerenciamento do Espaço de Identificadores

Cada nó fica responsável por um conjunto de identificadores e conseqüentemente dos recursos associados a eles. Entretanto, como em uma rede *overlay* as máquinas entram e saem da rede constantemente, é comum que um identificador fique sob a responsabilidade de vários nós diferentes durante a operação da rede. Dessa forma, localizar um recurso consiste em encontrar o nó responsável pelo seu identificador em um dado instante.

A forma mais comum de associação dos identificadores aos nós é atribuir um identificador *i* ao nó que possui como identificador o ponto do espaço de identificadores mais próximo de *i*.

Embora o comportamento comum seja um nó ficar responsável por vários identificadores, também é possível que um único identificador fique sob a responsabilidade de vários nós. Isso ocorrerá quando se desejar fornecer tolerância a falhas, pois implicará que um mesmo dado seja replicado entre diversos nós. Além disso, essa replicação pode ser aplicada uniformemente em todos os identificadores ou apenas em um subconjunto deles. Esse último caso é útil para evitar pontos de sobrecarga em dados muito requisitados.

No Chord, por exemplo, onde o espaço de identificadores é composto por um anel, cada nó é responsável pelos identificadores entre o ponto para o qual foi mapeado (incluindo-o) e o ponto anterior mais próximo para o qual algum outro nó tenha sido mapeado.

2.3.1.4 Estrutura Lógica da Rede

Como redes P2P são projetadas para suportarem um número muito grande de participantes, nenhum nó da rede deve possuir uma visão completa da mesma, mas sim conhecer apenas um pequeno conjunto de outros nós. Evidentemente, as conexões existentes entre os nós não podem gerar grupos isolados, uma vez que todos os nós do sistema devem estar, mesmo que indiretamente, conectados entre si. Além disso, as conexões não precisam estar restritas a nós próximos uns dos outros. Em muitas redes existem conexões tanto para nós vizinhos quanto para outros nós localizados em pontos distantes na estrutura da rede. Isso vai de acordo com o fenômeno chamado de "*small world"* [48], que é bastante considerado nos projetos das redes P2P, e afirma que para qualquer grupo existe uma curta cadeia de relacionamento interligando quaisquer pares de seus elementos.

Quanto mais conexões existem entre os nós, ou seja, quanto mais nós da rede cada nó conhece, maior suporte a tolerância a falhas e melhor o desempenho para encontrar um nó. Isso se deve ao fato de que existindo mais caminhos entre dois pontos, quando o caminho mais curto estiver indisponível outros poderão ser utilizados. Porém, quanto mais conexões um nó possuir mais informações tem que manter e mais operações precisa realizar quando ocorrem mudanças na topologia da rede. Dessa forma, é importante que esse parâmetro seja bem dimensionado para garantir a escalabilidade da rede. Para isso, o valor normalmente adotado como o número de nós que cada nó deve conhecer é O(log N), onde N é o número de nós na rede.

2.3.1.5 Estratégia de Roteamento

A operação principal desempenhada por uma rede *overlay* é encontrar o nó responsável por um determinado identificador. Para isso uma mensagem enviada por um nó deve ser reencaminhada por diversos nós na rede até o destino. Embora as estruturas lógicas de rede, discutidas na seção anterior, forneçam um meio para que dois nós comuniquem-se, é necessário um algoritmo de roteamento para que um nó decida para qual(is) dos nós, entre os que ele tem uma conexão, a mensagem deve ser reencaminhada. Existem basicamente três formas de realizar essa operação:

- Centralizada. Nessa estratégia praticamente não existe roteamento pois como todos os nós possuem uma conexão com um nó central, onde

publicam os recursos disponíveis, as requisições são enviadas diretamente para esse nó que escolhe qual dos nós registrados melhor resolve a consulta e retorna a sua identificação. Desse modo, o nó solicitante pode comunicar-se diretamente com o nó retornado. Apesar de apresentar problemas de escalabilidade, essa técnica foi utilizada pelo sistema Napster [12].

- Inundação. Nessa abordagem, cada nó repassa as requisições recebidas para todos os nós que estão diretamente conectados a ele. Esse processo repete-se em cada nó da rede até que a requisição seja atendida ou o número máximo de encaminhamentos definido seja atingido. Por sobrecarregar os nós e os enlaces de comunicação esse modelo apresenta problemas de escalabilidade. Entretanto, existem técnicas especiais, como a utilização de mecanismos de *cache* e a utilização de *super-peers*, que concentram um grande número de requisições, visando permitir a utilização desse modelo em redes com um grande número de nós. Um exemplo de sistema que utiliza a técnica de inundação é o Gnutella [14] .
- DHT (*Distributed Hash Table*). Atualmente tem surgido um grande número de redes P2P que seguem essa abordagem. Em uma rede DHT [25] [49] tanto os nós quanto os dados a serem armazenados são mapeados para pontos do espaço de identificadores utilizando uma função *hash*. O dado ficará sob a responsabilidade do nó que possui o identificador igual, ou mais próximo, ao seu. Cada nó possui conexões com apenas um pequeno conjunto de outros nós e repassa as mensagens para apenas um deles. O nó escolhido será o que possui o identificador mais próximo do identificador pesquisado. Algumas redes DHT utilizam mecanismos de *cache* onde os dados solicitados são copiados para todos os nós percorridos durante a requisição, de modo que novas consultas possam ser satisfeitas em pontos mais próximos do nó requisitante.

Duas propriedades importantes dos algoritmos de roteamento são: a garantia da localização do dado e o número de nós percorridos até o destino. Em redes por inundação, por exemplo, não existe garantia de que um dado será encontrado, mesmo que exista na rede. Quanto ao número de nós intermediários que são acessados em uma

pesquisa, para garantir a escalabilidade da rede esse valor deve ser no máximo O(log N), onde N é o número de nós na rede.

2.3.1.6 Estratégia de Manutenção

Como os nós têm autonomia para entrar e deixar a rede quando desejarem, a topologia de uma rede *overlay* muda constantemente. Dessa forma, faz-se necessário um mecanismo que mantenha as informações nas tabelas de roteamento consistentes de forma a permitir o correto encaminhamento das mensagens. Essa tarefa torna-se ainda mais complexa uma vez que, embora a operação de entrada na rede seja feita de forma explícita pelos nós, a saída pode não seguir o mesmo comportamento. Ou seja, nós podem deixar a rede silenciosamente devido a travamento nas máquinas ou perda de conectividade, por exemplo.

As estratégias de manutenção podem ser classificadas [10] como *proativas*, quando verificações periódicas são realizadas na estrutura da rede, e *reativas*, quando as medidas de manutenção são aplicadas apenas quando ocorre algum problema de conexão entre dois nós. Independente da estratégia escolhida, o sucesso da utilização prática de alguma rede P2P depende de forma vital da eficiência de seu sistema de manutenção.

2.3.2 Modelos de Arquiteturas P2P

Existem algumas classificações para as redes *overlay*, sendo que uma das mais utilizadas, principalmente na comunidade acadêmica, é baseada nos mecanismos de consulta e topologia da rede [24], e que as divide em três categorias:

- Centralizadas. A rede possui um nó central (possivelmente com algumas réplicas para melhorar a confiabilidade e o desempenho) que mantém informações sobre todos os nós e recursos da rede. Todas as consultas são feitas diretamente a esse nó central. Embora mais conhecida devido ao Napster, também é muito utilizada nos sistemas de mensagens instantâneas.
- Descentralizadas e não estruturadas. Nessas redes não existe um controle muito rígido sobre a topologia da rede e as consultas são propagadas de nó em nó até que encontrem o destino ou que algum mecanismo de

timeout encerre o processo. Redes como o Gnutella e Kazaa utilizam essa arquitetura.

- Descentralizadas e estruturadas. Possuem uma topologia bem definida e utilizam regras para distribuir os dados na rede de modo que facilite a posterior localização dos mesmos. Em geral, as redes baseadas em DHT se enquadram nessa categoria, como é o caso da Chord, CAN e Pastry.

2.4 Exemplos de Redes Peer-to-Peer

Nessa seção são apresentadas as principais redes P2P existentes. Para algumas delas existem, inclusive, redes públicas já implantadas que permitem o ingresso de qualquer usuário. Para isso as implementações são disponibilizadas para *download* e tudo que o usuário precisa fazer é instalar o software na sua máquina. Mesmo quando não existe uma única rede pública mundial, as implementações normalmente também estão disponíveis de modo que é possível criar redes isoladas, que podem estar espalhadas por toda a Internet.

2.4.1 Gnutella

O Gnutella [14] é um protocolo para compartilhamento de arquivos na Internet que permite a busca de arquivos através de seus nomes, ou partes dele, e a posterior obtenção do arquivo diretamente da máquina de outros usuários. No Gnutella não existe um diretório centralizado, como no Napster, e as buscas são feitas de forma distribuída. Além disso, também não existe um controle sobre a topologia da rede nem dos locais onde os dados são armazenados.

Para se conectar à rede, um nó precisa conhecer previamente algum outro nó da rede, o que pode ser feito através de publicações dos endereços de alguns nós em um serviço de diretórios ou servidor web bem conhecido. Desse modo, o nó entrando na rede solicita a alguns desses nós que lhe enviem a lista de nós conhecidos, que cada um possui, e seleciona um subconjunto dos nós retornados para manter conexões. Portanto, cada nó mantém informações a respeito de alguns nós vizinhos e propaga as consultas para todos esses nós que, por sua vez, repetem esse procedimento. A consulta será limitada a um determinado raio informado em um campo chamado TTL *(Time to Live)*, que especifica o número máximo de vezes que uma consulta pode ser propagada. Esse

método de propagação gera um alto tráfego na rede, porém permite consultas bastante flexíveis, especificando palavras-chave para serem comparadas com os nomes dos arquivos. Como os nós podem estar conectados a velocidades bastante heterogêneas, existe um campo nas consultas que determina a velocidade mínima que um nó deve possuir para poder responder a consulta com sucesso. Dessa forma, evita-se que o *download* dos arquivos seja feito de máquinas com baixa taxa de transmissão.

2.4.2 Chord

Normalmente utiliza-se o Chord [16] para armazenar pares contendo uma chave e seu valor associado em nós distribuídos pela rede. Posteriormente, o serviço de busca da rede permite que, dada uma chave, seja determinado o nó responsável pela mesma. As redes Chord são baseadas no modelo DHT (*Distributed Hash Table*).

O espaço de identificadores é formado por um anel conectado, onde cada identificador possui *m* bits (tipicamente 160 bits). Tanto os nós quanto os dados a serem armazenados são mapeados através de uma função *hash* consistente [50] para pontos desse espaço de identificadores. O mapeamento dos nós é feito aplicando-se a função *hash* ao seu endereço IP. Cada chave *k* é armazenada no nó cujo identificador é igual, ou segue *k*, no espaço de identificadores. Para permitir a localização das chaves seria necessário apenas que cada nó mantivesse um apontador para o nó sucessor e o predecessor no anel. Porém, como essa estratégia implicaria em buscas muito ineficientes, cada nó mantém uma tabela de roteamento para outros O(log N) nós, onde N é o número de nós da rede. Supondo que um determinado nó tem como identificador o valor *i*, a sua tabela conterá apontadores para os nós responsáveis pelos identificadores $i+2^0$, $i+2^1$, $i+2^2$, ... $i+2^{\log N}$. Portanto, essa tabela permite que o roteamento das mensagens seja feito de forma semelhante a uma busca em uma árvore binária, onde a cada passo o espaço de pesquisa é reduzido a metade. Com essa estratégia o número de nós consultados em uma busca é O(log N).

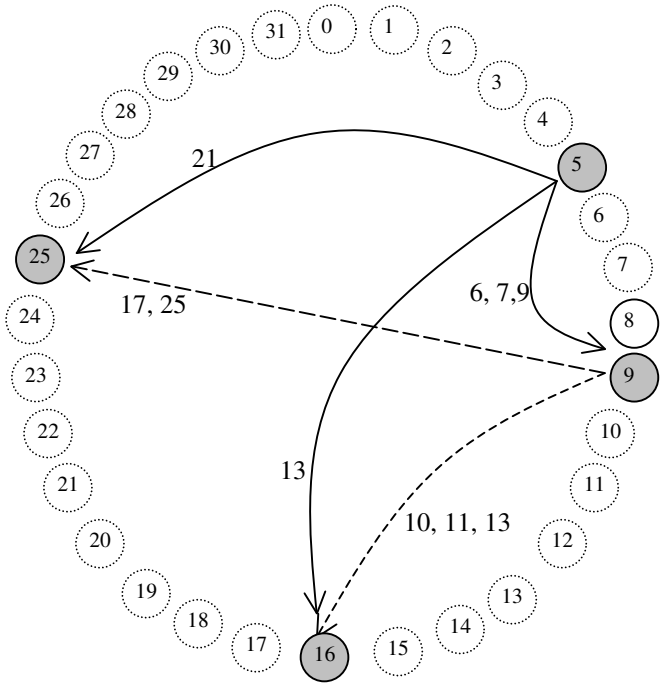


Figura 2-3. Rede Chord com 4 nós e 32 identificadores.

A figura 2.3 mostra uma rede Chord onde os identificadores possuem um tamanho de 5 bits, de modo que o espaço de identificadores será formado por 32 valores diferentes. Nessa rede existem quatro nós, representados pelos círculos sólidos que, portanto, foram mapeados pela função *hash* para os identificadores 5, 9, 16 e 25. As linhas entre os nós representam os apontadores mantidos por cada um deles e os números junto a elas indicam os identificadores para os quais o apontador se refere. Evidentemente, os nós 16 e 25 também possuem apontadores, mas esses não são mostrados na figura para não sobrecarregá-la e dificultar a compreensão. A tabela 2.1 mostra como as chaves são associadas aos nós e na tabela 2.2 é visto como fica a tabela de apontadores para os nós 5 e 9.

Tabela 2-1. Associação das chaves aos nós na rede Chord da figura 2.3.

Nós	**Chaves**
5	26–31 e 0-5
9	6-9
16	10-16
25	17-25

Tabela 2-2. Ponteiros mantidos pelos nós 5 e 9 na rede Chord da figura 2.3.

Nós	Índice (*i*)	Identificador ($5 + 2^i$)	Nó
5	0	6 (5 + 1)	9
	1	7 (5 + 2)	9
	2	9 (5 + 4)	9
	3	13 (5 + 8)	16
	4	21 (5 + 16)	25
9	0	10 (9 + 1)	16
	1	11 (9 + 2)	16
	2	13 (9 + 4)	16
	3	17 (9 + 8)	25
	4	25 (9 + 16)	25

2.4.3 CAN (*Content-Adressable Network*)

Uma rede CAN [17] é constituída por um espaço d-dimensional para o qual os dados são mapeados através de uma função *hash*. Cada nó fica responsável por uma região desse espaço. Ao entrar na rede, um nó escolhe aleatoriamente um ponto desse espaço e se comunica com o nó atualmente responsável por ele, solicitando que divida a sua região ao meio e passe a responsabilidade de uma das metades para esse novo nó. Dessa forma, todos os dados cujos identificadores correspondem a pontos de uma determinada região desse espaço, ficam sob a responsabilidade do nó encarregado dessa região. A figura 2.4 ilustra uma rede CAN com 5 nós formada por um espaço de 2 dimensões.

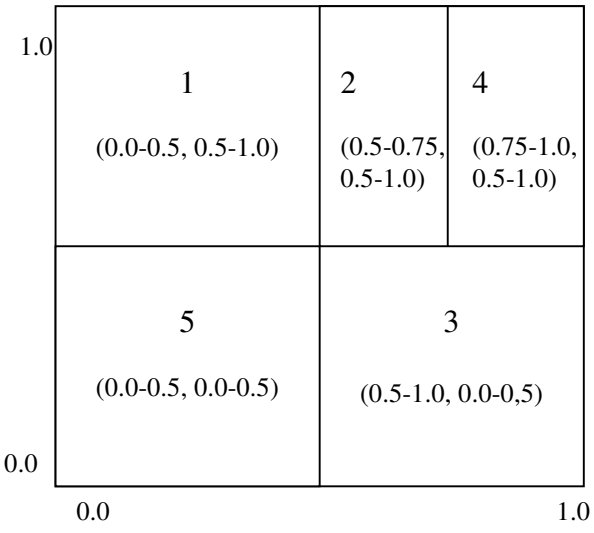


Figura 2-4. Rede CAN com 5 nós em um espaço de 2 dimensões.

Cada nó mantém apontadores para os seus vizinhos juntamente com informações sobre as regiões do espaço que são responsáveis. Para realizar uma busca por um dado, primeiramente a função de mapeamento é aplicada ao mesmo, obtendo-se o ponto do espaço equivalente. A seguir encaminha-se a consulta ao nó vizinho cuja região é mais próxima em direção a esse ponto.

Com essa arquitetura as consultas percorrem no máximo $d \cdot N^{1/d}$ nós e o número de apontadores mantidos em cada nó é $2 \cdot d$.

2.5 Super-Peers

Visando obter o bom desempenho proporcionado pelas arquiteturas centralizadas, mas mantendo a escalabilidade e tolerância a falhas que são inerentes às soluções distribuídas, existe uma variação da arquitetura P2P convencional, que atribui funções especiais a alguns nós da rede, fazendo com que atuem como servidores para um certo número de outros nós. Nessa arquitetura, apenas esses nós especiais, que são chamados de *super-peers*, é que participam da rede diretamente, ou seja, possuem conexões com outros nós, enviam e respondem pesquisas e roteiam mensagens. Os demais nós se conectam em algum *super-peer* e passam todas as informações que desejam disponibilizar na rede para o mesmo. Portanto, um *super-peer* reúne informações de diversos nós e as trata como se fossem suas, respondendo, ele próprio, a pesquisas buscando por essas informações. Um nó comum envia suas pesquisas para o *super-peer* ao qual está ligado e este, por sua vez é quem envia a pesquisa na rede P2P. Quando o *super-peer* recebe a resposta ele a repassa para o cliente que fez a solicitação. A Figura 2.5 ilustra o modelo de *super-peers*.

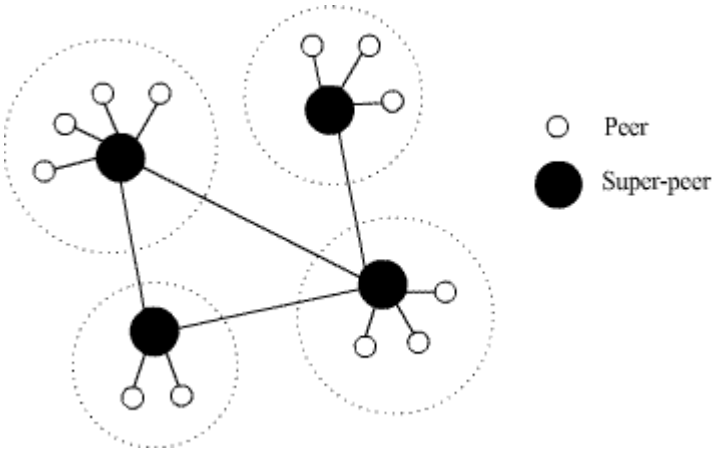


Figura 2-5. Arquitetura de *Super-peers*.

Uma arquitetura de *super-peers* nada mais é, portanto, que uma rede P2P composta apenas por *super-peers*, que atuam como nós iguais nessa rede e como servidores para um conjunto de outros nós. Esse modelo reduz bastante o número de reconfigurações na rede decorrentes da entrada e saída de nós, além de lidar com o problema da heterogeneidade dos mesmos. Nós com recursos computacionais limitados podem comprometer o desempenho da rede, uma vez que o modelo de roteamento das redes P2P, onde cada nó realiza o roteamento de mensagens com igual responsabilidade, é bastante vulnerável a falhas ou gargalos em qualquer um dos nós. Portanto, limitar que o roteamento seja realizado apenas pelos *super-peers* evita esse problema. Além disso, pesquisar em um conjunto menor de nós, que possuem índices com as informações de diversos outros nós, é mais eficiente que pesquisar cada um deles individualmente, principalmente se possuírem recursos limitados.

Embora a utilização de *super-peers* proporcione uma melhora significativa no desempenho da rede, ela acarreta uma grande sobrecarga nos nós que realizam essa função, de modo que não existe muito interesse para um nó em se tornar um *super-peer* [57]. Além disso, ainda existem alguns pontos que não estão bem definidos a respeito da forma exata como a arquitetura da rede deve se constituir, mais especificamente: qual deve ser o número de *super-peers* na rede, quantos nós cada *super-peer* deve atender, em qual *super-peer* um nó deve se conectar*,* em quantos *super-peers* cada nó deve se conectar, entre outras. Maiores detalhes sobre esses aspectos podem ser encontrados em [11].

As redes Edutella [64] utilizam uma arquitetura baseada em *super-peers*, onde os *super-peers* formam uma topologia em Hiper-cubo e os demais nós conectam-se em algum desses *super-peers*. São necessárias O(log N) mensagens para inserir um novo *super-peer* na rede. Embora o roteamento possa ser realizado por difusão, o método adotado normalmente utiliza índices construídos através da descrição em RDF [65] dos recursos de cada nó. Existem dois níveis de índices. Os índices de primeiro nível, chamados *super-peer*/nó, servem para descrever as características dos nós comuns registrados no *super-peer* e determinam, conseqüentemente, para quais nós conectados no *super-peer* as mensagens são repassadas. Os índices de segundo nível, chamados *super-peer/super-peer,* são construídos a partir dos índices de primeiro nível e servem para guiar o processo de repasse das mensagens na rede de *super-peers*. Cada índice pode possuir diferentes granularidades, ou seja, podem conter informações sobre os

nomes dos esquemas suportados, os nomes das propriedades ou os valores das propriedades. O número de nós vizinhos para os quais cada nó mantém conexões é $O(\log_2 N)$. Da mesma forma, o número máximo de nós visitados em uma pesquisa é $O(\log_2 N)$.

3 Arquitetura da Rede SGrid

Semelhante a outras redes P2P, o SGrid mapeia identificadores, também chamados de chaves, para nós. Para isso, cada nó da rede fica responsável por um conjunto de identificadores. Uma operação de busca por um determinado identificador retornará o nó (*peer*) responsável pelo mesmo.

O SGrid é composto por um espaço bi-dimensional, onde os lados são iguais a potências de 2, ou seja, o $\log_2$ do tamanho do lado deve ser um número inteiro. Dessa forma, dada uma grade com tamanho do lado 256, por exemplo, há 65.535 possíveis valores. Para a operação da rede, os dados a serem armazenados e os nós da rede, são mapeados para pontos desse espaço.

3.1 Mapeamento dos Nós para Pontos da Grade

O mapeamento dos nós para o espaço bi-dimensional do SGrid tem como base o endereço IP do nó. Dessa forma, o endereço IP determina a coordenada (x,y) da grade que será associada ao nó. A idéia é que cada nó fique responsável pela(s) chave(s) que mapeiam para a mesma coordenada para a qual ele é responsável (determinada pelo mapeamento do endereço IP).

É importante ressaltar que o mapeamento dos nós deve levar em consideração a localização física real dos nós, para que as distâncias entre dois nós na rede P2P sejam semelhantes às distâncias entre esses mesmos dois nós na rede IP real, ou seja, na Internet. Embora qualquer técnica que considere as distâncias entre dois nós na rede IP real, como recentes pesquisas em Coordenadas Virtuais na Internet [51] [52], possam ser utilizadas no SGrid, apresentamos uma nova proposta para realizar essa tarefa.

O modelo proposto apóia-se no fato de que os endereços IP são distribuídos obedecendo a um esquema hierárquico que facilita o roteamento e permite que o agrupamento de rotas para várias redes através de uma única rota. Nesse esquema hierárquico cada novo nível da hierarquia é representado por um conjunto de bits posicionados mais à direita no endereço IP e esses níveis normalmente refletem as conexões de rede existentes.

Para compreender melhor esse esquema analisemos rapidamente como a distribuição de endereços é realizada. Em qualquer país existe, evidentemente, um número muito grande de redes, e seria inviável se os roteadores na Internet precisassem

de rotas para cada uma delas. Se todas as redes de qualquer país possuírem um prefixo comum e esse prefixo for exclusivo para cada um deles, o modelo de agrupamento de rotas pode ser utilizado e apenas uma rota é necessária para encaminhar dados para todas as redes de um dado país. Evidentemente existem países que precisam ter mais de um prefixo. Dentro de cada país existem empresas habilitadas a fornecerem serviços de Internet e possuem *backbones* de abrangência nacional aos quais outras empresas podem conectar-se. Novamente, cada fornecedor de *backbone* possuirá várias redes e o roteamento será melhor realizado se todas elas possuírem um prefixo comum, ou seja, cada fornecedor de *backbone* também possui um prefixo exclusivo, que será acrescido ao prefixo já existente do país. Cada empresa que se conecta ao fornecedor de *backbone* receberá um prefixo diferente. Finalmente, cada máquina dentro da rede de cada empresa, receberá também, um identificador diferente.

Dessa forma, analisando os bits de um endereço de forma hierárquica, da esquerda para a direita, pode-se identificar máquinas que pertencem à mesma rede, redes que pertencem ao mesmo fornecedor de *backbone*, ou fornecedores de *backbone* que pertencem ao mesmo país. Usando essas características pode-se distribuir os nós na grade de forma que, quanto mais níveis em comum (país, *backbone*, empresas, etc) dois nós compartilharem, mais próximos estarão posicionados na grade. Para isso, dividimos o endereço IP da máquina em m grupos de n bits, onde m é igual ao número de níveis existentes na grade, e n é igual ao tamanho do endereço, em bits, divido por m. Cada um dos m grupos, seguindo da esquerda para a direita, determinará o quadrante em cada nível, do maior para o menor, que a máquina será posicionada. Para isso é calculado o resto da divisão do número equivalente aos n bits por 4. O valor resultante determina o quadrante escolhido, de acordo com a seguinte regra: 0, representa o quadrante superior esquerdo; 1 representa o quadrante superior direito; 2, representa o inferior esquerdo e 3 representa o quadrante inferior direito.

A figura 3.1 mostra como a posição do nó que possui o endereço IP 200.241.86.134 é determinada, para uma grade de lado 256. Como uma grade de lado 256 possui 8 níveis e um endereço IPv4, 32 bits, há 8 grupos de 4 bits cada. A tabela 3.1 mostra como fica a divisão do endereço IP em grupos e o quadrante equivalente a cada grupo uma vez que a representação do endereço 200.241.86.131 em binário é 11001000.11110001.01010110.10000011.

Tabela 3-1. Mapeamento do endereço IP para quadrantes em cada nível.

Nível	Bits	Bits(decimal)	Bits(decimal) Mod 4	Quadrante
8	1100	12	0	Superior esquerdo
7	1000	8	0	Superior esquerdo
6	1111	15	3	Inferior direito
5	0001	1	1	Superior direito
4	0101	5	1	Superior direito
3	0110	6	2	Inferior esquerdo
2	1000	8	0	Superior esquerdo
1	0011	3	3	Inferior direito

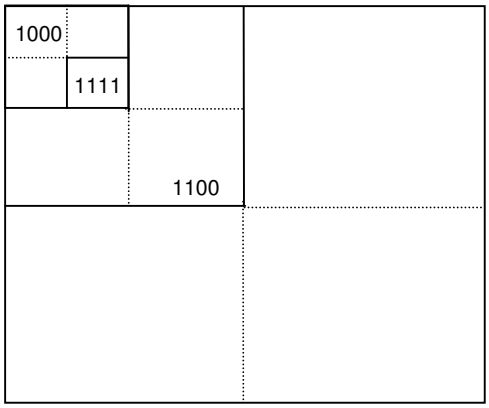


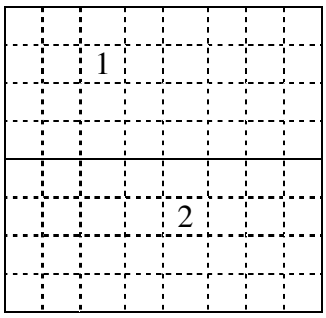
Figura 3-1. Mapeamento relativo aos 12 bits mais à esquerda do endereço IP.

3.2 Distribuição das Chaves aos Nós

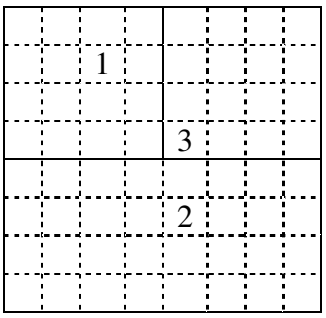
Conforme explicado na seção anterior, cada nó é associado a uma posição da grade e fica encarregado da chave que seja mapeada para essa mesma coordenada. Porém, como normalmente não existe um nó para cada ponto da grade, cada nó deve ficar responsável por um conjunto de chaves, conforme citado anteriormente. O primeiro nó da rede fica responsável por todas as chaves e a partir daí, para cada novo nó que entra na rede, um dos nós existentes divide suas chaves com o novo nó. Sobre esse processo três pontos devem ser obedecidos. Primeiro, o nó que tem as suas chaves divididas é o nó responsável pelo ponto do espaço para o qual o endereço IP do novo nó mapeia. Segundo, a divisão das chaves é feita dividindo-se a região formada pelas chaves do nó em duas de tamanhos iguais, o que implica em divisões horizontais ou verticais, tentando-se gerar regiões quadradas sempre que possível. Terceiro, caso o mapeamento do endereço IP do novo nó não fique dentro da nova região criada pela

divisão das chaves, a coordenada associada ao nó será o ponto dentro da nova região que fica mais próximo do ponto de mapeamento original.

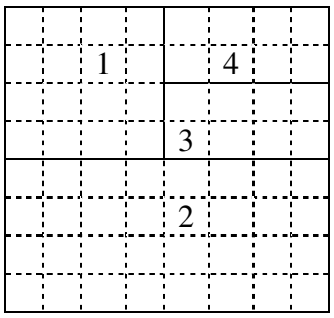
A figura 3.2 ilustra como fica a distribuição das chaves após a entrada de sete nós numa rede formada por uma grade de lado 8. O número dentro dos quadrados indica a ordem de entrada dos nós na rede. O quadrado onde o número encontra-se indica a chave (coordenada/posição) pela qual o nó é responsável. Essa deveria ser a coordenada para a qual o endereço IP do nó foi mapeado. Por exemplo, o fato do nó "1" está na posição x=2 e y=1 (os índices começam com zero), indica que o mapeamento do endereço IP resultou no identificador 11 (que transformado em coordenadas x e y equivalem a 2 e 1, respectivamente). Porém, conforme explicado no parágrafo anterior, se o mapeamento do endereço ficar fora da região pela qual o nó ficará responsável, essa coordenada deve ser deslocada para dentro da mesma. Por exemplo, suponha que o mapeamento do endereço IP do nó 3, tenha resultado no valor 25 (x=0 e y=3), como esse ponto não faz parte da região que será atribuída ao nó, a coordenada x foi deslocada 4 pontos para a direita para poder ficar dentro da referida região.



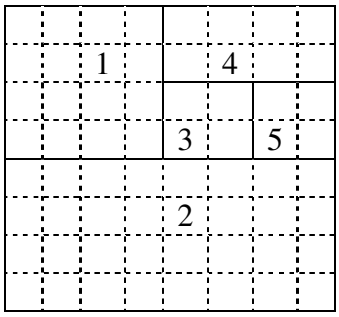
a) Divisão das chaves após a entrada dos nós 1 e 2.



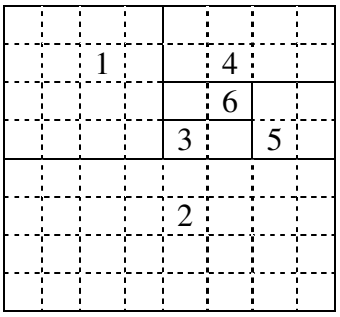
b) Divisão das chaves após a entrada do nó 3.



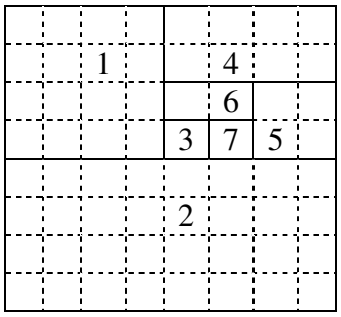
c) Divisão das chaves após a entrada do nó 4.



d) Divisão das chaves após a entrada do nó 5.



e) Divisão das chaves após a entrada do nó 6.



f) Divisão das chaves após a entrada do nó 7.

Figura 3-2. Divisão do espaço de chaves após a entrada dos nós.

3.3 Mapeamento dos Dados para Pontos da Grade

Além dos nós da rede, os dados a serem armazenados também precisam ser mapeados para o espaço bi-dimensional do SGrid. Esse processo é realizado através da aplicação de uma função *hash*, como SHA-1[56] por exemplo, no dado. Algumas vezes a informação utilizada para o cálculo da função *hash* não consiste no dado completo que será armazenado, mas sim em uma parte que pode ser usada como chave de pesquisa para indexar a informação completa. Um exemplo típico são aplicações de armazenamento de arquivos, onde a informação utilizada para o cálculo da função *hash* não consiste do conteúdo do arquivo, mas apenas do seu nome. Embora essa operação de mapeamento do dado a ser armazenado para nós resulte em identificadores "seqüenciais", no SGrid eles são manipulados como coordenadas x e y. Esse mapeamento pode ser facilmente realizado através de funções de conversão. Para a obtenção da coordenada x, por exemplo, basta obter-se o resto da divisão da chave pelo tamanho do lado. Procedimento semelhante pode ser aplicado para a obtenção da coordenada y. Para ilustrar esse processo de mapeamento tomemos como exemplo novamente a aplicação de armazenamento de arquivos. Supondo uma grade de lado 8 (área 64) e que a aplicação da função *hash* (módulo 64) no nome de arquivo "Redes Peer-to-Peer.pdf", retornou o valor 11, isso significa que esse arquivo deve ser armazenado no nó responsável pelo identificador 11 que, em termos de coordenadas x,y equivale ao identificador (3,1).

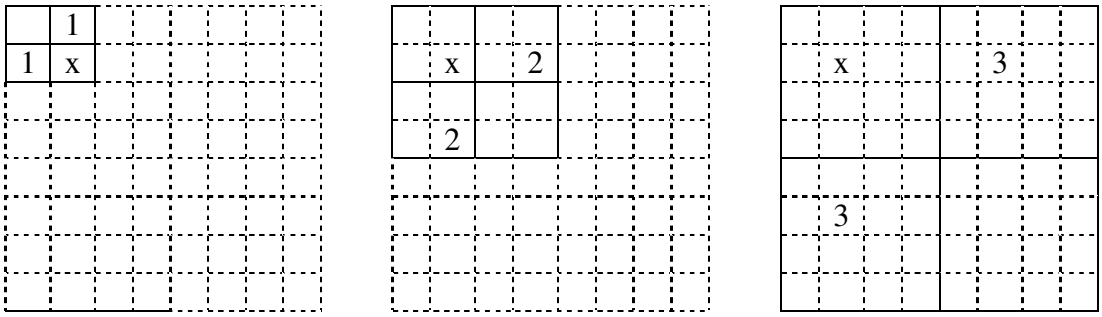
Um fator muito importante em qualquer rede P2P é que a distribuição dos dados entre os nós da rede seja feita de forma homogênea, ou seja, cada nó fique responsável por aproximadamente a mesma quantidade de chaves. Como os dados são mapeados para pontos do espaço através de uma função *hash*, isso garante uma distribuição homogênea dos dados entre pontos do espaço. Além disso, como cada novo nó que entra na rede obtém a responsabilidade sobre a metade das chaves de algum outro nó existente, conforme explicado anteriormente, resta apenas mostrar que embora o esquema de mapeamento dos nós para pontos do espaço gere agrupamentos de nós (por considerar proximidade dos nós na rede IP) ainda assim obtém-se uma distribuição satisfatória dos nós no espaço da grade que atende aos requisitos de homogeneidade na alocação das chaves. Essa característica poderá ser comprovada através dos resultados das simulações.

3.4 Conexões Entre os Nós

Um dos pontos mais importantes de uma rede P2P diz respeito aos requisitos necessários para que um nó encontre o nó responsável por uma determinada chave. Esses requisitos incluem as estruturas de dados que devem ser mantidas em cada nó e o número de nós consultados para a realização da pesquisa. Para se atingir uma alta escalabilidade esses números devem crescer no máximo logaritimicamente em relação ao número de nós na rede. Ou seja, para uma rede com N nós, é desejável que uma pesquisa passe por, no máximo, log N nós. Da mesma forma, cada nó deve manter informações sobre um pequeno número de outros nós da rede, mais especificamente sobre O(log N).

Para atender aos requisitos citados acima, a grade é manipulada por cada nó da rede como uma estrutura hierárquica composta de (Log N)/2 níveis. Cada nível é formado por um, ou mais, quadrados que são compostos, cada um, por quatro quadrantes, onde um quadrado de nível n-1 é um dos quadrantes de um quadrado de nível n. Ou seja, os níveis são formados dividindo-se a grade em quatro quadrantes e repetindo-se esse processo recursivamente para cada um dos quadrantes, até obtermos quadrados formados por quadrantes de tamanho 1 (apenas um ponto da grade), que formam o nível 1 da hierarquia.

Pela estrutura descrita no parágrafo anterior, vemos que, para o nível 1, temos quadrantes com tamanho de lado 1 e quadrados com tamanho de lado 2. Para o nível 2, temos quadrantes com tamanho de lado 2 e quadrados com tamanho de lado 4. Para o nível 3, temos quadrantes com tamanho de lado 4 e quadrados com tamanho de lado 8. Generalizando, para um nível n, temos quadrantes com tamanho de lado 2 (n – 1) e quadrados com tamanho de lado 2n. A figura 3.3 mostra uma grade de lado 8, onde, conseqüentemente, existem 3 níveis. O nível 3 é formado pela divisão do quadrado que corresponde à própria grade (com lado 8 e área 64) em quatro quadrantes com lado 4 e área 16. O nível 2 é formado pela divisão de cada um dos quadrantes de nível 3 novamente em quatro quadrantes, cada um com lado 2 e área 4. E, finalmente, o nível 1 é formado pela divisão de cada um dos quadrantes de nível 2 novamente em quatro quadrantes, cada um com lado 1 e área 4.



a) Nível 1 b) Nível 2 c) Nível 3

Figura 3-3. Grade com 3 níveis e os nós apontados em cada nível.

Definimos o conceito de *quadrante vizinho* em um nível como quadrantes que compartilham um de seus lados, horizontal ou verticalmente, e que fazem parte de um mesmo quadrado do nível, ou seja, compõem o mesmo quadrante do nível superior. Dessa forma, cada quadrante de um nível possui exatamente dois quadrantes vizinhos, um na horizontal e outro na vertical. Neste texto, referiremos aos quatro quadrantes de um determinado nível da seguinte forma: quadrante 0, como o quadrante superior esquerdo; quadrante 1, como o quadrante superior direito; quadrante 2, como o quadrante inferior esquerdo e quadrante 3, como o quadrante inferior direito.

A estrutura hierárquica do SGrid permite atingirmos o objetivo de manter em cada nó informações sobre apenas O(Log N) outros nós. Como cada ponto da grade pertence a exatamente um quadrante em cada nível, cada nó mantém um apontador para um nó em cada quadrante vizinho de cada nível. Como existem (Log N)/2 níveis e apenas dois vizinhos em cada nível, um na horizontal e outro na vertical, cada nó precisará guardar ((Log N)/2)* 2, ou seja, Log N, informações sobre outros nós da rede.

Os nós dos quadrantes vizinhos em um dado nível para os quais um nó mantém apontadores são os nós responsáveis pelas mesmas coordenadas, relativas ao início de cada quadrante, que as coordenadas do nó original calculadas em relação ao seu próprio quadrante. Na figura 3.3, o “x” indica a posição de um determinado nó e os números, 1,2 e 3, indicam tanto os nós para os quais o nó “x” possui apontadores, quanto o nível ao qual o apontador se refere. Dessa forma, na figura 3.3a, os vizinhos de nível 1 do nó x são os pontos (1,0) - vizinho vertical - e (0,1) vizinho horizontal. Na figura 3.3b os vizinhos de nível 2 do nó x são os pontos (1,3) como vizinho vertical e (3,1) como vizinho horizontal. E, finalmente, na figura 3.3c, os vizinhos de nível 3 do nó x são os pontos (1,5) como vizinho vertical e (5,1) como vizinho horizontal. Vale ressaltar que

nem sempre existirá um nó exatamente na posição para a qual um nó aponta. Nesse caso o nó apontado será o nó responsável por aquela posição da grade.

3.5 Algoritmo de Busca (Roteamento das Mensagens)

A principal tarefa desempenhada por uma rede P2P é encontrar o nó associado a uma determinada chave. No SGrid os ponteiros mantidos por cada nó para os quadrantes vizinhos em cada nível permitem que as buscas sejam feitas de forma eficiente pois, embora um nó possua a maior parte de seus ponteiros para nós que estão mais próximos de si, os ponteiros para nós em quadrantes vizinhos nos níveis maiores permitem que regiões distantes da grade sejam alcançadas com poucos "saltos". A idéia básica do algoritmo de busca é que cada nó calcule o quadrado de menor nível ao qual o nó e a chave sendo pesquisada pertencem e repasse a consulta para o nó no quadrante vizinho desse nível. Quando o nó e a chave estão no mesmo quadrante de um nível o processo se repete, ou seja, calcula-se o quadrado de menor nível ao qual o nó e a chave sendo pesquisada pertencem e a consulta é repassada para o nó no quadrante vizinho desse nível. Esse procedimento é repetido até que o nó que recebe a consulta seja o nó responsável pela chave.

Uma vez que só existem quatro quadrantes em cada nível, e cada nó tem um ponteiro para o seu vizinho vertical e outro para o vizinho horizontal, para que o nó e a chave estejam no mesmo quadrante, em um dado nível, são necessários, no máximo, dois saltos, ou seja, contatar dois nós. Como depois que o nó e a chave estão no mesmo quadrante, a busca continua em um nível menor, e existem (Log N)/2 níveis, o número máximo de nós consultados em uma pesquisa é ((Log N)/2) * 2, ou seja, Log N. Esse é o pior caso, conforme veremos nos resultados dos testes. Na prática, a média das pesquisas se dá em (Log N)/2. Isso se deve a duas razões. Primeiro, nem sempre todos os níveis precisam ser consultados. Além disso, a situação onde precisamos de 2 saltos para que o nó e a chave estejam no mesmo quadrante deve ocorrer em apenas 25% dos casos, uma vez que há ponteiros para os vizinhos horizontais e verticais.

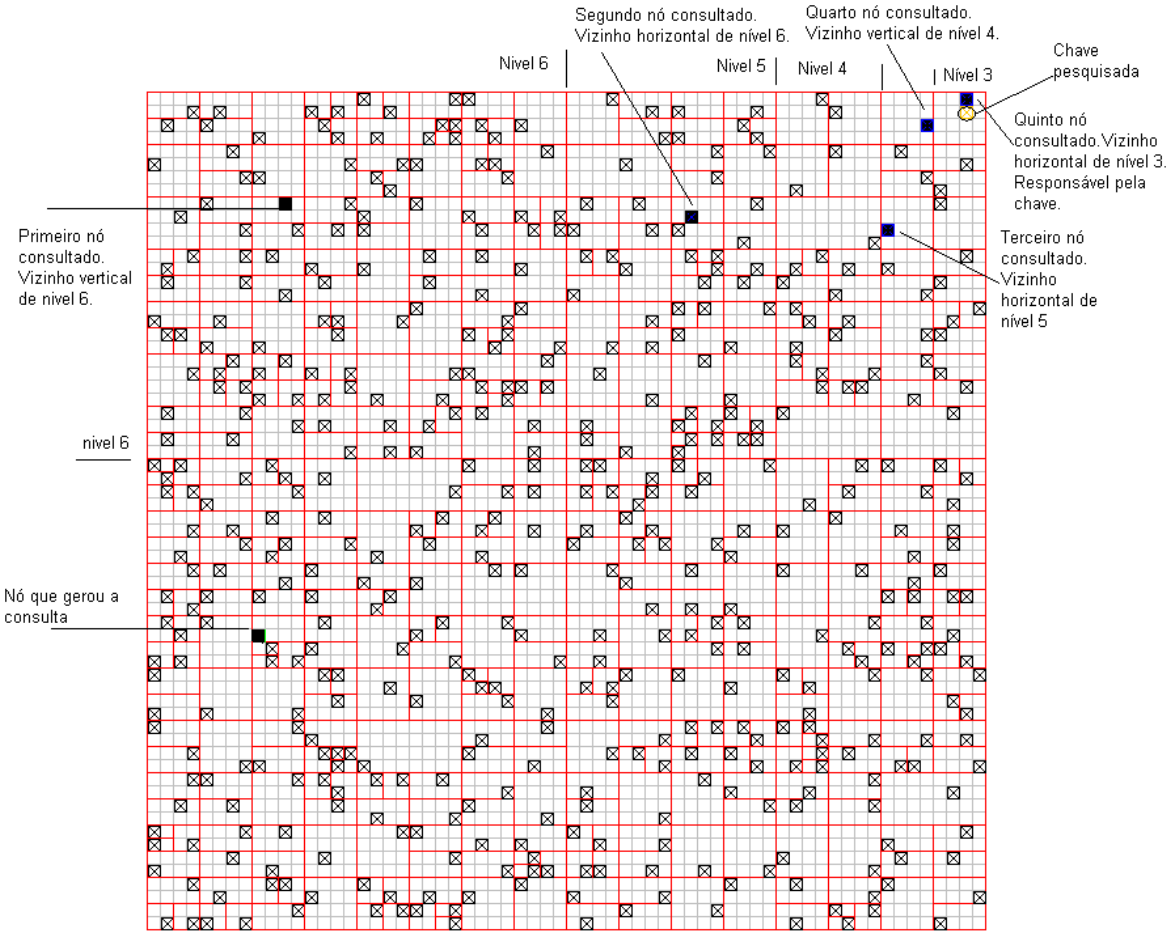


Figura 3-4. Pesquisa em uma grade de lado 64, com 512 nós.

A figura 3.4 ilustra a pesquisa pela chave (62,1), ponto marcado com um círculo, em uma rede com 512 nós em uma grade com lado 64. Os pontos sólidos indicam os nós acessados durante a consulta (incluindo o nó onde a mesma foi gerada). Os pontos marcados com um quadrado com um "x" dentro, identificam os demais nós da rede. As linhas sólidas delimitam a região de identificadores pela qual cada nó é responsável. A busca é realizada através das seguintes etapas:

- O nó que inicia a pesquisa calcula qual o menor nível para o qual o próprio nó e a chave são internos ao quadrado desse nível. Verificará que se trata do nível 6 (quadrantes de lado 32). Como o nó encontra-se no quadrante 2 e a chave no quadrante 1 ele repassa a consulta para um dos seus vizinhos de nível 6 (no caso foi escolhido o vizinho vertical).
- Ao receber a consulta esse nó, referenciado na figura como "primeiro nó consultado", refaz o cálculo para verificar qual o menor nível que engloba tanto ele quanto a chave, e em quais quadrantes esses pontos

estão localizados. Verifica que se trata do nível 6 (quadrantes de lado 32) e que o nó está no quadrante 0 e a chave no quadrante 1. Dessa forma, a consulta é repassada para o vizinho horizontal de nível 6.

- Ao receber a consulta, esse nó, referenciado na figura como "segundo nó consultado", refaz o cálculo para verificar qual o menor nível que engloba tanto ele quanto a chave, e em quais quadrantes esses pontos estão localizados. O nó verifica que se trata do nível 5 (quadrantes de lado 16) e que o nó esta no quadrante 0 e a chave no quadrante 1. Dessa forma, a consulta é repassada para o vizinho horizontal de nível 5.
- Ao receber a consulta, esse nó, referenciado na figura como "terceiro nó consultado", refaz o cálculo para verificar qual o menor nível que engloba tanto ele quanto a chave, e em quais quadrantes esses pontos estão localizados. O nó verifica que se trata do nível 4 (quadrantes de lado 8) e que o nó está no quadrante 3 e a chave no quadrante 1. Dessa forma, a consulta é repassada para o vizinho vertical de nível 4.
- Ao receber a consulta, esse nó, referenciado na figura como "quarto nó consultado", refaz o cálculo para verificar qual o menor nível que engloba tanto ele quanto a chave, e em quais quadrantes esses pontos estão localizados. O nó verifica que se trata do nível 3 (quadrantes de lado 4) e que o nó está no quadrante 0 e a chave no quadrante 1. Dessa forma, a consulta é repassada para o vizinho horizontal de nível 3.
- Como esse nó, referenciado na figura como "quinto nó consultado", é o nó responsável pela chave pesquisada, a consulta é encerrada nesse ponto.

3.6 Entrada de Novos Nós na Rede

Pelas próprias características de uma rede P2P, a entrada de novos nós acontece com relativa freqüência e é uma operação de complexidade elevada uma vez que requer a reconfiguração de diversos nós na rede. Para isso, quatro tarefas precisam ser realizadas por um nó n que entra na rede:

- Contatar o nó da rede que terá as chaves divididas e solicitar ao mesmo o intervalo de chaves pelas quais ficará responsável.

- Inicializar sua tabela de apontadores para os nós nos quadrantes vizinhos em cada nível.
- Atualizar os apontadores dos nós da rede que se referem às chaves pelas quais n ficará responsável.
- Notificar o software da camada superior, ou seja, o software utilizando a rede P2P, no nó da rede que terá as chaves divididas para migrar os dados associado às chaves pelas quais *n* ficou responsável.

Para a realização de qualquer operação na rede P2P é necessário, evidentemente, que a mesma seja solicitada a algum nó da rede. Portanto, para um novo nó entrar na rede, faz-se necessário o conhecimento de algum nó existente na mesma. Isso normalmente é feito através da publicação, em algum serviço globalmente acessível, como é o caso do DNS, do nome de algumas máquinas pertencentes à rede e que fiquem conectadas, senão na totalidade, pelo menos na maior parte do tempo.

3.6.1 Cálculo das Chaves

Para entrar na rede, o nó *n* primeiro calcula as coordenadas da grade pelas quais ficará encarregado, utilizando o método de mapeamento do endereço IP explicado anteriormente (ver seção 3.1). Obtido esse valor, *n* precisa descobrir o nó atualmente encarregado por esse ponto. Para isso, basta fazer uma pesquisa utilizando algum nó que esteja publicado no DNS. O nó retornado por essa consulta, *n'*, terá as suas chaves divididas com *n*, de acordo com os procedimentos explicados na seção 3.2. Porém, nesse instante, os dados associados às chaves ainda não são migrados para *n*, apenas os valores das chaves são informados, pois são necessários para calcular os nós que precisam ser apontados e os que precisam ser reconfigurados.

3.6.2 Atualização dos Ponteiros do Novo Nó

Uma vez calculada a coordenada da grade relativa ao novo nó *n* e definidas as chaves pelas quais ficará responsável, *n* precisa calcular os nós dos quadrantes vizinhos em cada nível, para os quais deve manter um apontador. Para isso, *n* calcula as coordenadas desses nós, através dos procedimentos descritos na seção 3.4, e pergunta a *n'* quem são os nós responsáveis por cada uma delas.

Executar uma pesquisa para cada um dos *m* nós apontados leva a uma complexidade de O(m log N) para essa operação de atualização. Porém, como normalmente cada nó fica responsável por um número de chaves maior que 1, pode acontecer de, um mesmo nó, ser o responsável pelas coordenadas dos quadrantes vizinhos para mais de um nível. Um caso especial dessa situação, que ilustra claramente esse fato, é quando o próprio nó é responsável pelas coordenadas vizinhas nos primeiros níveis da hierarquia. Supondo, por exemplo, que um nó qualquer seja responsável pelas chaves que formem uma região quadrada na grade, de lado 8, esse nó também será o responsável pelos vizinhos de níveis 1, 2 e 3, tanto horizontal quanto verticalmente, uma vez que esses pontos estarão dentro do espaço de chaves do próprio nó. Essa característica reduz o valor de m, ou seja, o número de nós apontados, para Log N. Como uma pesquisa também possui complexidade O(log N), a complexidade total para atualização dos apontadores será O(log2 N). Para beneficiar-se dessa característica basta que um nó, ao pesquisar pelo nó responsável pelas coordenadas vizinhas em um determinado nível, solicite ao mesmo que retorne o conjunto de chaves pelas quais é responsável. Assim, poderá verificar se esse nó também é responsável pelas coordenadas vizinhas em outros níveis na hierarquia e evitar pesquisas desnecessárias que retornariam o mesmo nó como resultado.

Uma otimização, semelhante à utilizada pelo Chord, pode ser aplicada ao processo de atualização dos ponteiros de um nó e que reduz a sua complexidade para O(log N). Tal otimização consiste em obter uma cópia da tabela de ponteiros de *n'* e realizar as buscas utilizando essa tabela, de acordo com a seguinte regra: para pesquisar pelo vizinho horizontal de nível *i* para o novo nó n deve-se enviar a busca para o nó que consta como vizinho horizontal de nível *i* na tabela de apontadores (a mesma regra vale para o vizinho vertical). A melhora de desempenho obtida deve-se ao fato de que como esses nós são responsáveis por chaves muito próximas, uma vez que as chaves do nó n foram obtidas pela divisão das chaves de *n'*, provavelmente seus vizinhos em cada nível serão os mesmos, ou pelo menos nós muito próximos um do outro. Evidentemente, mesmo quando os nós não são os mesmos, perguntar pelo vizinho de *n*, em um dado nível, a um nó próximo a ele, ou seja, ao nó constando na tabela de apontadores naquele nível, é bem mais eficiente que fazer a mesma consulta a *n'*.

3.6.3 Atualização dos Ponteiros dos Nós Existentes na Rede

Como uma parte das chaves de *n'* passa a ficar sob a responsabilidade do novo nó *n*, evidentemente os apontadores de alguns outros nós da rede precisam ser reconfigurados de modo a refletir essa mudança. Mais precisamente, os nós que possuem apontadores para nós que tenham como coordenadas, pelas quais são responsáveis, pontos dentro da região das chaves atribuídas a *n*, serão reconfigurados. Conforme poderemos constatar pelos resultados das simulações, o número de nós que precisam ser reconfigurados quando um novo nó entra na rede é O(log2 N), mesmo quando se inclui os nós que precisam apenas ser consultados nesse processo.

Como a divisão do espaço de chaves de *n'* após a entrada do novo nó *n*, pode gerar apenas regiões quadradas (após uma divisão vertical), ou retangulares que possuem o mesmo tamanho (após uma divisão horizontal), podemos considerar que as chaves de *n'* estão dentro de um quadrado de um determinado nível e ocupam exatamente um ou dois quadrantes vizinhos. O processo de atualização será bastante simplificado se utilizarmos como base para o cálculo dos nós que precisam ser atualizados em cada nível o referido quadrado e seus quadrantes internos, através dos seguintes procedimentos:

- Passo1: Calcular as coordenadas iniciais (ponto superior esquerdo) do quadrado de menor nível que englobe as chaves de *n* e *n'*. Chamaremos esse nível de *i* e essas coordenadas de x0,y0.
- Passo2: Verificar qual(is) quadrante(s), dentro do quadrado de nível *i* calculado no passo anterior, representam as chaves de *n*.
- Passo 3: Calcular as coordenadas dos nós vizinhos a x0,y0 para cada um dos níveis, iniciando em *i*, e enviar uma mensagem a cada um deles solicitando que atualizem seus apontadores e repassem a mensagem aos outros nós relativos às chaves especificadas, caso não seja o responsável por elas. Essa especificação das chaves se dá através da informação do tamanho do quadrado, que será o valor de *i* calculado no passo 1, e os quadrantes que precisam ser atualizados. A especificação dos quadrantes segue as seguintes regras:
    - Para a atualização do nível *i*, os quadrantes a serem atualizados são os quadrantes vizinhos ao quadrante calculado no passo 2.

Caso as chaves de *n* ocupem dois quadrantes, então os outros dois quadrantes devem ser atualizados.

  - Para atualização dos níveis maiores que *i*, os quadrantes a serem atualizados são os mesmos que foram calculados no passo 2.

- Passo 4: Após receber um pedido de atualização dos apontadores, um nó precisa fazer duas verificações. Primeiro, caso não seja o responsável pelas chaves de algum dos quadrantes, o nó responsável pelas coordenadas (x0,y0) desse quadrante é pesquisado e a mensagem de atualização é repassada para ele. Neste caso, a nova mensagem deve informar como número do nível para o cálculo do quadrado o valor recebido para esse parâmetro menos 1, e solicitar a atualização de todos os quadrantes. A segunda verificação a ser feita serve para garantir que a atualização é realmente necessária, uma vez que todos os nós responsáveis pelas chaves da grade que são vizinhas, em algum nível, dos pontos da grade relativos às chaves de *n*, recebem as mensagens de atualização. Porém, só precisam ser atualizados os nós cujo ponto da grade para o qual o endereço IP do nó é mapeado é vizinho em algum nível da região de chaves de n. Um exemplo dessa situação pode ser mostrado através da figura 3.2d (pg.33), onde após a entrada do nó 5 na rede, os ponteiros do nó 4 não precisam ser atualizados pois, embora esse nó seja responsável pelas chaves do quadrante superior direito, considerando-se o nível 2, as coordenadas pelas quais está encarregado encontram-se no quadrante superior esquerdo. Assim sendo, mesmo após a entrada do nó 5, o nó 4 permanece tendo como vizinho de nível dois o nó 3.

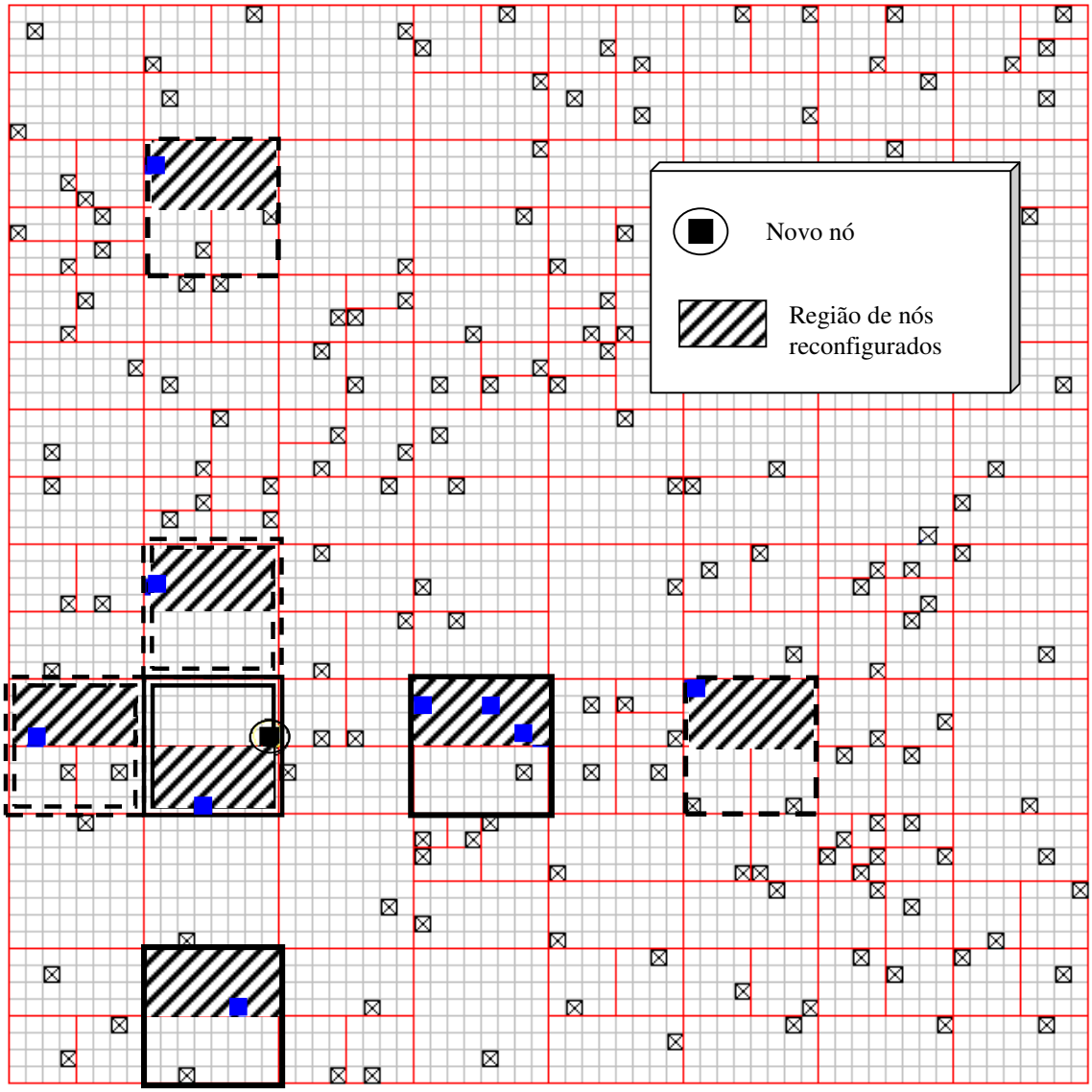


Figura 3-5. Nós que precisam ser atualizados após a entrada de um novo nó.

A figura 3-5 mostra como o esquema do quadrado, e seus quadrantes internos, pode ser usado para identificar os nós que precisam ser reconfigurados após a entrada de um novo nó na rede. Nessa figura o novo nó *n* está identificado com um círculo ao seu redor e obteve suas chaves através de uma divisão horizontal no espaço de chaves do nó *n'*, que é o nó dentro do mesmo quadrado de *n*. Os nós com uma pintura sólida na figura identificam os nós que tiveram seus apontadores, em algum nível, atualizados. O nível exato para o qual o apontador foi alterado, para apontar para *n*, é indicado através do tipo de borda dos quadrados, de acordo com a seguinte identificação: duas linha sólidas indica vizinhos de nível 3, duas linhas pontilhadas indica vizinhos de nível 4, uma linha sólida indica vizinhos de nível 5 e uma linha pontilhada indica vizinhos de nível 6. O quadrado utilizado como base para o processo de atualização possui tamanho de lado 8,

o que indica um quadrado de nível 3, pois este é o quadrado de menor nível que engloba as chaves de *n* e *n'*. Os quadrantes a serem atualizados dentro de cada quadrado estão indicados por linhas diagonais.

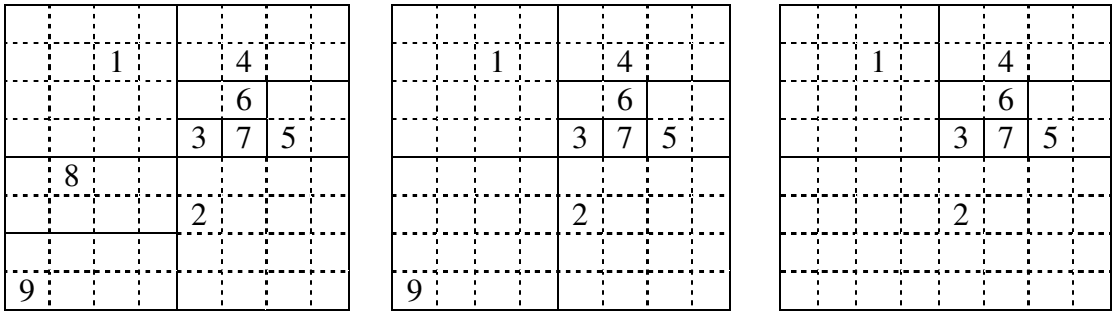
3.6.4 Transferência de Chaves

A última etapa no processo de adição de um novo nó consiste em *n'* transferir para *n* a responsabilidade sobre as chaves para as quais esse nó ficou responsável. Como uma rede P2P é uma rede genérica, os detalhes do que exatamente essa operação deve fazer, dependem da aplicação, mas normalmente corresponde a transferir os dados associados às chaves.

3.7 Saída de Nós da Rede

Quando um nó *n'* deixa a rede os procedimentos a serem realizados são praticamente idênticos aos realizados quando um nó entra na rede. Mais precisamente:

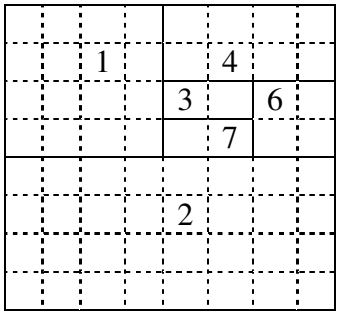
- Identificar o nó n para o qual *n'* deve transferir as suas chaves.
- Atualizar os apontadores dos nós da rede que se referem às chaves pelas quais *n'* transferiu para *n*.
- Notificar o software da camada superior, ou seja, o software utilizando a rede P2P, sendo executado em *n'* para migrar os dados associados as suas chaves para *n*.



a) Configuração inicial antes da saída dos nós.

b) Nova configuração após a saída do nó 8.

c) Nova configuração após a saída do nó 9.



d) Nova configuração após a saída do nó 5.

Figura 3-6. Exemplos de reorganização da rede após a saída de nós.

3.7.1 Identificação do Nó Vizinho

Um fato importante a ser considerado durante as operações de exclusão de nós é que as regiões formadas pelo espaço de chaves de cada nó devem ser retangulares, ou quadradas. Portanto, quando um nó deixa a rede deve passar a responsabilidade de suas chaves para outro nó que mantenha essa condição. Esse fato pode ser observado nas figura 3.6b e 3.6c que ilustram a nova organização das chaves após a saída de nós da rede mostrada na figura 3.6a. A figura 3.6b mostra que após a saída do nó 8, o nó 9 assumiu a responsabilidade pelas suas chaves gerando assim uma região quadrada. A figura 3.6c mostra que após a saída do nó 9, o nó 2 assumiu a responsabilidade pelas suas chaves gerando assim uma região retangular. Vale ressaltar que na figura 3.6b, embora se o nó 1 tivesse adquirido a responsabilidade sobre as chaves do nó 9, uma região retangular também seria formada, o nó 2 deve realmente ser o escolhido. Isso se deve ao fato que para simplificar a cálculo das regiões das chaves, a entrada de um novo nó divide uma região quadrada em duas regiões horizontais, ou divide uma região

horizontal em duas quadradas. Na saída de um nó, o processo inverso deve ser aplicado, ou seja: deve-se unir duas regiões horizontais em uma quadrada, ou duas quadradas em uma horizontal.

Nem sempre, porém, é possível manter as regiões de chaves formando espaços retangulares, ou quadrados, de uma forma simples como mostrado nas figuras 3.6b e 3.6c, reconfigurando apenas o nó vizinho. Para verificar essa situação, consideremos que o nó 5 resolva deixar a rede da figura 3.6c. Nesse caso, não é possível transferir as chaves de 5 para nenhum dos nós vizinhos, ou seja, os nós 2, 4, 6 e 7 , mantendo o espaço de chaves formando uma região conforme o desejado. Em situações como essa um procedimento de reestruturação envolvendo os nós mais próximos do nó deixando a rede, no caso o nó 5, deve ser realizado.

Essa reestruturação consiste na movimentação dos nós que passam a ficar encarregados de outro ponto do espaço, como se o mapeamento de seus endereços IPs tivesse tido como resultado os novos pontos especificados nessa operação. Para isso o nó *n* deixando a rede elegerá um de seus vizinhos *n'*, que será movido para o ponto do espaço dentro da região de chaves de *n* que é mais próximo da região de chaves de *n'*. Uma operação de movimentação é equivalente ao fato de um nó deixar a rede e entrar em outro ponto do espaço da grade. Portanto, se *n'* deixa a rede, deve executar também o procedimento de exclusão, que envolve escolher um vizinho para ficar responsável pelas suas chaves. Embora teoricamente esse processo possa desencadear uma série de exclusões de nós, na prática, apenas um número bem pequeno de nós precisa ser movido até se atingir um vizinho capaz de englobar as chaves do nó que sai da rede de modo a formar uma região retangular ou quadrada. Isso é decorrente do fato dos nós serem responsáveis por regiões de aproximadamente o mesmo tamanho.

A escolha do vizinho *n'* do nó *n* deixando a rede é feita da seguinte forma:

- O nó *n* calcula o menor quadrado de nível que englobe n e algum outro nó.
- Após *n* verificar em qual quadrante ele se encontra, *n'* é escolhido de acordo com as seguintes situações:
    - Caso *n* ocupe apenas um quadrante *n'* será um nó do quadrante vizinho horizontalmente que faça fronteira com o quadrante de *n*. Embora qualquer um desses nós pudesse ser utilizado, *n'* será o nó responsável pela chave do canto superior (esquerdo ou direto, dependendo do lado onde faz fronteira) desse quadrante.

- Caso *n* ocupe dois quadrantes, *n'* será o nó responsável pela chave mais à esquerda da região simétrica às chaves de *n* que faça fronteira com algum desses quadrantes.

De acordo com essas regras, a configuração da rede mostrada na figura 3.6d, que corresponde à reestruturação dos nós após a saída do nó 5 da rede da figura 3.6c, pode ser detalhadamente analisada. Para sair da rede, o nó 5 deve identificar para qual vizinho deve passar as suas chaves. Para isso, esse nó verifica que o menor quadrado de nível que o engloba juntamente com algum outro nó é um quadrado de nível 2 e, portanto, com tamanho de lado 4. Além disso, como suas chaves ocupam apenas o quadrante inferior direito, o nó vizinho a ser escolhido deve ser o nó responsável pela chave do canto superior direito do quadrante inferior esquerdo, ou seja, o nó 6. Como esse nó não é capaz de englobar as chaves do nó 5 mantendo uma região quadrada ou retangular, será movido para o ponto do espaço dentro dessa região de chaves mais próximo de suas coordenadas iniciais.

Como o nó 6 foi movido, ele precisa executar o procedimento de exclusão do nó e descobrir o vizinho que ficará encarregado de suas chaves. Para isso, o nó 6 verifica que o menor quadrado de nível que o engloba juntamente com algum outro nó é um quadrado de nível 1 e, portanto, com tamanho de lado 2. Além disso, como suas chaves ocupam os dois quadrantes superiores, o nó vizinho a ser escolhido deve ser o nó responsável pela chave do canto superior esquerdo do quadrante inferior esquerdo, ou seja, o nó 3. Como esse nó não é capaz de englobar as chaves do nó 6 mantendo uma região quadrada ou retangular, será movido para o ponto do espaço dentro dessa região de chaves mais próximo de suas coordenadas iniciais.

Como o nó 3 foi movido, ele precisa executar o procedimento de exclusão e descobrir o vizinho que ficará encarregado de suas chaves. Para isso, o nó 3 verifica que o menor quadrado de nível que o engloba juntamente com algum outro nó é um quadrado de nível 1 e portanto com tamanho de lado 2. Além disso, como suas chaves ocupam o quadrante inferior esquerdo, o nó vizinho a ser escolhido deve ser o nó responsável pela chave do canto superior esquerdo do quadrante inferior direito, ou seja, o nó 7. Como esse nó é capaz de englobar as chaves do nó 3 e manter uma região retangular, nenhuma movimentação de nós é mais necessária, obtendo-se nesse ponto a configuração final da rede, conforme mostrado na figura 3.6d.

3.7.2 Atualização de Ponteiros e Transferência de Chaves

Quando um nó *n'* deixa a rede e outro nó se apropria de suas chaves, havendo ou não movimentação de nós, se faz necessário, evidentemente, que alguns nós da rede sejam reconfigurados para que seus apontadores reflitam essas mudanças. Entretanto, essa operação é idêntica à atualização decorrente da entrada de novos nós na rede e utiliza os mesmos procedimentos já explicados anteriormente. O mesmo ocorre para a transferência de chaves, onde o processo aplicado na exclusão de um nó é idêntico ao utilizado quando da inserção de novos nós na rede.

4 Arquitetura da Rede SGrid com *Super-Peers*

Tipicamente as redes P2P são compostas de um grande número de máquinas de diferentes localidades e os nós da rede apresentam diferentes capacidades computacionais (processamento e memória) e diferentes tipos de conexões, desde *links* discados e temporários de 56kbps até conexões permanentes de alguns megabits por segundo. Diante dessa heterogeneidade, tratar todos os nós da mesma maneira evidentemente não é a melhor abordagem. Dessa forma, algumas redes P2P atribuem maior responsabilidade aos nós que possuem melhores recursos, chamando-os de *super-peers*. Cada *super-peer* assume responsabilidade sobre os recursos disponibilizados por um conjunto de outros nós. Isso significa que cada *super-peer* responde as pesquisas endereçadas ao conjunto de nós sob sua responsabilidade e realiza o roteamento das mensagens circulando na rede. Essa estratégia melhora o desempenho da rede devido, principalmente, a duas razões. Primeiramente porque evita gargalos gerados por máquinas com recursos limitados (sejam de processamento ou largura de banda). Segundo porque reduz um dos grandes problemas das redes P2P que se refere às mudanças constantes de topologia geradas a cada novo nó que entra na rede, uma vez que essa operação requer que um conjunto de outros nós sejam reconfigurados. Portanto, esconder os nós atrás de *super-peers* reduz bastante esse problema, uma vez que apenas a entrada de um novo *super-peer* é que acarreta mudanças na topologia da rede.

Embora a abordagem de *super-peers* produza uma melhora significativa no desempenho das redes P2P, a única diferença de uma rede P2P convencional, onde todos os nós realizam as mesmas tarefas, para as arquiteturas de *super-peers* atuais limita-se ao fato de que nessas últimas apenas os nós que são *super-peers* participam da rede. Por esse motivo, as arquiteturas de *super-peers atuais* ainda apresentam alguns problemas, entre os quais podemos citar:

- Os *super-peers* são implementados na camada de aplicação, o que aumenta o processamento necessário para o encaminhamento dos pacotes, pois os mesmos precisam ser passados do kernel para a aplicação sendo executada no espaço do usuário. Conseqüentemente, esse fato acarreta uma maior utilização do processador da máquina e implica no aumento do tempo necessário para entrega dos pacotes.

- Os *super-peers* roteiam mensagens que não são destinadas nem a ele próprio nem aos nós sobre os quais está responsável. Isso não apenas sobrecarrega o *super-peer*, como também gera tráfego desnecessário passando por dentro das redes das organizações, sobrecarregando seus *links* de comunicação.
- A entrada de um novo *super-peer* na rede implica na alteração da sua topologia, e conseqüentemente requer a reconfiguração de outros nós na rede.
- Nas arquiteturas de *super-peers* baseadas em redes DHT, apenas os *super-peers* são mapeados para o espaço de identificadores, portanto os nós não se beneficiam de algumas características dessas redes, como, por exemplo, a distribuição homogênea dos recursos entre os nós, que é proporcionada pela utilização das funções *hash*.

Diante desses fatores e de que decorrente da sobrecarga imposta aos *super-peers* não há nenhum interesse por parte dos nós em se tornarem *super-peers* [57]. Por isto, aqui propomos uma arquitetura P2P que emprega um novo conceito/tipo de *super-peer*, chamado *lightware super-peer* (LSP). Para resolver os problemas supra citados o objetivo principal do LSP é separar as funções de roteamento de mensagens das funções de pesquisas endereçadas aos nós sob a responsabilidade do LSP. Essa tarefa é realizada através de uma arquitetura baseada no protocolo NATal (*NAT Application Layer*), que consiste em rotear (e fazer NAT [58]) de acordo com dados da camada de aplicação, onde transferimos o roteamento das mensagens para roteadores com suporte a esse protocolo. Com essa abordagem a única função adicional de um LSP, além das tarefas de um nó comum, é comunicar-se com o roteador para passar as informações a serem utilizadas no processo de mapeamento e roteamento. Essa tarefa consiste em informar a região de chaves pela qual os nós da organização se responsabilizam, além da tabela de roteamento da rede P2P. Cada nó da rede interna informa ao roteador a sub-região, dentro da região de chaves informada pelo LSP, pela qual o nó está responsável. Dessa forma, o roteador encaminhará para cada nó sob a responsabilidade do *super-peer*, as mensagens que lhes são destinadas, sem sobrecarregar o *super-peer*. Além disso, mensagens que não são destinadas a nenhum desses nós são roteadas no próprio roteador sem entrar na rede interna da organização.

Embora exista uma certa centralização de informações no roteador, essa abordagem não segue o modelo cliente-servidor, pois além de existir um número muito alto de roteadores NATal na rede P2P, o tráfego que passa por cada um deles é apenas

uma fração muito pequena do tráfego total da rede. Além disso, sobre o fato de que os roteadores NATal constituem um único ponto de falha, vale ressaltar que a forma como as redes IP são projetadas, onde todo o tráfego da rede passa pelo roteador, já impõe essa vulnerabilidade. Portanto, adicionar novas funcionalidades ao roteador não diminui a disponibilidade da rede, uma vez que, se o roteador ficar inoperante, toda a rede ficará isolada, independente dos serviços adicionais desempenhados pelo mesmo.

4.1 Incorporando LSPs e Roteadores NATal ao SGrid

Nessa seção analisaremos como a arquitetura básica do SGrid, mostrada no Capítulo 3, pode ser modificada para suportar o emprego de *lightware super-peers* (LSPs) e Roteadores NATal.

Como a idéia principal do SGrid consiste em transferir a tarefa de roteamento de mensagens dos *super-peers* para os roteadores, apenas a primeira máquina de cada rede precisa entrar na rede P2P global, tornando-se um LSP. Essa máquina é vista na Internet através do IP do roteador (que utiliza NAT para mascarar os endereços da rede interna, inclusive o do LSP) e propaga sua tabela de roteamento P2P para o mesmo. Como a região de chaves pela qual o LSP responsabiliza-se inclui a região de chaves da organização, todas as pesquisas realizadas por qualquer uma delas serão encaminhadas para o endereço IP do roteador. As demais máquinas da organização apenas registram-se no roteador e não entram na rede P2P global. Esse registro consiste em informar qual o seu endereço IP e quais as chaves pelas quais é responsável. Desse modo, todas as máquinas de uma organização são vistas como uma única máquina na rede P2P global, ou seja, apenas o IP do roteador NATal é conhecido. No que diz respeito aos recursos computacionais disponíveis, tipicamente as máquinas de uma organização são consideradas pela rede P2P como um *cluster*, ou seja, o LSP propaga o valor correspondente à soma dos recursos computacionais de todas as máquinas da rede interna e não os recursos de cada uma individualmente.

Uma vez que o roteador possui uma cópia da tabela de roteamento (da rede P2P) do *super-peer* e conhece os nós internos e suas chaves, ao receber uma mensagem, o roteador verifica se ela se refere às chaves registradas. Em caso positivo, a mensagem é encaminhada diretamente ao nó da rede interna responsável pela chave, sem passar pelo *super-peer*. Em caso negativo, o roteador consulta sua tabela de roteamento (vinda do

super-peer) e encaminha a mensagem para o nó especificado pela mesma, sem que a mensagem entre na rede interna da empresa. A Figura 4.1 ilustra uma rede com um roteador e quatro nós, onde um deles é um LSP. Os valores a_x,b_x,c_x,d_x representam a região retangular de chaves pelas quais cada nó é responsável.

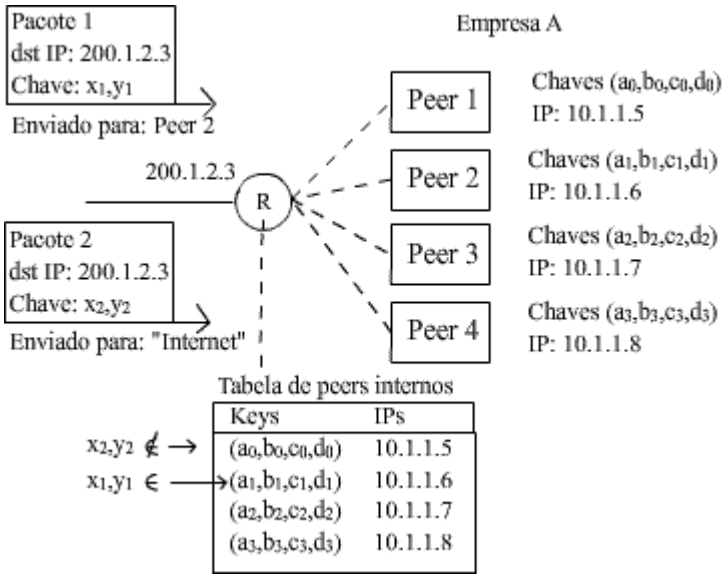


Figura 4-1. Roteamento utilizando um Roteador NATal.

Quatro observações importantes precisam ser feitas a respeito da Figura 4.1:

- O processo de repasse das mensagens para a rede interna utiliza NAT para trocar o endereço de destino das mensagens, uma vez que as mesmas são destinadas ao IP do roteador e não dos nós internos.
- Todas as mensagens enviadas pelos nós internos para a rede P2P global são mascaradas (através de NAT) com o IP do roteador. Isso é válido inclusive para o LSP.
- O roteador possui duas tabelas. A tabela de nós internos com suas respectivas chaves (mostrada na figura) e a tabela de roteamento da rede P2P global, contendo os vizinhos horizontal e vertical em cada nível (essa tabela não é mostrada na figura 4.1).
- O LSP é a única máquina que fará parte da rede P2P global. Esse nó será visível através do IP utilizado pelo roteador para fazer o NAT. No caso da Figura 4.1 esse IP é o 200.1.2.3. Sempre que a sua tabela de roteamento é modificada devido a alterações na topologia da rede P2P, essas modificações são propagadas para o roteador.

Ainda na Figura 4.1, pode-se observar que, ao receber o pacote 1, o roteador o encaminha diretamente ao Peer 2 pois, embora esse pacote esteja endereçado ao IP do roteador, refere-se à chave (x_1,y_1) que pertence ao conjunto de chaves (a_1,b_1,c_1,d_1) registrado pelo Peer 2. Ao receber o pacote 2, o roteador verifica que o mesmo refere-se a chave (x_2,y_2) que não foi registrada por nenhum nó interno. Portanto, após consultar sua tabela de roteamento (não mostrada na figura), encaminha o pacote para o nó especificado na mesma e que está fora da rede da organização.

Evidentemente o tipo de roteamento proposto exige mais do roteador do que simplesmente analisar endereços IPs e portas. As três novas funções que o roteador precisa implementar são: (1) roteamento e NAT baseados nos dados da camada de aplicação (NATal); (2) implementação da tabela de roteamento da rede P2P (que lhe é passada pelo *super-peer*); (3) suporte ao recebimento de atualizações dessa tabela pela rede. Roteadores com suporte ao protocolo NATal são chamados, desse ponto em diante, apenas de NR (*NATal Routers*). O SGrid define o protocolo NATal, que especifica tanto o formato das mensagens de atualização dessa tabela quanto a forma como o roteamento e o NAT devem ser executados.

Um requisito importante a ser considerado em eventuais propostas para utilizar esse modelo em outras arquiteturas de rede P2P diferentes do SGrid é que os nós dentro da rede sob o roteador NATal devem ser responsáveis por chaves consecutivas no espaço de chaves, de modo que possam ser vistas como um único conjunto de chaves pelos outros nós na rede P2P. No SGrid nós pertencentes a redes diferentes mapeiam para regiões diferentes do espaço de chaves, e nós de uma mesma rede mapeiam para pontos diferentes porém pertencentes a uma mesma região. Dessa forma, não existe interseção entre a região de chaves de duas organizações diferentes. Quando um nó de uma organização A entra na rede, torna-se responsável por todas as chaves da organização. Cada novo nó da organização A que entra na rede obtém uma parte dessas chaves. Portanto, esse primeiro nó a entrar na rede torna-se o *super-peer* e avisa tanto ao roteador da empresa quanto à rede P2P que ele é o encarregado do intervalo completo de chaves da empresa (entretanto, o IP propagado será o do roteador). Como a partir desse ponto cada novo nó da empresa A ao entrar na rede obtém apenas um subconjunto das chaves, que para a rede P2P já estão sob a responsabilidade do IP do roteador, é necessário alterar apenas a configuração dos mapeamentos nesse roteador.

Dentro da empresa, os nós mantém informações a respeito de quem é o *super-peer* e de alguns outros nós. Caso o *super-peer* deixe a rede, algum outro nó assume o seu lugar. Como o IP com o qual esse *super-peer* é conhecido na rede P2P é o IP do roteador, novamente não há necessidade de qualquer reconfiguração na rede P2P, apenas os mapeamentos do roteador local devem ser atualizados. Desse modo, se em cada instante do tempo existir pelo menos uma máquina da empresa na rede P2P, nenhuma alteração de topologia ocorre após a entrada da primeira máquina na rede, obtendo-se, assim, uma topologia constante, no que diz respeito às máquinas dessa rede. Como esse comportamento deve ser apresentado em muitas outras redes na Internet que participam da rede P2P, isso leva a uma topologia relativamente estável.

4.2 Operação da Rede na Presença de Roteadores NATal

Como ao se utilizar roteadores NATal e LSPs no SGrid apenas o primeiro nó de uma organização a entrar na rede é que faz parte da rede P2P global, tornando-se um LSP, os nós podem apresentar dois tipos de comportamento diferentes. Basicamente existe uma rede global composta apenas de LSP, com os demais nós "escondidos" atrás dos roteadores NATal. Os LSPs seguem praticamente os mesmos procedimentos descritos no Capítulo 3, além de precisar se comunicar com o roteador para repassar a sua tabela de roteamento. Por outro lado, o comportamento dos demais nós da mesma rede interna do LSP é baseado no registro e troca de informações com o roteador NATal. Nessa seção descrevemos o comportamento dos LSPs e dos demais nós.

Diversas operações requerem a comunicação dos nós internos com o NR (*NATal Router*) e para isso esses nós precisam conhecer o endereço IP de tal roteador. Essa tarefa pode ser realizada de várias maneiras, entre as quais podemos citar: configuração estática na aplicação, parâmetro passado através do DHCP [60] , publicação no DNS da empresa, utilização do SLP [61] e utilização de Multicast/Broadcast. Além desses, desenvolvemos um método de descoberta que se beneficia do fato de que todo o tráfego gerado pelos nós da rede interna passa pelo NR para atingir a Internet. Dessa forma, basta que os nós enviem a requisição solicitando o endereço IP do NR para um endereço pré-determinado que seja fora do escopo de endereçamento da empresa e que o NR intercepte essa requisição e responda com o seu próprio endereço IP.

4.2.1 Mapeamento dos Nós para Pontos da Grade

Repassar a tarefa de roteamento para o NR evita que tráfego desnecessário entre na rede da organização. Entretanto, o grande benefício proporcionado pelo SGrid é permitir que todas as máquinas da organização sejam vistas como um único IP, pois isso significa que existirá apenas uma única referência (ponteiro) nos outros nós da rede P2P que englobará todas essas máquinas.

Isso só é possível devido ao esquema de mapeamento dos nós para pontos da grade e alocação das chaves do SGrid, que foi inspirado no modelo de *supernets* [26] do endereçamento IP, e possui a característica de que todos os nós da rede interna de uma organização são mapeados para regiões adjacentes na grade, além de garantir que não há interseção dessas regiões com regiões de outras empresas. Mais especificamente, o esquema funciona da seguinte forma:

- Cada endereço IP público diferente (os endereços dos NRs) mapeia para uma região diferente da grade, ou seja, não existe interseção entre o espaço de chaves de dois endereços públicos diferentes.
- Cada endereço IP público mapeia para uma região da grade e não para um único ponto. Desse modo, todos os nós (inclusive os LSPs) das redes internas de uma empresa são mapeados para pontos dessa região.

O SGrid trabalha com uma grade com tamanho de lado 2^{32}, o que significa que existem 32 níveis hierárquicos. De acordo com o esquema explicado na seção 3.1, se dividirmos um endereço Ipv4, que possui 32 bits, em 16 grupos de 2 bits, com cada grupo determinando um quadrante, podemos mapear 16 níveis da hierarquia, ou seja determinamos a região para a qual o endereço IP é mapeado. Como existem 32 níveis na hierarquia e foram percorridos apenas 16, essa região é composta por um quadrado que ainda possui 16 níveis, e que, portanto, possui tamanho de lado 2^{16}.

O esquema utilizado aplica esse método ao IP do NR e obtém, portanto, uma região do espaço de chaves. Os IPs da rede interna são mapeados para pontos dentro dessa região. Como cada IP público mapeia para uma região diferente, não existe possibilidade de dois NR, nem conseqüentemente de dois nós de redes internas, mapearem para o mesmo ponto.

Embora pudesse ter sido utilizado o esquema de mapeamento hierárquico da seção 3.1 para mapear os nós dentro da região da organização, como os endereços das

redes de uma mesma empresa normalmente são muito semelhantes, por exemplo, usando uma classe A privada, eles seriam mapeados para pontos próximos dentro dessa região, não havendo uma distribuição homogênea dos nós pelo espaço de identificadores. Essa distribuição é importante para fazer com que cada nó fique responsável por aproximadamente o mesmo número de chaves, uma vez que o mapeamento dos dados para esse mesmo espaço utiliza uma função *hash* que possui essa característica.

Por essa razão são utilizados dois esquemas de mapeamento diferentes, um para o NR e outro para os nós, que são mapeados aplicando-se uma função *hash*, como SHA ao seu endereço IP. O resultado é considerado a partir da coordenada inicial do espaço de chaves da empresa que foi calculada aplicando-se o método hierárquico ao endereço IP do NR, conforme explicado anteriormente. A Figura 4.2 ilustra uma grade com tamanho de lado 16 e que conseqüentemente possui 4 níveis. Para simplificar a explicação suponhamos que o endereço IP 200.1.2.3 do NR possua apenas os seguintes 4 bits: 0110 (embora fosse possível dividir os 32 bits em 2 grupos de 16 bits). Desse modo, os dois primeiros bits (01) indicam o quadrante superior direito (no nível 4) e os próximos dois bits (10) indicam o quadrante inferior esquerdo (no nível 3). Isso resulta na área sombreada indicando a região para onde o IP do NR foi mapeado. O ponto marcado com "x" representa a coordenada que será propagada na rede P2P para o LSP. Para o nó da rede interna 10.1.2.3 o mapeamento é feito calculando-se a função *hash* nesse endereço e obtendo o módulo desse resultado por 16, que é a área dessa região. O valor final é calculado com relação à coordenada inicial da região, ou seja, "x".

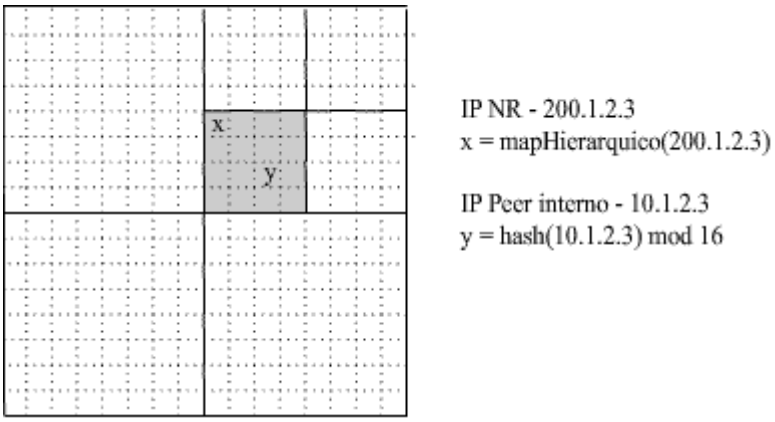


Figura 4-2. Mapeamento do NR e dos nós internos.

4.2.2 Distribuição das Chaves aos Nós

Conforme explicado na seção 3.2, cada nó é associado a uma posição da grade e fica encarregado da chave que é mapeada para essa mesma coordenada. Quando um novo nó entra na rede ele contata o nó responsável pela coordenada da grade para a qual seu endereço IP mapeia e solicita ao mesmo que divida as suas chaves.

Embora esse modelo pudesse ser adotado sem nenhuma modificação para a arquitetura com LSP e NR, uma pequena alteração no comportamento do LSP é aplicada para evitar que nós internos precisem ser reconfigurados quando novos nós/LSP forem adicionados na rede P2P global. Ao entrar na rede, um LSP fica encarregado por uma região de chaves que é no mínimo igual à região de chaves da organização (para a qual seu endereço IP mapeia), ou seja, essa região pode abranger o espaço de chaves de outras empresas. Na figura 4.2, por exemplo, embora a região de chaves para a qual o IP do NR 200.1.2.3 (e conseqüentemente do LSP) mapeia seja a região sombreada, esse LSP pode ficar encarregado da região composta pelos quatro quadrados de área 16, dependendo dos nós existentes na rede na hora em que ele se junta à mesma. Nesse caso, a medida que os LSPs das outras empresas fossem entrando na rede essa região seria divida até o ponto onde o LSP 200.1.2.3 ficasse encarregado apenas da região sombreada.

O problema decorre do fato de que o LSP é mapeado para um ponto dentro da região de chaves da empresa e tem que dividir suas chaves com outros nós da rede interna. Se o LSP realizar esse processo considerando a região inteira de chaves pelas quais é responsável, quando um LSP de outra empresa entrar na rede os nós da rede interna podem precisar ser reconfigurados, pois podem estar com a responsabilidade sobre as chaves dessa empresa. Isso vai contra um dos objetivos do SGrid que é procurar manter a topologia da rede estável. Portanto, a pequena modificação citada anteriormente refere-se apenas a fazer com que o LSP diferencie o conjunto de chaves completo pelo qual é responsável (e que pode incluir chaves de outras empresas) do conjunto de chaves da própria empresa, dividindo com os nós da rede interna apenas as chaves relativas a esse último. Com essa abordagem, novos LSPs podem entrar na rede P2P global sem que qualquer modificação seja necessária nas redes internas.

4.2.3 Mapeamento dos Dados para Pontos da Grade

O mapeamento dos dados para pontos da grade é feito exatamente da mesma maneira conforme explicado na seção 3.3, ou seja, aplicando-se uma função *hash* aos dados.

Vale ressaltar que embora o cálculo do ponto para onde os nós das redes internas são mapeados seja feito tomando por base a região de chaves da empresa, cada nó é responsável pela coordenada considerando a grade por inteiro. Na Figura 4.2, por exemplo, embora o ponto y referente ao nó 10.1.2.3 seja calculado como o valor (2,2), dentro da região de chaves da empresa, na verdade esse nó é responsável pela chave (10,10) que é essa mesma coordenada relativa à grade inteira. Desse modo, a distribuição dos dados é feita considerando todos os nós da rede, incluindo os das redes internas que não são vistos pelos nós de outras organizações. Essa abordagem difere da utilizada por muitas redes com *super-peers* onde apenas os *super-peers* responsabilizam-se pelos dados. O modelo do SGrid permite a participação efetiva de um número qualquer de nós na rede sem que isso acarrete grandes mudanças na sua topologia.

4.2.4 Conexões Entre os Nós

Conforme citado anteriormente, apenas os LSPs participam da rede P2P global. Cada LSP mantém apontadores para outros LSPs exatamente da mesma maneira que em uma rede SGrid simples, conforme explicado na seção 3.4, e essa tabela de roteamento é propagada para o NR.

O fato de que o NR possui os registros de todos os nós da rede interna pertencentes à rede P2P faz com os que esses nós não precisem manter nenhuma estrutura com informações a respeito uns dos outros. Os nós internos obtêm todas as informações que precisam a partir do NR.

Alguns outros modelos foram considerados para o caso onde existem muitas redes por trás do NR e se deseje diminuir a carga no mesmo. As duas principais abordagens consideradas foram: 1) O NR ou o LSP propagarem periodicamente sua tabela de roteamento para todos os nós da rede interna, 2) Os nós internos criarem uma rede P2P interna seguindo o mesmo modelo de apontadores para os nós vizinhos. Vale ressaltar, entretanto, que mesmo utilizando algum desses métodos qualquer novo nó que entra na rede interna continua precisando registrar-se no roteador.

4.2.5 Algoritmo de Busca (Roteamento de Mensagens)

Os nós das redes internas (com exceção do LSP que possui uma tabela de roteamento) enviam suas pesquisas para o endereço IP do NR. Esses roteadores encaminham os pacotes recebidos, vindos da rede interna ou da Internet, de acordo com suas tabelas internas, ou seja, a tabela de roteamento da rede P2P global e a tabela contendo o registro dos nós da rede interna. O NR guarda também os valores da região de chaves para as quais o LSP é responsável, que engloba a região de chaves da organização (para a qual podem existir nós internos registrados) e eventualmente regiões de chaves de outras empresas que podem estar sob a responsabilidade do LSP. A tabela utilizada será determinada pelo fato da chave pesquisada estar ou não dentro da região de chaves do LSP. Desse modo, ao receber um pacote relacionado a uma operação de pesquisa, o NR analisa a chave pesquisada (no cabeçalho da aplicação) e determina qual das duas situações deve ocorrer:

- A chave pesquisada não está dentro da região de chaves do LSP. Nesse caso o NR utiliza a tabela de roteamento da rede P2P, que contém apontadores para os vizinhos em cada nível, para encaminhar o pacote seguindo o mesmo procedimento de uma rede SGrid sem NRs e LSPs, conforme explicado na seção 3.5.
- A chave pesquisada está dentro da região de chaves do LSP. Nesse caso o NR procura na lista de nós internos registrados, se algum nó é responsável por essa chave. Em caso positivo repassa a pesquisa para o referido nó e caso contrário repassa a pesquisa para o LSP.

Em ambos os casos o NR realiza NAT durante a operação de roteamento. Na primeira situação, onde os pacotes são roteados para dentro da rede interna, o endereço de destino do pacote é alterado, modificando-o para o endereço do nó na rede interna. Na outra situação, onde o pacote é roteado utilizando a tabela de roteamento P2P global, tanto o endereço de origem quanto o de destino do pacote são modificados. O endereço de destino será alterado para o endereço do próximo nó, conforme especificado na tabela de roteamento, e o endereço de origem do pacote é alterado para o endereço do próprio NR. Isso se faz necessário para evitar problemas de *spoofing* [66], pois se não fosse feito, haveria pacotes deixando uma rede com endereço de origem de outra rede.

Esses pacotes provavelmente seriam descartados pelos mecanismos de proteção contra *spoofing* que normalmente são utilizados pelos roteadores/*firewalls* na Internet.

Evidentemente após um pacote passar por diversos NR na rede P2P e chegar ao destino, o endereço de origem nele não é o do nó que gerou a pesquisa, mas sim do último NR pelo qual o pacote passou. Para que o nó de destino possa responder diretamente para o nó que iniciou a busca é necessário salvar o endereço desse nó que gerou a busca na parte de dados do pacote, de modo que existe um campo no cabeçalho da camada de aplicação do protocolo SGrid com essa finalidade. O nó que gera uma consulta preenche esse campo com o valor 0 (zero) e os roteadores NR verificam o valor desse campo ao realizarem NAT no endereço de origem. Caso o valor seja 0 (zero), o NR insere o seu endereço IP nesse campo. Caso já exista algum endereço IP nesse campo ele é mantido. Com essa estratégia apenas o primeiro NR no caminho dos pacotes, ou seja, o NR da empresa onde o nó que gerou a consulta está localizado, irá alterar esse campo. Nenhum dos demais NR pelos quais o pacote passa, modifica o valor desse campo. Quando um nó recebe um pacote ele analisa o valor desse campo. Se for zero ele responde para o endereço IP de origem da conexão e se for diferente de zero ele ignora o IP de origem da conexão e responde para o endereço contido nesse campo.

A Figura 4.3 mostra um exemplo de uma grade que será utilizada para ilustrar o processo de busca. Essa grade possui 4 níveis onde o mapeamento dos endereços IP dos NR utilizou apenas dois níveis. Desse modo a região de chaves de cada organização é composta por um quadrado de lado 4. As áreas sombreadas são exemplos dessas regiões para os NRs: R_1, R_2, $R3_3$ e R_4. Os pontos indicados pelas letras de “a”, “b”, “c”, “d”, “e”, “f”, “g”, “h”, “i”, “j”, “k” e “m” representam os nós das redes internas a cada NR e o ponto para o qual eles são mapeados (calculado usando uma função *hash*). As letras em negrito e itálico representam os LSPs e as linhas sólidas delimitam a região de chaves de cada nó. Nessa figura pode-se observar que embora alguns LSP sejam responsáveis por uma região de chaves maior que a região de chaves da empresa, a divisão de chaves com os nós da rede interna é feita considerando-se apenas essa região. Isso ocorre para os LSPs “*e”* e “*h”*.

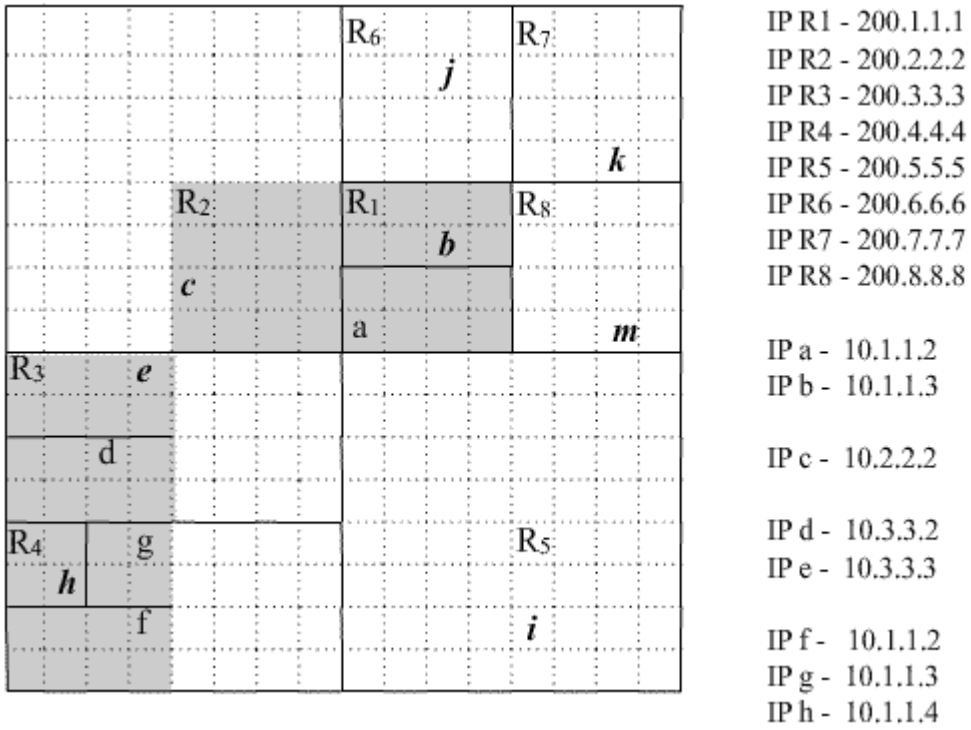


Figura 4.3. Roteamento através dos NRs.

A Tabela 4.1 mostra as tabelas internas de cada NR e os valores referentes às duas regiões de chaves para as quais mantém informações: um para a região de chaves referente apenas às chaves da empresa e outro considerando as chaves da rede P2P global, ou seja, as chaves sob a responsabilidade do LSP. Caso o nó "a" realize uma pesquisa pela chave (0,11), o ponto acima de R4, ocorrem os seguintes procedimentos:

- O nó "a" envia a pesquisa diretamente para o IP do seu NR, no caso, R1. O campo que armazena o IP do remetente da mensagem no cabeçalho de aplicação é preenchido com 0 (zero).
- R1 verifica que a chave pesquisada (0,11) não faz parte da região de chaves sob a responsabilidade do LSP (8,4-11,7) e, portanto, utiliza a tabela de roteamento P2P global para encaminhar o pacote, trocando o IP de destino para 200.2.2.2 (R2). O endereço de origem do pacote também é alterado para o IP do NR1 (200.1.1.1). Como o campo de endereço no cabeçalho de aplicação contém o valor 0, o endereço do próprio NR1 (200.1.1.1.) é colocado nesse campo.
- Ao receber o pacote, R2 verifica que a chave pesquisada (0,11) não faz parte da região de chaves sob a responsabilidade do LSP (0,0-7,7) e, portanto, utiliza a tabela de roteamento P2P global para encaminhar o pacote, trocando o IP de destino para 200.4.4.4 (R4). O endereço de origem do pacote também é alterado

para o IP do NR2 (200.2.2.2). Como o campo de endereço no cabeçalho de aplicação contém um valor diferente de 0 (contém o IP 200.1.1.1, preenchido no passo anterior), esse campo não é alterado.

- Ao receber o pacote, R4 verifica que a chave pesquisada (0,11) não faz parte da região de chaves sob a responsabilidade do LSP (0,12-7,15) e, portanto, utiliza a tabela de roteamento P2P global para encaminhar o pacote, trocando o IP de destino para 200.3.3.3 (R3). O endereço de origem do pacote também é alterado para o IP do NR4 (200.4.4.4). Como o campo de endereço no cabeçalho de aplicação contém um valor diferente de 0 (contém o IP 200.1.1.1), esse campo não é alterado.
- Ao receber o pacote, R3 verifica que a chave pesquisada (0,11) faz parte da região de chaves sob a responsabilidade do LSP (0,8-7,11) e, portanto, utiliza a tabela de registro dos nós internos para encaminhar o pacote. Como essa tabela mostra que a chave (0,11) está sob a responsabilidade do nó interno "d" (10.3.3.2), o endereço de destino do pacote é alterado para esse valor. Uma vez que o fluxo do pacote foi da interface externa do NR para a rede interna, não é feito NAT no endereço de origem, que permanece 200.4.4.4, nem o campo de endereço no cabeçalho da aplicação é alterado, permanecendo 200.1.1.1.
- O nó "d" recebe o pacote processa a informação envia a resposta para o IP 200.1.1.1, uma vez que o campo de endereço no cabeçalho da aplicação é diferente de 0 e contém esse endereço.

Para que a resposta enviada por "d" consiga chegar até "a", uma vez que será enviada para R1 (200.1.1.1), o seguinte mecanismo é utilizado: além da coordenada da grade relativa à chave pesquisada, o remetente de uma mensagem adiciona ao cabeçalho a coordenada da grade pela qual é responsável. Ao responder uma pesquisa o nó destino inverte a ordem desses campos, de modo que o NAT para as mensagens de resposta analisam a coordenada equivalente ao ponto onde o nó se encontra. Evidentemente existe uma entrada na tabela de registros de nós internos do NR associando o nó a essa coordenada/chave. A seção 5.2 detalha os formatos das mensagens. Para o exemplo acima, ao enviar a pesquisa, o nó "a" insere além do valor da chaves pesquisada (0,11) o valor da coordenada para a qual ele mapeia (8,7). O nó "d" ao responder a mensagem inverte a ordem desses campos, de modo que quando essa mensagem chegar em R1 esse NR irá pesquisar em suas tabelas internas pelo nó responsável pela chave (8,7) e não

(0,11). Como, evidentemente, o registro das chaves do nó "a" engloba esse valor (8,6-11-7), R1 sabe que o pacote deve ser entre para esse nó.

Tabela 4-1. Tabelas internas dos NRs da Figura 4.3.

NR	Tabela de nós da rede interna registrados		Tabela de roteamento P2P global		
	Chaves	IP (peer)	Nível	Tipo	IP
NR1 Região de Chaves sob a responsabilidade do LSP: (8,4-11,7) Região de Chaves da empresa: (8,4-11,7)	8,4-11,5 8,6-11-7	10.1.1.3 (b) 10.1.1.2 (a)	3 3 4 4	V H V H	200.6.6.6 (R6) 200.8.8.8 (R8) 200.2.2.2 (R2) 200.5.5.5 (R5)
NR2 Região de Chaves sob a responsabilidade do LSP: (0,0-7,7) Região de Chaves da empresa: (4,4-7,7)	4,4-7,7	10.2.2.2 (c)	4 4	V H	200.4.4.4 (R4) 200.8.8.8 (R8)
NR3 Região de Chaves sob a responsabilidade do LSP: (0,8-7,11) Região de Chaves da empresa: (0,8-3,11)	(0,8-3,9) (0,9-3,11)	10.3.3.3 (e) 10.3.3.2 (d)	3 4 4	V V H	200.4.4.4 (R4) 200.2.2.2 (R2) 200.5.5.5 (R5)
NR4 Região de Chaves sob a responsabilidade do LSP: (0,12-7,15) Região de Chaves da empresa: (0,12-3,15)	(0,12-1,13) (2,12-3,13) (0,14-3,15)	10.1.1.4 (h) 10.1.1.3 (g) 10.1.1.2 (f)	3 4 4	V V H	200.3.3.3 (R3) 200.2.2.2 (R2) 200.5.5.5 (R5)

Caso não se deseje ter os nós das redes internas enviando suas consultas diretamente para o NR, para diminuir a carga sobre o mesmo, o LSP poderia propagar sua tabela de roteamento para esses nós sempre que houvesse uma alteração na mesma.

Desse modo, os nós internos poderiam enviar suas pesquisas diretamente para algum nó na Internet, conforme especificado pela tabela de roteamento.

A mesma idéia poderia ser utilizada para que os nós internos tomassem conhecimento uns dos outros, ou seja, o NR propagaria a tabela de nós registrados para todos os nós da rede interna. Entretanto, devido à natureza homogênea com que os dados são alocados aos nós, a maior parte das pesquisas é destinada a nós na Internet. Diante dessa característica a complexidade adicional que o procedimento de propagação da tabela acarretaria, devido à necessidade de implementar novas operações, e o tráfego adicional gerado, decorrente desse processo, inviabilizam a utilização dessa técnica.

4.2.6 Entrada de Novos Nós na Rede

Quando um novo nó deseja entrar na rede SGrid ele deve perguntar ao NR quem é o nó da rede interna responsável pelas coordenadas do espaço de identificadores para a qual esse novo nó mapeia. Existem duas possibilidades:

- Caso a resposta seja "nenhum" isso significa que não existe nenhum nó das redes por trás desse NR na rede SGrid. Portanto, ele se torna um LSP e entra na rede P2P global. Esse procedimento é idêntico ao da rede SGrid sem os NRs e LSP, conforme explicado na seção 3.6. Após criar a sua tabela de roteamento o novo LSP se registra no NR e propaga essa tabela para o mesmo, informando também o espaço de chaves pelo qual é responsável.
- Caso o NR informe o endereço IP do nó responsável pela coordenada solicitada, isso significa que já existe algum nó da rede interna pertencendo à rede P2P global, e, portanto, já existe um LSP. Desse modo, o nó entrando na rede solicita ao nó retornado pelo NR que divida as suas chaves com ele. Após isso o novo nó registra-se no NR.

4.2.6.1 Cálculo das Chaves

O cálculo das chaves que cada nó fica responsável é idêntico ao utilizado na rede SGrid sem NR e LSP, conforme explicado na seção 3.2, onde cada nó fica responsável por uma região quadrada ou retangular.

4.2.6.2 Atualização dos Ponteiros do Novo Nó

Caso o novo nó entrando na rede seja um LSP ele criará a sua tabela de roteamento P2P, contendo os apontadores para nós vizinhos em todos os níveis, utilizando o mesmo procedimento de uma rede SGrid sem NR e LSP, conforme explicado na seção 3.6.2.

Se o novo nó não for um LSP ele não precisará possuir a tabela de roteamento, uma vez que direcionará todas as suas pesquisas para o NR.

4.2.6.3 Atualização dos Ponteiros dos Outros Nós Existentes na Rede

Caso o novo nó entrando na rede seja um LSP ele ficará responsável por uma região de chaves até então sob a responsabilidade de outro LSP, de modo que isso causa uma mudança da topologia da rede, e, portanto, os apontadores das tabelas de roteamento de outros LSPs precisam ser atualizados. Esse procedimento é realizado utilizando o mesmo método para a rede SGrid sem LSP, conforme explicado na seção 3.6.3.

Se o novo nó não for um LSP, a sua entrada na rede não causa nenhuma mudança de topologia na rede P2P global. Isso acontece porque essa rede é composta apenas de LSPs e cada LSP ao entrar na mesma se responsabiliza por uma região da grade que engloba todas as chaves da empresa. Desse modo, a forma como essas chaves são distribuídas internamente não diz respeito aos outros LSP da rede global. Além disso, como os nós internos não mantêm informações a respeito uns dos outros, nenhuma atualização de ponteiros em outros nós é necessária.

4.2.6.4 Transferência de Chaves

Esse processo ocorre exatamente da mesma forma que em uma rede SGrid sem NR e LSP, conforme explicado na seção 3.6.4, independente do novo nó ser ou não um LSP.

4.2.6.5 Saída de Nós da Rede

Quando um nó deseja deixar a rede, pode acontecer uma das três situações seguintes:

- O nó não é um LSP. Nesse caso o nó deixando a rede deve identificar o nó vizinho para o qual deve transferir suas chaves. Isso é feito de acordo com o método explicado na seção 3.7.1, porém utilizando o NR para saber qual nó da rede interna é responsável pelas coordenadas desejadas. Após a transferência das chaves para esse nó, o nó deixando a rede cancela o seu registro no NR, e o nó que recebeu as chaves atualiza seu registro no NR para informar a nova região de chaves sobre a qual ficou responsável. Nenhuma alteração na rede P2P global (outros LSPs) é necessária.
- O nó é um LSP, mas existe(m) outro(s) nó(s) registrado(s) no mesmo NR que esse LSP. O mesmo procedimento do passo anterior é realizado, entretanto um novo nó da rede interna precisa ser promovido a LSP. Ao se registrarem no NR os nós fornecem informações a respeito de suas capacidades (memória, CPU, links). O LSP periodicamente solicita essas informações ao NR e constrói uma lista hierárquica, com os nós mais indicados a se tornarem LSP. Dessa forma ao deixar a rede o LSP pode escolher qual nó deve se tornar o novo LSP. A tabela de roteamento da rede P2P global é transferida para esse nó e o NR é avisado dessa mudança. Novamente, nenhuma alteração na rede P2P global (outros LSPs) é necessária.
- O nó é um LSP e é o único nó registrado no NR. O nó cancela o seu registro no NR, e executa o mesmo procedimento descrito na seção 3.7.1. Nesse caso, há uma reconfiguração na rede P2P global.

No que diz respeito a reconfiguração dos apontadores de outros nós para refletir a nova divisão do espaço de chaves essa operação só é necessária no terceiro caso citado acima, pois nos dois primeiros as modificações são apenas internas e, conforme citado anteriormente, esses nós não mantêm informações a respeito uns dos outros. Além disso, a alteração desses ponteiros segue o mesmo procedimento utilizado na adição de novos nós, conforme explicado na seção 3.6.3.

4.3 O Problema do NAT nas Redes P2P

O modelo de endereçamento original da Internet foi projetado de modo que cada máquina na rede mundial possui um endereço IP diferente, ou seja, o endereço de cada máquina é único na rede. Desse modo, as máquinas podem enviar pacotes diretamente umas para as outras. Esses endereços são chamados de *endereços IP públicos*. Diante do crescimento da rede e do problema crescente da escassez de endereços IPv4, que possuem apenas 32 bits, esse padrão de endereçamento foi cedendo espaço para um novo modelo, que atualmente tornou-se o *padrão de fato*, onde redes diferentes podem utilizar os mesmos endereços IP. Existem faixas de endereços reservadas para essa finalidade e que são conhecidas como *endereços IP privados*. Esse nome deve-se ao fato de que, evidentemente, esses endereços são conhecidos apenas dentro das redes internas das organizações, uma vez que, para uma máquina ser acessível globalmente seu endereço continua precisando ser único.

Para que máquinas com endereços privados possam acessar outras máquinas na Internet que possuem endereços públicos, um esquema especial de tradução de endereços foi desenvolvido, chamado NAT (*Network Address Translation*), que permite aos roteadores alterarem dinamicamente os endereços dos pacotes. Essa alteração consiste em trocar o endereço privado por um endereço público ao enviar o pacote para a Internet, e fazer o processo inverso, ou seja, trocar o endereço público para um privado, ao receber as mensagens de resposta retornando da Internet. Esse mecanismo permite que várias máquinas com endereços privados compartilhem um único endereço público para acessar a Internet, utilizando para isso a tradução tanto dos endereços IP dos pacotes quanto dos números das portas. Ao enviar um pacote para a Internet o roteador realizando NAT cria o que se chama de sessão. Uma sessão consiste na identificação dos pontos em comunicação, ou seja, no par (IP/porta/protocolo) de origem e (IP/porta/protocolo) de destino. Cada sessão é associada às alterações feitas no pacote enviado para que o processo possa ser desfeito nos pacotes de retorno.

Como as sessões são criadas apenas quando pacotes são enviados da rede privada para a pública isso significa que apenas as máquinas da rede privada podem iniciar as comunicações. Não há como uma máquina da rede pública enviar dados para uma máquina da rede privada, sem que essa última tenha enviado antes algum pacote destinado para ela, pois não haverá nenhuma sessão criada no roteador. Nesse caso o pacote seria descartado pelo mesmo. Por essa razão, embora esse esquema de NAT seja

adequado ao modelo cliente-servidor, onde uma máquina na rede privada acessa um servidor na rede pública, ele dificulta bastante a comunicação quando as duas máquinas envolvidas estão em redes privadas diferentes. Evidentemente, as aplicações P2P enquadram-se nesse caso onde, normalmente, as máquinas em comunicação estão atrás de roteadores realizando NAT. Esse problema é conhecido como *transversal NAT* [59].

Embora o protocolo IPv6 possa reduzir o motivo original que gerou a necessidade de NAT, ou seja, a escassez de endereços IP, uma vez que endereços IPv6 possuem 20 bytes, atualmente existem outras razões para a utilização do mesmo. As principais são: segurança proporcionada pelo fato de proteger as máquinas da rede contra acessos externos, e o fato de esconder a identificação das máquinas acessando a rede externa, uma vez que para as máquinas sendo acessadas o endereço de origem dos pacotes é o endereço do roteador realizando NAT.

Nessa seção a principal técnica de transversal NAT utilizada atualmente é apresentada e comparada com a abordagem adotada pelo SGrid para resolver o mesmo problema.

4.3.1 Hole Punching

As principais técnicas utilizadas para resolver o problema do transversal NAT utilizam servidores intermediários mediando a comunicação entre os nós nas redes privadas. Como dois exemplos dessas técnicas podemos citar:

- *Relay*. Essa é a técnica mais segura porém, a menos eficiente, sendo utilizada quando os dois nós estão em redes privadas diferentes. A técnica consiste em utilizar o servidor intermediário para reencaminhar as mensagens entre os nós, ou seja, as mensagens são enviadas para o servidor que as repassa para o outro nó. Além de consumir muito processamento do servidor intermediário, essa estratégia utiliza muita banda de rede e gera um tempo alto de transmissão.
- *Conexão reversa*. Quando apenas um dos nós envolvidos na comunicação está em uma rede privada, uma técnica simples pode ser utilizada. Quando o nó na rede pública deseja comunicar-se com o nó na rede privada, ele solicita ao servidor intermediário para informar ao nó na rede privada (através da conexão que esse nó mantém com o servidor) que deseja se comunicar com ele. Nesse ponto, o nó na rede privada abre

uma conexão com o nó na rede pública sem nenhum problema, uma vez que essa é a situação para a qual o NAT foi desenvolvido. Evidentemente, essa técnica só pode ser aplicada quando uma das máquinas está na rede privada e a outra na rede pública, não se adequando à situação mais comum onde as duas máquinas encontram-se em redes privadas.

Diante das limitações dessas duas técnicas, um novo método foi desenvolvido, chamado *hole punching* [59], e é atualmente o método mais utilizado para a comunicação entre máquinas de diferentes redes privadas. Embora seja mais amplamente utilizado com o protocolo UDP também é possível utilizá-lo com o TCP.

Basicamente essa técnica faz com que dois nós em redes privadas diferentes, descubram, através de um servidor intermediário, o endereço público para o qual o NAT traduz o endereço do outro nó, e enviem pacotes um para o outro simultaneamente. Isso criará sessões idênticas nos roteadores NAT utilizados por cada um, permitindo que pacotes subseqüentes sejam recebidos utilizando esses mapeamentos.

Para o funcionamento do *UDP Hole Punching* os dois nós que desejam comunicar-se devem manter sessões ativas com o servidor intermediário. Esse servidor mantém informações a respeito dos *endpoints* (IP/Porta) privado e público de cada cliente. O *endpoint* privado é obtido durante o processo de registro do nó no servidor intermediário, onde o nó informa em um campo dentro do pacote, qual o endereço IP e porta usados para comunicação. O *endpoint* público consiste do IP e Porta (de origem) contidos no cabeçalho IP do pacote recebido pelo servidor, e permite ao mesmo identificar como o nó é visto na Internet após a aplicação do NAT, caso seja aplicado para o cliente.

A primeira tarefa do *hole punching* consiste em fazer com que os dois nós em comunicação criem entradas nas tabelas NAT dos roteadores utilizados por cada um para acessar a Internet. Supondo que um nó A deseje estabelecer uma sessão UDP diretamente com o nó B, e ambos possuem uma sessão com o servidor intermediário S, os seguintes passos são realizados:

- Como A inicialmente não sabe como encontrar B, ele solicita a S que o ajude nessa tarefa.
- S responde para A com uma mensagem contendo os *endpoints* público e privado de B. Além disso, S usa sua sessão com B para enviar uma

requisição de conexão contendo os *endpoints* público e privado de A. Desse modo, cada nó conhecerá os *endpoints* público e privado do outro.

- Cada nó envia pacotes UDP para os *endpoints* público e privado do outro e utilizará o que responder primeiro com uma mensagem válida. Como nesse ponto já existia um mapeamento do *endpoint* privado do nó para um *endpoint* público, uma vez que o mesmo já havia se comunicado com o servidor, todos os pacotes gerados com esse mesmo *endpoint* privado são traduzidos para o mesmo *endpoint* público utilizado anteriormente, ou seja, durante a comunicação com o servidor.

A utilização do *endpoint* privado só é necessária porque os dois nós podem estar atrás do mesmo NAT e a outra técnica utilizada para resolver essa situação, *Hairpin*[59], ainda não é largamente implementada pelos roteadores.

A Figura 4.4 ilustra a utilização do *hole punching* quando os dois nós estão em redes privadas diferentes. A Figura 4.4a mostra para quais valores os endereços e portas são alterados pelos NATs. O *endpoint* privado do nó A é 10.1.1.1:5555 e o público é 200.1.1.1:5000. O *endpoint* privado do nó B é 10.2.2.2:6666 e o público é 200.2.2.2:6000. Quando cada nó registra-se no servidor S esses *endpoints* são propagados para o outro nó. Nesse instante o nó A envia pacotes UDP para os endereços 10.2.2.2:6666 e 200.2.2.2:6000. Os pacotes enviados para 200.2.2.2:6000 criam a seguinte sessão no NAT1: 200.1.1.1:5000 – 200.2.2.2:6000. O nó B, por sua vez, envia pacotes UDP para os endereços 10.1.1.1:5555 e 200.1.1.1:5000. Os pacotes enviados para 200.1.1.1:5000 criam a seguinte sessão no NAT2: 200.2.2.2:6000 - 200.1.1.1:5000. Se um pacote enviado ao *endpoint* público chegar primeiro no NAT relacionado a esse *endpoint* antes que o pacote enviado pelo nó interno a esse NAT tenha passado pelo mesmo, o pacote será descartado, pois ainda não haverá uma entrada criada para essa sessão. Para a Figura 4.4, um exemplo dessa situação ocorreria se o pacote enviado pelo nó B para 200.1.1.1:5000 chegasse ao NAT1 antes que pacote enviado pelo nó A para 200.2.2.2:6000 passasse pelo mesmo NAT1. Entretanto, como o nó interno está mais próximo de seu NAT que o nó externo, isso provavelmente não ocorrerá. Caso ocorra, apenas o(s) primeiro(s) pacotes serão descartados, pois assim que o pacote enviado a partir da rede interna passar pelo NAT a sessão será criada e os pacotes externos serão aceitos. A Figura 4.4b mostra a configuração dos NATs após o *hole punching*

estabelecido. Ambos os NATs possuem a sessão 200.1.1.1:5000 – 200.2.2.2:6000 criada, e portanto os nós podem comunicar-se diretamente.

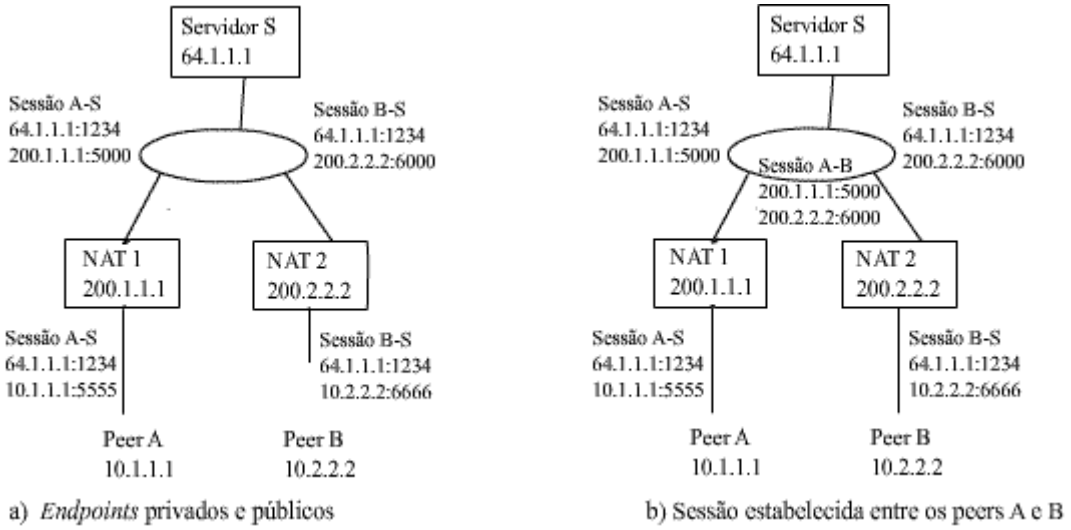


Figura 4-3. *Hole Punching*.

4.3.2 Comparando o Hole Punching com o Protocolo NATal

Para que o protocolo *Hole Punching* funcione adequadamente as implementações de NAT devem seguir um comportamento padronizado a respeito de algumas características. Em [59] são descritas algumas dessas características, como: (1) tradução consistente das portas de origem; (2) controle das conexões TCP não solicitadas; (3) não realização de traduções de endereços IP na parte de dados dos pacotes; (4) tradução de *loopback*. A principal delas, tradução consistente das portas de origem, refere-se ao fato de que se já houve uma tradução para um determinado *endpoint* (IP/Porta) de origem, todos os outros pacotes gerados a partir desse *endpoint* devem ser traduzidos da mesma forma, mesmo que sejam endereçados para um *endpoint* de destino diferente. Desse modo, se o *endpoint* de origem $IP_a/Porta_x$ de um pacote destinado ao *endpoint* $IP_b/Porta_y$ foi traduzido para $IP_c/Porta_z$ qualquer outro pacote com *endpoint* de origem $IP_a/Porta_x$ destinado a qualquer *endpoint* de destino, por exemplo, $IP_d/Porta_w$ tem que ser traduzido para $IP_c/Porta_z$.

Diferenças na implementação dessas características limitam a utilização do *hole punching*. Um estudo apresentado em [59], mostrou que 82% das implementações de NAT testadas suportam *hole punching* para UDP e apenas 62% suportam *hole punching* para TCP. Um outro problema que pode ocorrer com essa técnica é que devido ao fato

de que também são enviados pacotes para os *endpoints* privados, que possuem endereços que podem ser utilizados por várias redes, pode existir alguma máquina na rede interna da empresa que possua o mesmo endereço privado do nó com o qual se deseja comunicar. Além disso, o *hole punching* requer a utilização de um servidor intermediário bem-conhecido, o que impõe um certo nível de centralização e gera, portanto, problemas de desempenho e confiabilidade.

Como na arquitetura do SGrid com a utilização dos NRs e LSPs os pacotes são sempre enviados para o endereço dos NR, que são públicos, e repassados para os nós internos através da análise dos dados do cabeçalho de aplicação, o problema do transversal NAT não existe. Isso decore do fato de que os nós internos não são vistos na Internet, pois o que determina o nó da rede interna para o qual a mensagem será entregue não é o seu endereço IP, mas sim o valor da chave pesquisada. Entretanto, essas considerações a respeito do NAT no SGrid são válidas apenas porque o protocolo de transporte utilizado nessa arquitetura é o UDP, de modo que não poderiam ser aplicadas caso o protocolo utilizado fosse o TCP porque os pacotes de estabelecimento de conexão não carregam dados da aplicação.

4.4 Expandindo as Funções dos NRs

Pelo fato das primeiras redes P2P serem utilizadas para compartilhamento de arquivos e essas aplicações consumirem muita largura de banda criou-se uma cultura, por parte de muitos administradores de rede, de bloquear a utilização dessas aplicações. Isso fez com que as arquiteturas de redes P2P passassem a procurar esconder o seu modo de funcionamento na tentativa de burlar os mecanismos de proteção. Uma das principais técnicas utilizadas é variar as portas utilizadas para receber as solicitações. Entretanto, diante de técnicas de filtragem mais elaboradas, como, por exemplo, a análise do conteúdo dos pacotes, é possível descobrir como qualquer modelo de rede P2P funciona e realizar o bloqueio da aplicação, se assim for desejado. Desse modo, complicar e esconder a arquitetura da rede, normalmente apenas retarda o processo de bloqueio.

Diante da grande variedade de aplicações, além de compartilhamento de arquivos, que podem ser desenvolvidas utilizando as redes P2P, provavelmente essas redes farão parte do ambiente de qualquer empresa. Desse modo, o SGrid optou por adotar uma arquitetura clara e simples. Além das vantagens dessa arquitetura, citadas

anteriormente nesse texto, o modelo de utilização dos NRs permite que novas funcionalidades sejam incorporadas em versões futuras. Entre as quais podemos citar:

- Desenvolvimento de módulos de monitoramento para o NR, onde se possa analisar e contabilizar o tráfego consumido por cada nó.
- Desenvolvimento de módulos de autenticação e controle de banda permitindo restringir o uso desse tipo de aplicação a determinados usuários, além de restringir a banda de rede que cada um pode utilizar.
- Desenvolvimento de um mecanismo de *cache* comum a todos os nós da rede interna.
- Desenvolvimento de um módulo de *firewall.*
- Tornar o NR um *gateway* entre o SGrid e outras arquiteturas diferentes. Desse modo, apenas o NR precisaria conhecer e fazer parte das outras redes e ainda assim, os nós poderiam realizar pesquisas nas mesmas.

Embora algumas dessas funcionalidades possam ser desenvolvidas como uma aplicação separada, a implementação integrada com o NR fornece mais possibilidades em termos de recursos e informações disponíveis. Determinar as rotas mais utilizadas da tabela de roteamento do LSP, por exemplo, não poderia ser feito simplesmente capturando os pacotes em alguma máquina intermediária, pois é necessária a análise da tabela de roteamento.

4.5 Usando Super-Peers Convencionais no SGrid

Na arquitetura de *super-peers* do SGrid, com exceção do LSP, a única função dos demais nós da rede é responder as pesquisas pelas chaves que são responsáveis, pois não realizam roteamento nem mantêm informações a respeito de outros nós na rede P2P global. Desse modo, os requisitos necessários em termos dos recursos computacionais que cada nó precisa ter para realizar suas tarefas na rede P2P são bastante reduzidos, quando comparados com as arquiteturas P2P convencionais. Além disso, uma consideração importante a respeito da capacidade dos nós, é que por não realizam roteamento, não há como um nó comum prejudicar o desempenho da rede, pois mesmo que possua poucos recursos computacionais e esteja sobrecarregado isso afetará apenas as buscas pelas chaves que esse nó é responsável.

Entretanto, caso existam máquinas com poucos recursos computacionais e se permita que as mesmas utilizem a rede P2P sem que isso comprometa nem mesmo as buscas pelas chaves sob a responsabilidade desse nó, alguns nós da rede interna poderiam assumir o papel de *super-peers* para outros nós da mesma organização. A única diferença desse modelo para o modelo de *super-peers* do SGrid, com LSPs e NRs, é que os nós que desejassem utilizar um *super-peer* não se registrariam no NR. No lugar dessa operação, registrariam os recursos que desejassem disponibilizar para a rede P2P no *super-peer*, que os trataria como se fossem seus próprios recursos. A escolha do *super-peer* ao qual um nó registraria seus recursos poderia ser feita usando o modelo de mapeamento do endereço IP para pontos da grade, ou seja, o nó registraria-se no nó interno que fosse responsável pelo ponto da grade para a qual o mesmo mapeia.

5 Especificação dos Protocolos NATal e SGrid

Para permitir que a rede SGrid possa operar tanto no modelo de *super-peers*, com a utilização dos LSPs e NRs, quanto no modo convencional, onde todos os nós que participam da rede realizam procedimento idênticos para a operação da mesma, a especificação da rede foi dividida em dois protocolos:

- *Protocolo SGrid*. Especifica todas as mensagens relacionadas à criação e operação da rede SGrid, que não estão relacionadas à arquitetura de *super-peers*. A arquitetura subjacente de tal protocolo foi descrita no capítulo 3. Dentre as principais operações especificadas por esse protocolo, podemos citar: entrada de nós na rede P2P, saída de nós da rede P2P e pesquisa por uma determinada chave.
- *Protocolo NATal*. Especifica todas as mensagens relacionadas especificamente ao modelo de *super-peers*, o que corresponde à comunicação entre os nós da rede interna e o NR.

5.1 Protocolo SGrid

O protocolo SGrid utiliza como camada de transporte o protocolo UDP e segue o modelo cliente-servidor, entretanto cada nó desempenha as tarefas de cliente e servidor simultaneamente, com o servidor escutando na porta 1972. Embora as requisições dos clientes sejam enviadas utilizando uma porta aleatória maior ou igual a 1024, as respostas não são devolvidas por esse mesmo *socket* (ip origem / porta origem – ip destino / porta destino). A Figura 5.1 ilustra esse modelo.

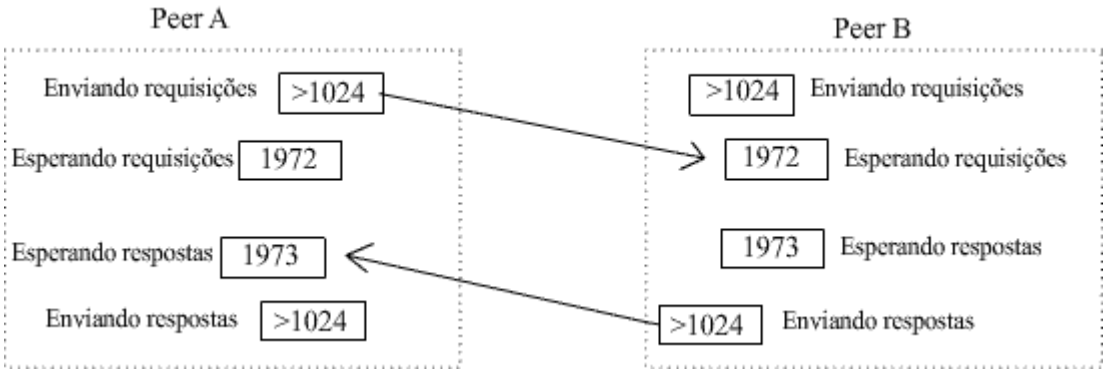


Figura 5-1. Modelo de comunicação entre os nós.

Para que uma aplicação possa associar as respostas chegando na porta 1973 com as requisições enviadas, as mensagens possuem um campo de identificação cujo valor é enviado de volta nas mensagens de resposta. Embora essa abordagem seja um pouco mais complexa do que o modelo tradicional onde as respostas são enviadas para a mesma porta utilizada para as requisições, ela é justifica por três razões principais:

- IP *Spoofing*. Inicialmente as requisições são enviadas para um endereço que normalmente não é o do nó responsável pela chave desejada, de modo que o pacote terá o endereço de destino alterado diversas vezes antes de atingir o destino correto. Isso significa que, quando esse nó enviar a resposta, o endereço de origem não corresponderá ao endereço de destino enviado na requisição. Portanto, mesmo que a resposta fosse enviada para a porta utilizada na requisição, ela seria descartada, pois não corresponderia ao *socket* utilizado pelo cliente. Uma possível estratégia seria armazenar o endereço de destino original dentro do pacote que será utilizado, como endereço de origem, pelo nó final ao responder a mensagem. Essa abordagem, entretanto, significa gerar pacotes com endereço de origem de uma rede a qual o nó não pertence, ou seja, realizar *spoofing* de IP. Conseqüentemente, esse pacote provavelmente seria filtrado e descartado por algum roteador no caminho de volta.
- Simplificação da implementação do NAT no NR. Caso as requisições dos clientes utilizassem portas aleatórias, o NR precisaria criar uma regra de NAT dinamicamente associada a cada uma delas para que as respostas pudessem ser devolvidas ao nó e aplicação corretos, como fazem as implementações de NAT convencionais. Isso não só aumenta a complexidade para o processamento dos pacotes como também requer a utilização de mais memória do NR. Ao utilizar uma porta única para todas as respostas e deixar que a identificação da aplicação para a qual o pacote de destina seja realizada pelo próprio nó de destino do pacote, o SGrid permite que o NR determine qual é esse nó sem precisar criar mapeamentos dinâmicos, utilizando apenas a sua tabela de registro dos nós internos. Isso é possível uma vez que essa tabela contém as chaves sob a responsabilidade de cada nó e os pacotes de resposta possuem

campos indicando a coordenada/chave para a qual o nó que enviou a solicitação é mapeado. Ver seção 4.2.5.

- Desacoplamento do código da rede SGrid do código da aplicação. Fazer com que o cliente escute em uma porta fixa permite que várias aplicações na mesma máquina compartilhem uma só instância da implementação da rede P2P. Na redes P2P atuais se existem duas aplicações na mesma máquina usando um mesmo tipo de rede P2P, cada uma possui uma implementação da mesma sendo executada separadamente. Evidentemente, mesmo compartilhando uma implementação comum da rede P2P, as aplicações podem participar tanto de uma mesma rede P2P, quanto de redes diferentes. Além disso, a rede deve saber quais chaves estão sob a responsabilidade de qual aplicação. O modelo do SGrid foi projetado para suportar essa características em versões futuras do protocolo. Na versão atual esse recurso ainda não está completamente especificado. Entretanto, a adaptação do modelo atual, conforme mostrado na Figura 5.1, para suportar várias aplicações requer basicamente que cada uma delas se registre na implementação da rede SGrid local. Ao fazer isso cada uma receberia um código de aplicação que seria enviado nas mensagens de requisição e devolvido nas mensagens de resposta. Assim, os pacotes chegando na porta 1973 poderiam ser demultiplexados e entregues à aplicação correta.

5.1.1 Formato das Mensagens

Nessa seção são descritos os formatos das mensagens e qual a finalidade de cada uma. Todas as mensagens possuem como um de seus campos o código da operação, que serve para indicar o tipo de mensagem. A Tabela 5.1 mostra os códigos de todas as mensagens do protocolo SGrid, onde pode-se observar que mensagens de resposta possuem códigos diferentes das mensagens de requisição. Os textos na coluna “Nome Abreviado” são utilizados ao longo desse texto para simplificar as referências a cada uma das mensagens. Todos os campos das mensagens são transmitidos em “*network byte order*”. Além disso, em diversas mensagens existe um campo “reservado” que é utilizado nesse texto para tornar o tamanho das mensagens múltiplo de 32 bytes, o que

simplifica a visualização do formato das mesmas. Em uma implementação real esses campos não precisam existir.

Tabela 5-1. Códigos de operação do protocolo SGrid.

Código	Significado	Nome Abreviado
1	Pesquisar pelo nó responsável por uma chave	SgridSearch
2	Resposta da mensagem anterior (1)	SgridSearchResp
3	Inserir um nó na rede	SgridJoin
4	Resposta da mensagem anterior (3)	SgridJoinResp
5	Atualizar apontadores dos nós em um quadrante da grade	SgridUpdatePtrQuad
6	Resposta da mensagem anterior (5)	SgridUpdatePtrQuadResp
7	Reconfigurar um nó	SgridReconfigurePeer
8	Resposta da mensagem anterior (7)	SgridReconfigurePeerResp

Embora cada operação possua um formato de mensagem de resposta específico, existem dois campos que são comuns a todas. O primeiro é o campo indicando o código de operação, e o segundo é um campo que serve para indicar o status da operação, ou seja, se foi ou não bem sucedida. O código de operação será preenchido de acordo com a operação a qual está relacionada. A Figura 5.2 ilustra esses dois campos.

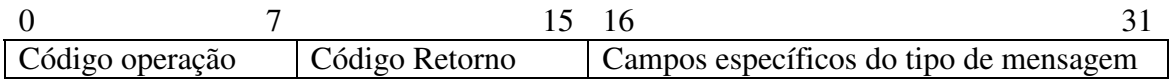


Figura 5-2. Campos comuns a todas as mensagens de resposta Sgrid.

A Tabela 5.2 mostra os possíveis códigos de Retorno relacionados às diversas operações da Tabela 5.1.

Tabela 5-2. Códigos de retorno das operações do protocolo SGrid.

Código de Retorno	Significado
0	Sucesso na realização da operação
1	Falha na realização da operação
2	Erro: parâmetros inválidos
3	Não existem no nó informações associadas à chave
4	Nó não é responsável pela chave informada

5.1.1.1 Mensagem para Pesquisar pelo Nó Responsável por uma Chave

A mensagem para pesquisar pelo nó responsável por uma determinada chave, mostrada na Figura 5.3, utiliza o código 1. Os campos *Xchave* e *Ychave* indicam a coordenada da chave a ser pesquisada. Os campos *Xcliente* e *Ycliente* indicam a coordenada da grade para a qual o cliente é mapeado e são copiados para as mensagens de resposta, para que o NR possa identificar o nó interno que deve receber a mensagem (ver seção 4.2.5). O campo IP cliente é utilizado para que o destino final possa identificar o remetente original da mensagem após as operações de NAT no endereço de origem que os NR realizam. Esse campo é preenchido com zero pelo nó enviando a requisição e será alterado pelo primeiro NR no caminho, conforme foi explicado na seção 4.2.5. Conforme comentado anteriormente, o campo identificador é utilizado para o cliente associar as requisições com as respostas, uma vez que o nó de destino apenas copia esse valor na mensagem de resposta. Cada nó mantém internamente um contador, que é utilizado para preencher esse campo, e é incrementado a cada nova mensagem enviada.

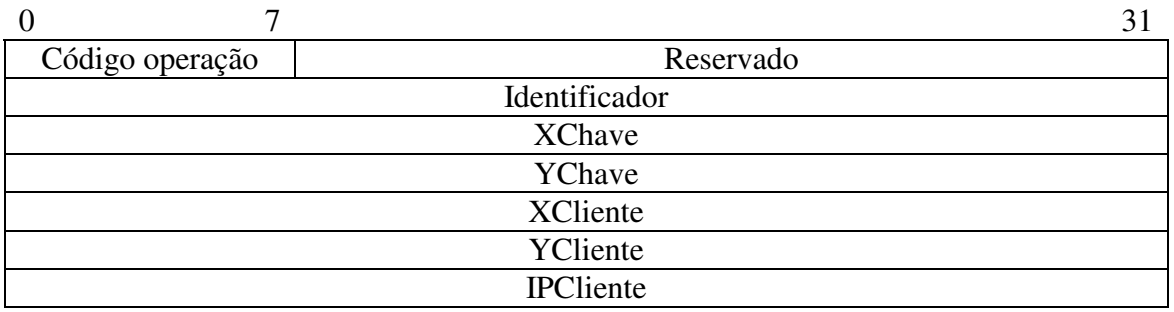


Figura 5-3. Formato da mensagem SgridSearch.

A mensagem de resposta indicando o nó responsável por uma determinada chave é mostrada na Figura 5.4 e utiliza o código de operação 2. Como qualquer chave dentro do espaço da grade tem que estar sob a responsabilidade de algum nó, a operação será sempre bem sucedida. Entretanto isso não significa que o nó responsável pela chave possua dados associados a ela. Como esse fato deve ser informado à aplicação que enviou a pesquisa, os possíveis códigos de retorno são: 0 – significando que o nó retornado possui informações relacionadas à chave e 3 indicando que não existem

informações associadas à chave no referido nó. Além disso, o código de erro genérico 1 pode ser utilizado quando algum outro tipo de problema ocorrer.

Como as mensagens de resposta não sofrem NAT no endereço de origem, pois são enviadas diretamente para o NR associado ao nó que fez a solicitação (e não para os NRs intermediários), o endereço do nó responsável pela chave é obtido através do endereço de origem do pacote. O campo identificador possui o mesmo valor existente no campo de mesmo nome na mensagem de requisição.

Os campos *Xcliente, Ycliente, XChave* e *YChave*, contêm os valores recebidos nesses campos na mensagem de requisição. Entretanto, a ordem desses campos é trocada nas mensagens de requisição e resposta, de modo que o NR para o qual a mensagem de resposta é endereçada possa, mais facilmente, determinar, através da sua tabela de registro dos nós da rede interna, qual o nó que deve recebê-la. Na realidade o que possibilita o correto encaminhamento das mensagens por parte do NR é o fato de existirem os campos com as coordenadas dos clientes e não o fato de estarem invertidos. Essa inversão é feita apenas para simplificar a implementação dos procedimentos no NR.

0	7	31
Código operação	Código Retorno	Reservado
Identificador		
Xcliente		
Ycliente		
Xchave		
Ychave		

Figura 5-4. Formato da mensagem SgridSearchResp.

5.1.1.2 Mensagem para um Nó Entrar na Rede

Para entrar na rede um nó calcula a coordenada da grade pela qual ficará responsável, de acordo com a função de mapeamento do endereço IP, e após descobrir o nó atualmente responsável por ela solicita ao mesmo que divida a sua região de chaves. A mensagem para solicitar essa divisão de chaves é *SgridJoin* e é ilustrada na Figura 5.5. O código de operação é 3 e os campos X e Y indicam a coordenada para a qual o novo nó foi mapeado. O campo identificador serve para associar requisições com mensagens de resposta.

0	7	31
Código operação	Reservado	
Identificador		
X		
Y		

Figura 5-5. Formato da mensagem SgridJoin.

O nó que recebe a mensagem *SgridJoin* processa a requisição e após dividir sua região de chaves com o novo nó, responde com uma mensagem *SgridJoinResp*. O formato dessa mensagem é ilustrado na Figura 5.6. O código de operação é 4 e os códigos de retorno são: 0 – operação realizada com sucesso, e 4 - o nó não é responsável pela chave informada, indicando portanto, que nenhuma operação foi realizada. Além disso, o código de erro genérico 1 pode ser utilizado quando algum outro tipo de problema ocorrer. O campo identificador tem a mesma função citada nas outras mensagens, ou seja, associar as requisições com as respostas. Os campos X e Y indicam a coordenada pela qual o novo nó é responsável, que podem ou não ser os mesmos valores enviados na requisição, uma vez que a operação de entrada de um nó pode deslocar as coordenadas obtidas através do mapeamento IP (ver seção 3.2). Os campos x0, y0, x1 e y1 indicam a região de chaves pela qual o novo nó ficou responsável, onde (x0,y0) indicam a coordenada superior esquerda dessa região e (x1,y1) indicam a coordenada inferior direita. Os demais campos são utilizados para auxiliar no processo de construção da tabela de apontadores do novo nó, indicando os vizinhos em determinados níveis. O campo "Num. Entradas" indica quantos vizinhos estão contidos na mensagem, ou seja, o número de triplas (Tipo – nível – IP). O campo *Tipo* pode conter os valores ´H´ ou ´V´ indicando se a entrada se refere a um vizinho horizontal ou vertical. O campo Nível indica a que nível o vizinho se refere e o campo IP contém o endereço IP do vizinho.

0	7	31
Código operação	Código Retorno	Reservado
Identificador		
X		
Y		
X0		
Y0		
X1		
Y1		
Num. Entradas	Tipo	Nível
IP		

Figura 5-6. Formato da mensagem SgridJoinResp.

5.1.1.3 Mensagem de Atualização da Tabela de Apontadores dos Nós Contidos em uma Região da Grade

Após qualquer alteração na região de chaves de um nó, as tabelas de apontadores de outros nós da rede precisam ser reconfigurados para refletir essa mudança. As alterações ocorrem sempre que um nó entra ou deixa a rede e são realizadas de acordo com o algoritmo explicado na seção 3.6.3. A mensagem para solicitar a alteração na tabela de apontadores é *SgridUpdatePtrQuad* e está ilustrada na Figura 5.7. O código de operação é 5 e o campo identificador é utilizado para associar requisições com as respostas. Os campos X0 e Y0 indicam a coordenada superior esquerda da região cujos nós devem ter seus apontadores atualizados. Essa região é formada por uma área quadrada cujo lado possui como tamanho o valor especificado no campo "Tamanho de Lado". O campo *Quadrantes* é um campo de bits onde os quatro bits menos significativos especificam os quadrantes da região que devem ser reconfigurados. Um bit definido como 1 significa que os nós do quadrante devem ser atualizados. Os campos *Tipo, Nível* e *IP* seguem o esquema usual para atualização da tabela de vizinhos e significam, portanto: vizinho horizontal ou vertical, nível ao qual o vizinho se refere e endereço do nó vizinho, respectivamente.

Figura 5-7. Formato da mensagem SgridUpdatePtrQuad.

A mensagem de resposta indica apenas se a operação foi bem sucedida, através dos códigos de retorno 0 – para sucesso e 1 para falha. O campo de identificação possui o comportamento usual de associar requisições a respostas. O formato dessa mensagem, que utiliza 6 como código de operação, é mostrado na Figura 5.8.

0	7	31
Código operação	Código Retorno	Reservado
Identificador		

Figura 5-8. Formato da mensagem SgridUpdatePtrQuadResp.

5.1.1.4 Mensagem de Reconfiguração da Região de Chaves de um Nó

Durante o processo de exclusão de um nó da rede, um ou mais nós precisam ser reconfigurados para reorganizar o espaço de chaves, conforme explicado na seção 3.7. Para solicitar a um nó que altere sua região de chaves e possivelmente reconfigure sua tabela de apontadores, a mensagem *SgridPeerReconfigure* é utilizada. Seu formato está ilustrado na Figura 5.9 e o código de operação utilizado é 7.

0	7	31
Código operação	Tipo Reconf.	Reservado
Identificador		
X0		
Y0		
X1		
Y1		

Figura 5-9. Formato da mensagem SgridReconfigurePeer.

O campo *identificador* serve para associar as requisições com as respostas. Os campos x0, y0, x1 e y1 indicam a nova região de chaves pela qual o novo nó deve ficar responsável, onde (x0,y0) indicam a coordenada superior esquerda dessa região e (x1,y1) indicam a coordenada inferior direita. O campo “Tipo Reconf.” indica o tipo de reconfiguração sendo realizada e se refere ao relacionamento existente entre as antigas chaves do nó e as novas chaves pela qual ficará responsável. Essa informação é importante tanto para verificar se há necessidade de alterar o ponto da grade pela qual o nó é responsável, quanto para determinar quais entradas da tabela de apontadores precisam ser reconfiguradas. A Tabela 5.3 ilustra os possíveis valores para esse campo e seus significados.

Tabela 5-3. Códigos referentes ao tipo de reconfiguração do nó.

Código	Descrição
1	Região de chaves do Nó foi expandida para englobar a região adjacente a DIREITA, de mesmo tamanho que a sua.
2	Região de chaves do Nó foi expandida para englobar a região adjacente a ESQUERDA, de mesmo tamanho que a sua.
3	Região de chaves do Nó foi expandida para englobar a região adjacente ACIMA, de mesmo tamanho que a sua.
4	Região de chaves do Nó foi expandida para englobar a região adjacente ABAIXO, de mesmo tamanho que a sua.
5	Nó ficou responsável pela região de chaves do nó que está deixando a rede, ou de um nó que foi reconfigurado devido a outro nó ter deixado a rede. Essa nova região de chaves faz fronteira HORIZONTALMENTE com a sua antiga região de chaves.
6	Nó ficou responsável pela região de chaves do nó que está deixando a rede, ou de um nó que foi reconfigurado devido a outro nó ter deixado a rede. Essa nova região de chaves faz fronteira VERTICALMENTE com a sua antiga região de chaves.

Quando a reconfiguração realizada é do tipo 5 ou 6, o nó sendo reconfigurado precisa transferir a sua antiga região de chaves para algum nó pois ficará responsável por outra região completamente diferente. A mensagem de resposta deve retornar o nó para o qual as antigas chaves foram transferidas, pois essa informação é útil na atualização da tabela de apontadores (ver seção 3.7).

A mensagem de resposta indica apenas se a operação foi bem sucedida, através dos códigos de retorno 0 – para sucesso e 1 para falha. O campo de identificação possui o comportamento usual de associar requisições a respostas. O formato dessa mensagem, que utiliza 8 como código de operação, é mostrado na Figura 5.10.

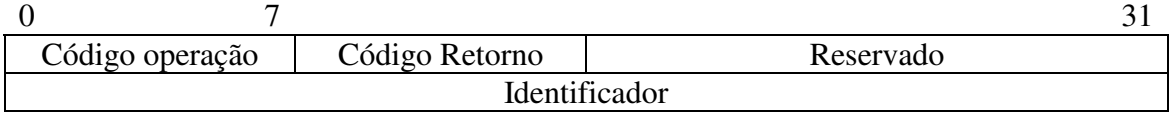


Figura 5-10. Formato da mensagem SgridReconfigurePeerResp.

5.2 Protocolo NATal

O protocolo NATal utiliza como camada de transporte o protocolo UDP e segue o modelo cliente-servidor, onde os nós das redes internas desempenham o papel de clientes e o Roteador (Roteador NATal - NR), desempenha o papel de servidor. A porta de destino a ser utilizada para os pacotes endereçados ao NR deve ser a porta 1975. A porta utilizada pelo cliente segue a forma usual do modelo TCP/IP, onde os clientes utilizam portas aleatórias maiores ou iguais a 1024.

5.2.1 Formato das Mensagens

Nessa seção são descritos os formatos das mensagens e qual a finalidade de cada uma. Todas as mensagens possuem, como um de seus campos, o código da operação, que serve para indicar o tipo de mensagem. A Tabela 5.4 mostra os códigos de todas as mensagens do protocolo NATal, onde pode-se observar que mensagens de resposta possuem códigos diferentes das mensagens de requisição. Os textos na coluna "*Nome Abreviado*" são utilizados ao longo desse texto para simplificar as referências a cada uma das mensagens. Todos os campos das mensagens são transmitidos em "*network byte order*". Além disso, em diversas mensagens existe um campo "*reservado*" que é utilizado nesse texto para tornar o tamanho das mensagens múltiplo de 32 bytes, o que simplifica a visualização do formato das mesmas. Em uma implementação real esses campos não precisam existir.

Tabela 5-4. Códigos de operação do protocolo NATal.

Código	Significado	Nome Abreviado
1	Obter o IP utilizado para o NAT no NR	NatalGetIPNat
2	Resposta da mensagem anterior (1)	NatalGetIPNatResp
3	Obter o nó interno responsável por uma dada chave	NatalGetPeerKey
4	Resposta da mensagem anterior (3)	NatalGetPeerKeyResp
5	Registrar nó no NR	NatalRegPeer
6	Resposta da mensagem anterior (5)	NatalRegPeerResp

7	Cancelar Registro de nó no NR	NatalUnregPeer
8	Resposta da mensagem anterior (7)	NatalUnregPeerResp
9	Obter o IP do LSP	NatalGetLSP
10	Resposta da mensagem anterior (9)	NatalGetLSPResp
11	Definir o LSP no NR	NatalSetLSP
12	Resposta da mensagem anterior (11)	NatalSetLSPResp
13	Alterar a região de chaves no registro de um nó	NatalChangeKeys
14	Resposta da mensagem anterior (13)	NatalChangeKeysResp
15	Definir entradas na tabela de roteamento do NR	NatalSetNGB
16	Resposta da mensagem anterior (15)	NatalSetNGBResp
17	Obter a lista de Peers registrados	NatalGetPeers
18	Resposta da mensagem anterior (17)	NatalGetPeersResp

Embora algumas operações possuam um formato de mensagem de resposta específico existem operações que compartilham o mesmo formato de mensagem de resposta. Essa mensagem comum de resposta está ilustrada na Figura 5.11 e é referenciada ao longo desse texto como *NatalResp*. Além do campo com o código de operação, essa mensagem inclui apenas um outro campo que serve para indicar o *status* da operação, ou seja, se foi ou não bem sucedida. O código de operação será preenchido de acordo com a operação à qual está relacionada.

0 7	15 16	31
Código operação	Código Retorno	Reservado

Figura 5-11. Formato da mensagem de resposta comum (NatalResp).

A Tabela 5.5 mostra os possíveis códigos de retorno relacionados às diversas operações da Tabela 5.4.

Tabela 5-5. Códigos de retorno das operações do protocolo NATal.

Código de Retorno	Significado
0	Sucesso na realização da operação
1	Falha na realização da operação
2	Erro: parâmetros inválidos
3	Nenhum nó registrado no NR para a chave pesquisada
4	Nenhum nó registrado como LSP
5	Nó não registrado no NR
6	Não tem autorização

5.2.1.1 Mensagem para Obtenção do IP do LSP

O NR é um roteador e possui, portanto, pelo menos dois endereços IP. Como o mapeamento do endereço IP do cliente para algum ponto da grade é feito relativo ao IP utilizado pelo NR/LSP na rede P2P global, o cliente precisa conhecer qual dos endereços do NR é utilizado para essa finalidade. Esse endereço normalmente é o endereço da interface externa do NR e pode ser obtido através da mensagem *NatalGetIPNat*, que é mostrada na Figura 5.12. O código de operação utilizado é 1. Essa operação é importante pois embora o cliente conheça "um" IP do NR, como o método de informar esse endereço ao nó pode variar (DHCP, DNS, SLP, ...), não necessariamente o endereço IP passado aos nós da rede através desses métodos é o endereço utilizado pelo NR para realizar o NAT. Um caso típico dessa situação seria informar aos clientes o endereço da interface interna do NR quando o endereço utilizado para o NAT é o endereço da interface externa.

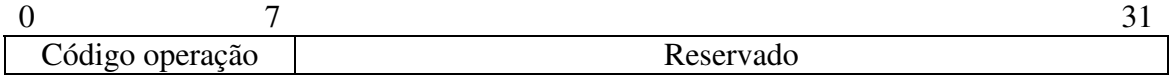


Figura 5-12. Formato da mensagem NatalGetIPNat.

A mensagem de resposta deve conter o referido IP e é mostrada na Figura 5.13. O código de operação é 2 e o campo "Código Retorno" indica se a operação foi bem sucedida. O campo IP contém o endereço solicitado.

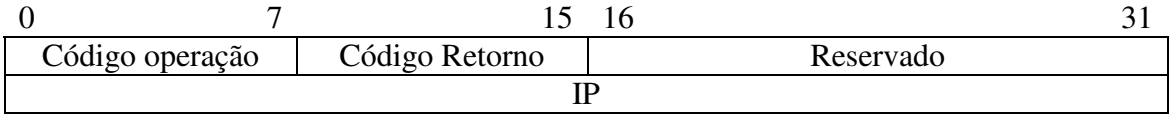


Figura 5-13. Formato da mensagem NatalGetIPNatResp.

5.2.1.2 Mensagem para Obter o Nó Responsável por uma Determinada Chave no NR (*NatalGetPeerKey*)

Antes de se registrar no NR um nó que deseja entrar na rede deve verificar se já existe algum nó da rede interna responsável pela coordenada da grade para a qual ele é mapeado. Em caso positivo o novo nó solicita a esse nó que divida as suas chaves. Em caso negativo o novo nó torna-se o LSP da rede interna, conforme explicado na seção 4.2.6. A mensagem utilizada para solicitar ao NR que informe o nó responsável por uma coordenada da rede interna é *NatalGetPeerKey* que possui o formato mostrado na Figura 5.14. O código de operação é 3 e os campos X e Y indicam a coordenada da grade para a qual se deseja descobrir o nó responsável.

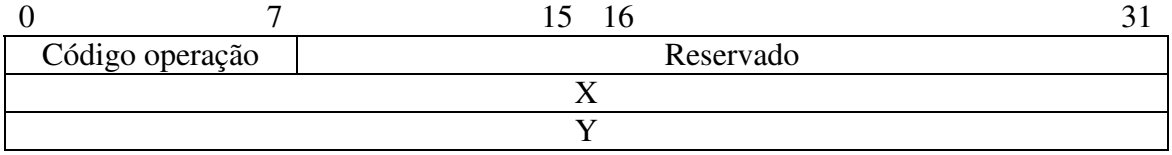


Figura 5-14. Formato da mensagem NatalGetPeerKey.

A mensagem de resposta deve conter o referido IP e é mostrada na Figura 5.15. O código de operação é 4 e o campo "Código Retorno" indica se a operação foi bem sucedida. O campo IP contém o endereço do nó responsável pela chave solicitada ou zero caso nenhum nó esteja registrado como responsável pela mesma. Nesse último casso tal situação também é indicada através do campo *status*.

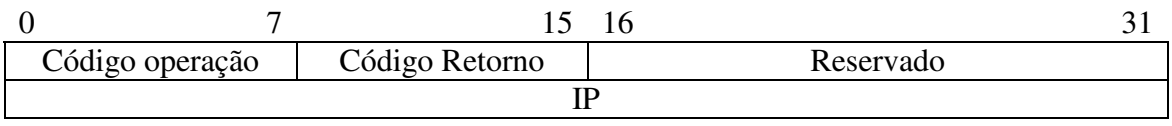


Figura 5-15. Formato da mensagem NatalGetPeerKeyResp.

5.2.1.3 Mensagem de Registro do Nó no NR (NATalRegPeer)

Essa mensagem, que é mostrada na Figura 5.16, é enviada por cada nó da rede interna ao entrar na rede SGrid. Ela registra o nó no NR juntamente com as chaves sob a sua responsabilidade. Informações adicionais, a respeito da capacidade do nó, podem ser enviadas ao NR e são utilizadas para no processo de escolha de um novo LSP.

0 7 31

Código operação	Unidade Memória	Unidade CPU	Unidade Link
Memória		CPU	
Link		Reservado	
X0			
Y0			
X1			
Y1			

Figura 5-16. Formato da mensagem NatalRegPeer.

Os campos x0,y0,x1,y1 determinam a região de chaves sob a responsabilidade do nó, onde (x0,y0) indica a coordenada superior esquerda dessa região e (x1,y1) a coordenada inferior direita. O código de operação utilizado é 5. Os campos *Memória*, *CPU* e *Link* representam a configuração da máquina fazendo o registro e a unidade em que são expressos é indicada nos respectivos campos de unidade. O campo *Link* indica o tempo médio de acesso do nó ao NR e deve ser calculado pelo nó antes de fazer o registro. Caso o nó não deseje informar o valor de algum desses três parâmetros basta preencher o referido campo com zero. Não existe um campo com o endereço do nó a ser registrado, pois o endereço utilizado será obtido através do IP de origem do pacote.

Não existe um formato de mensagem de resposta específico para o registro de um nó de modo que a mensagem de resposta comum (*NatalResp*) é utilizada com o código de operação 6. Os códigos de retorno a serem informados são: 0 - quando o registro for bem sucedido, 2- quando houver erro nos parâmetros, por exemplo, um código de unidade inválido ou coordenadas que já estão sob a responsabilidade de algum outro nó. Além desses, o código de erro genérico 1 pode ser utilizado quando algum outro tipo de problema ocorrer.

Os campos referentes às unidades são valores numéricos de acordo com a tabela 5.6.

Tabela 5-6. Códigos de unidades para recursos dos nós.

Tipo	Código	Descrição
Memória	0	Bytes
	1	Kbytes
	2	MBytes
	3	GBytes
CPU	0	Mhz
	1	GHz
Link	1	Ms (milisegundos)

5.2.1.4 Mensagem de Cancelamento do Registro de um Nó no NR (*NatalUnregPeer*)

Essa mensagem é enviada por um nó ao deixar a rede interna e apaga todas as informações relacionadas a esse nó do NR. Seu formato é muito simples, consistindo apenas do código da operação a ser realizada, que no caso é o código 7. Da mesma maneira que no registro de um nó, o endereço do nó a ter o registro cancelado é obtido através do IP de origem do pacote. A Figura 5.17 ilustra o formato dessa mensagem.

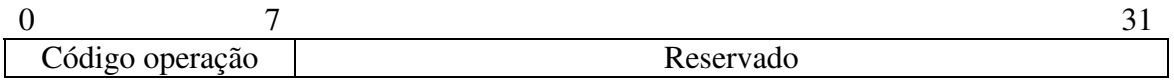


Figura 5-17. Formato da mensagem NatalUnregPeer.

Não existe um formato de mensagem de resposta específico para o cancelamento do registro de um nó, de modo que a mensagem de resposta comum (*NatalResp*) é utilizada, com código de operação 8. Os códigos de retorno a serem informados são: 0 - quando o cancelamento do registro for bem sucedido e 5 - quando o nó solicitando o cancelamento não estiver registrado no NR. Além disso, o código de erro genérico 1 pode ser utilizado quando algum outro tipo de problema ocorrer.

5.2.1.5 Mensagem para Obter o LSP

É importante que os nós da rede interna monitorem periodicamente o LSP para verificar se o mesmo está operando normalmente. A mensagem utilizada para um nó

descobrir quem é o LSP da rede interna é *NatalGetLSP* e está ilustrada na Figura 5.18. O código de operação a ser utilizado é 9.

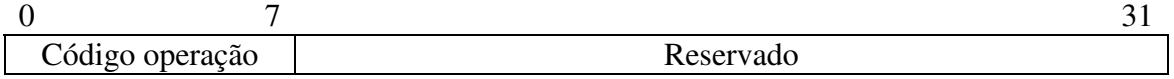


Figura 5-18. Formato da mensagem NatalGetLSP.

A mensagem de resposta deve conter o referido IP e é mostrada na Figura 5.19. O código de operação é 10 e o campo "Código Retorno" indica se a operação foi bem sucedida. O campo IP contém o endereço do LSP da rede interna.

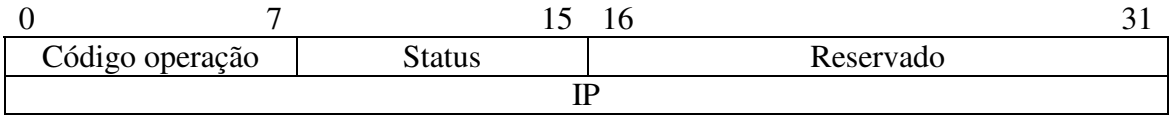


Figura 5-19. Formato da mensagem NatalGetLSPResp.

5.2.1.6 Mensagem para Definir o LSP

Caso algum nó da rede interna detecte que o LSP não está operacional, esse nó pode solicitar ao NR que o promova a LSP. A mensagem *NatalSetLSP* é utilizada com essa finalidade e está ilustrada na Figura 5.20. O código de operação a ser utilizado é 11. Não há necessidade de nenhum campo informnado o endereço IP do nó, pois essa informação é obtida através do endereço de origem do pacote. Antes de promover o nó como o novo LSP, o NR pode verificar se o LSP atual realmente apresenta problemas enviando uma requisição de pesquisa para o mesmo.

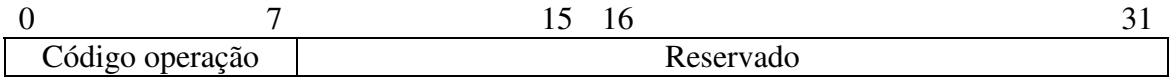


Figura 5-20. Formato da mensagem NatalSetLSP.

Não existe um formato de mensagem de resposta específico essa operação, de modo que a mensagem de resposta comum (*NatalResp*) é utilizada, com código de

operação 12. Os códigos de retorno a serem informados são: 0 – significando que o peer foi promovido a LSP e 5 - quando for detectado que o antigo LSP ainda está operacional. Além disso, o código de erro genérico 1 pode ser utilizado quando algum outro tipo de problema ocorrer.

5.2.1.7 Mensagem para Alterar a Região de Chaves de um Nó no NR (*NATalChangeKeys)*

Quando um nó deixa a rede e transfere suas chaves para outro nó, o nó que recebe as chaves deve informar ao NR suas nova região de chaves. A mensagem utilizada para essa finalidade é *NatalChangeKey* e possui o formato ilustrado na Figura 5.21. O código de operação a ser utilizado é 13 e os valores x0, y0, x1,y1 representam a nova região de chaves do nó, onde (x0,y0) indica a coordenada superior esquerda e (x1,y1) a coordenada inferior direita da mesma. Nessa mensagem, não existe um campo com o endereço do nó a ter a região de chaves alteradas pois essa informação é obtida através do IP de origem do pacote.

0 7	31
Código operação	Reservado
X0	
Y0	
X1	
Y1	

Figura 5-21. Formato da mensagem NatalChangeKeys.

Não existe um formato de mensagem de resposta específico para essa operação, de modo que a mensagem de resposta comum (*NatalResp*) é utilizada com o código de operação 14. Os códigos de retorno a serem informados são: 0 - quando a alteração for bem sucedido, 2 - quando a região informada, ou parte dela, ainda estiver sob a responsabilidade de outro nó. Além desses dois valores, o código de erro genérico 1 pode ser utilizado quando algum outro tipo de problema ocorrer.

5.2.1.8 Mensagem para Definir Entradas na Tabela de Roteamento Global do NR (*NATalSetNGB*)

O LSP precisa propagar sua tabela de roteamento da rede P2P global para o NR. Essa tabela contém os vizinhos horizontal e vertical do LSP em cada nível da grade. Sempre que essa tabela for alterada no LSP a modificação deve ser informada ao NR. A mensagem para realizar essa tarefa é *NatalSetNGB* com o código de operação 15, e está ilustrada na Figura 5.22. O campo tipo contém o valor ´H´ ou ´V´ indicando se a entrada refere-se a um vizinho horizontal ou vertical. O campo nível indica o nível ao qual a entrada se refere e o campo IP informa o endereço IP do vizinho. Finalmente o campo "Num. Entradas" indica quantas entradas, ou seja, quantas triplas (Tipo – Nível – IP) estão contidas na mensagem. Para cancelar uma entrada na tabela basta preencher o campo do endereço IP com zeros. Como essa mensagem só pode ser enviada pelo LSP da rede, o NR faz uma verificação do endereço de origem do pacote antes de realizar a operação.

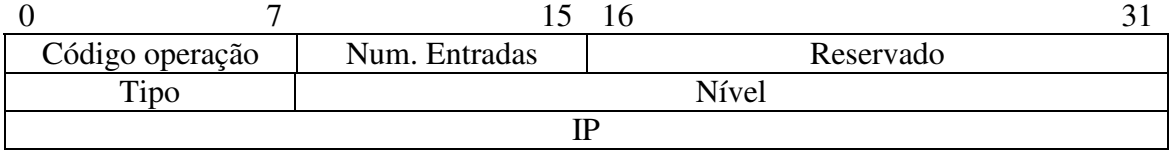


Figura 5-22. Formato da mensagem NatalSetNGB.

Não existe um formato de mensagem de resposta específico para essa operação, de modo que a mensagem de resposta comum (*NatalResp*) é utilizada, com código de operação 16. Os códigos de retorno a serem informados são: 0 – significando que a tabela foi alterada com sucesso, 2 – significando erro em algum parâmetro, por exemplo, o campo tipo com valor diferente de ´H´ e ´V´, e 6 - quando o nó requisitando a operação não for o LSP. Além disso, o código de erro genérico 1 pode ser utilizado quando algum outro tipo de problema ocorrer. Quando a mensagem de requisição possuir mais de uma entrada e alguma delas apresentar problemas nenhuma alteração é feita na tabela. Ou seja, essa é uma operação atômica: ou todas as alterações são feitas, ou nenhuma é executada.

5.2.1.9 Mensagem para Obter a Lista dos Nós Internos (*NATalGetPeers*)

O LSP mantém uma lista com os nós registrados (e suas características) para determinar o nó a se tornar LSP quando ele deixar a rede. Essa lista pode ser obtida através da mensagem *NataGetPeers* que é mostrada na Figura 5.23. O código de operação a ser utilizado é 17. Como essa mensagem só pode ser enviada pelo LSP da rede, o NR faz uma verificação do endereço de origem do pacote antes de realizar a operação. O campo "Num. Entradas" indica quantos nós devem ser retornados e caso seja definido com zero, todos os nós registrados no NR são retornados. Como essa mensagem só pode ser enviada pelo LSP da rede, o NR faz uma verificação do endereço de origem do pacote antes de realizar a operação.

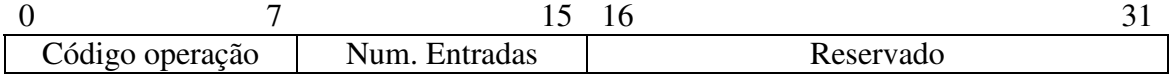


Figura 5-23. Formato da mensagem NatalGetPeers.

A Figura 5.24 ilustra a mensagem de resposta com a listagem dos nós. O código a ser utilizado nessa mensagem é 18. Os códigos de retorno a serem informados são: 0 - quando a listagem for enviada com sucesso, 6 - quando o nó requisitando a operação não for o LSP. Além disso, o código de erro genérico 1 pode ser utilizado quando algum outro tipo de problema ocorrer. Como a relação de nós pode não caber em um único pacote UDP, alguns campos são utilizados para que o receptor determine o número total de pacotes a serem recebidos. O campo "*Num Peers Tota*l" indica quantas triplas serão retornadas (possivelmente em várias mensagens). O campo "*Num entradas*" indicam quantas triplas estão sendo enviadas nesse pacote UDP. Se esses valores forem iguais significa que a resposta coube em um único pacote. Se forem diferentes o campo "Num. Seqüência" é utilizado para numerar os pacotes e permitir ao receptor detectar caso algum deles seja perdido. O receptor da mensagem determinará se ainda existem outros pacotes a serem recebidos acumulando o número de triplas recebidas e comparando esse valor com o número total indicado no campo "*Num Peers Total*". Os campos referentes às unidades da memória, CPU e Link (e os campos com os valores associados) possuem o mesmo significado da mensagem de registro de nós, ou seja, informam a capacidade

do nó, cujo endereço é informado no campo IP. Os campos referentes às unidades são valores numéricos de acordo com a Tabela 5.6.

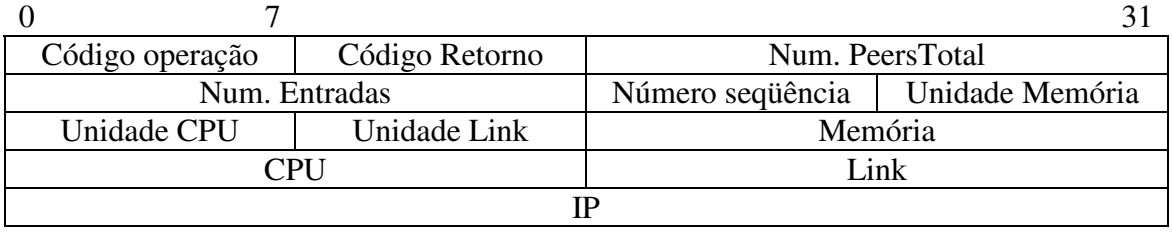


Figura 5-24. Formato da mensagem NatalGetPeersResp.

6 Implementação

A implementação relacionada ao modelo proposto nesse texto pode ser dividida em três partes: (1) Simulador do SGrid, (2) Módulo SGrid/NATal para o Kernel do Linux, e (3) aplicação P2P.

A criação do simulador do Sgrid visa permitir a análise dessa arquitetura em uma rede composta por um número muito elevado de nós, visto que isso seria bastante difícil de se realizar utilizando máquinas reais. As duas outras partes permitem a utilização da arquitetura em um ambiente real, onde o módulo desenvolvido para o kernel do Linux, implementando os protocolos SGrid e NATal, permite que máquinas executando esse sistema operacional sejam utilizadas como roteadores NATal. A parte da aplicação P2P refere-se ao software necessário na máquina do usuário para que o mesmo possa utilizar a rede P2P.

6.1 Simulador SGrid

O simulador possui diversas funcionalidades implementadas que permitem a criação de uma rede P2P e a execução de diversas operações que comprovam que a rede comporta-se de acordo com o modelo proposto. Os resultados apresentados consideram que cada rede possui apenas um nó, e evidentemente esse nó é o LSP. Essa abordagem foi utilizada, pois uma rede P2P formada apenas por LSPs produz o pior caso para as operações analisadas, uma vez que nós internos não executam roteamento e não causam reconfigurações na rede.

A interface gráfica da aplicação, mostrada na figura 6.1, possui basicamente duas partes: uma área onde é desenhada uma grade representando a rede e seus nós, e outra contendo botões e caixas de texto, que controla as operações a serem executadas.

A área de desenho da rede utiliza recursos de cores para facilitar a compreensão das operações executadas. Para operações de inserção de um novo nó, por exemplo, são utilizadas quatro cores diferentes para identificar os seguintes tipos de nós envolvidos no processo: primeiro nó contatado na rede, local onde o novo nó foi inserido, nós visitados durante a operação e nós que tiveram seus apontadores alterados.

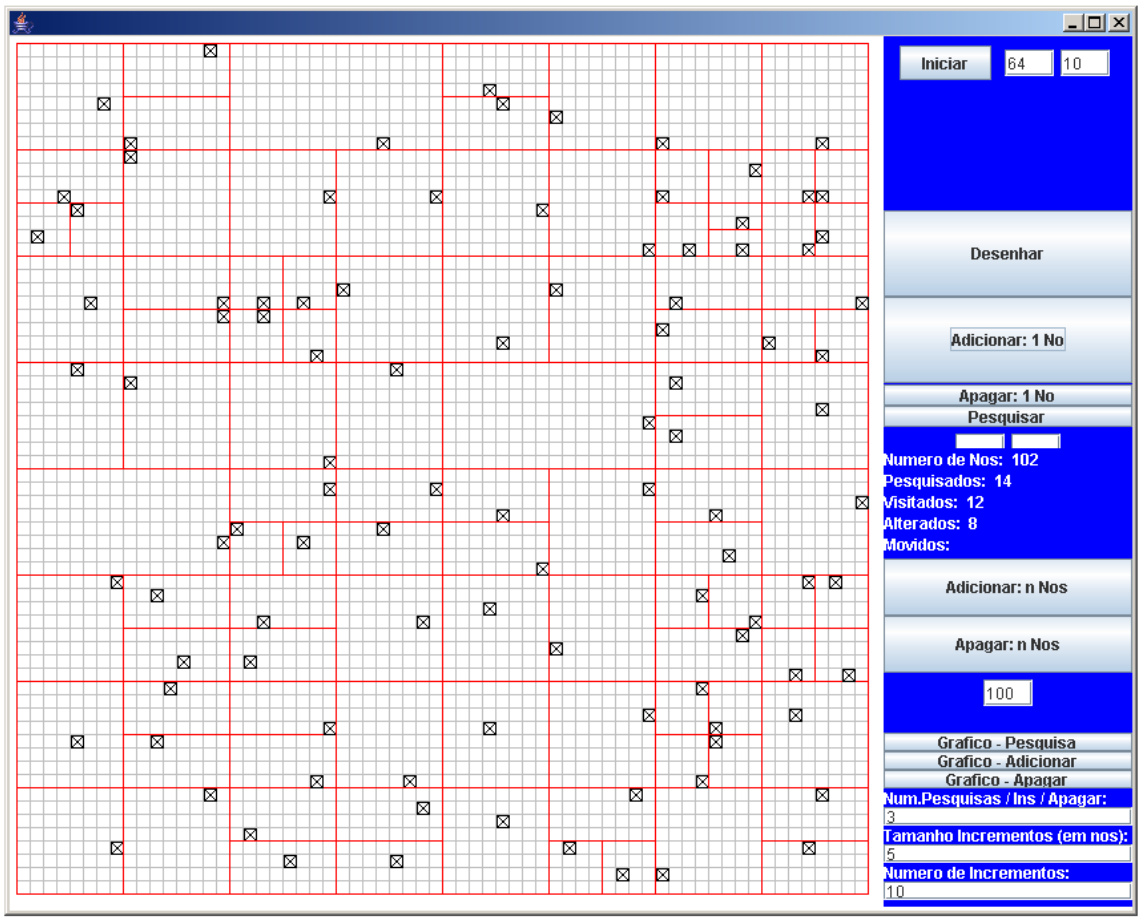


Figura 6-1. Tela do simulador SGrid.

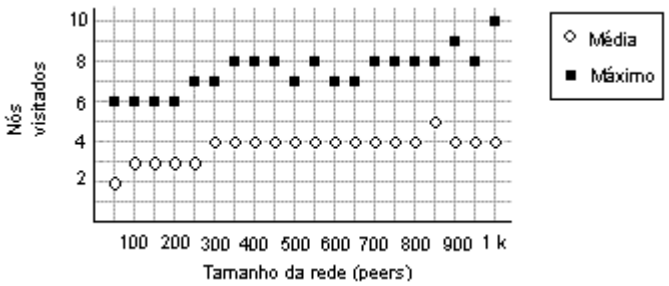
Além das operações de adição e remoção de nós, que podem ser feitas individualmente ou em grupo, e da operação de pesquisa de chaves, o sistema permite a geração de gráficos estatísticos para analisar o comportamento de algumas variáveis importantes para o desempenho e escalabilidade da rede. Os dois principais estão ilustrados na Figura 6.2: (1) o primeiro gráfico mostra o número de nós visitados (média utilizando linha pontilhada e máximo utilizando linha cheia) em uma operação de pesquisa variando o número de nós na rede; (2) o segundo gráfico ilustra os nós reconfigurados (média utilizando linha pontilhada e máximo utilizando linha cheia) quando um novo nó entra na rede. Nesse segundo gráfico também houve variação do número de nós na rede.

Para o experimento cujo gráfico de pesquisa está ilustrado na Figura 6.1, as consultas foram realizadas a cada intervalo de 50 novos nós inseridos na rede. O número de consultas foi 300 (trezentos) e as chaves pesquisadas foram geradas aleatoriamente. Para o segundo gráfico, a avaliação de desempenho relacionada à inserção de novos nós considerou redes com variações de 100 (cem) nós. O número de

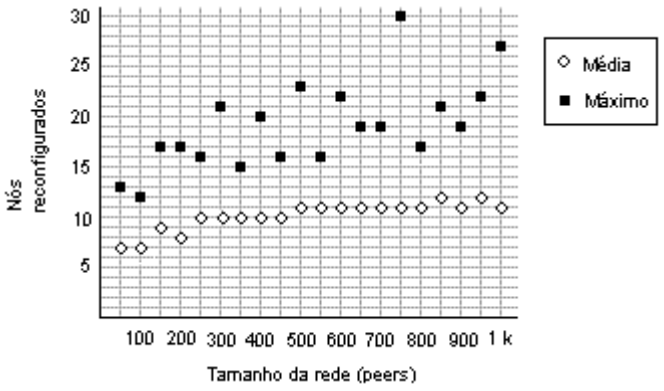
nós inseridos para avaliar a operação de reconfiguração para cada rede foi 50 (cinqüenta).

As redes utilizadas nos experimentos foram criadas gerando-se os nós em posições aleatórias. Além disso, todas as operações de busca e inserção foram realizadas a partir de um mesmo nó base, que foi o primeiro nó a entrar na rede. As configurações de *hardware* e *software* utilizados no experimento foram: Pentium IV 2.4 GHz, 512 MB memória RAM, Windows XP, Java J2SE5.0 e Eclipse 3.1. Entretanto todos os valores medidos pelo experimento, como, por exemplo, o número de nós visitados em uma busca, o número de nós reconfigurados após mudanças de topologia, o posicionamento dos nós na grade, não são afetados pelo ambiente operacional.

Os resultados mostraram que o número máximo de nós visitados em um processo de pesquisa é O(Log N). Contudo, na média, este valor é reduzido pela metade. Em uma rede com 1000 (mil) nós o número máximo de nós visitados em uma pesquisa foi 10 (dez), que corresponde a O(Log N). Na média o número de nós visitados foi 4 (quatro), que é menor que O(Log N)/2. O número de nós reconfigurados depois de inserir um novo nó é inferior a O(Log2 N).



a) Pesquisa por chaves



b) Inserção de nós

Figura 6-2. Gráficos gerados pela ferramenta de simulação do SGrid.

O sistema foi desenvolvido em Java e o principal elemento de sua arquitetura de implementação é mostrado na figura 6.3.

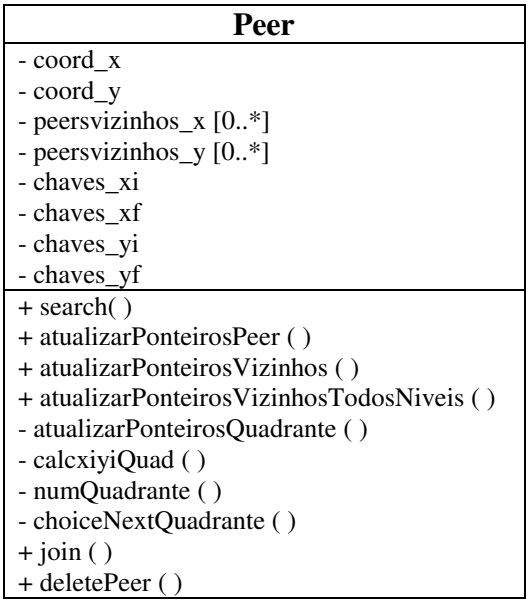


Figura 6-3. Classe referente à implementação de um *Peer* no SGrid.

Os atributos *coord_x* e coord_y armazenam as coordenadas do espaço onde o nó encontra-se e os atributos *chaves_** delimitam o conjunto de chaves (região do espaço) pela qual o nó está encarregado. Os atributos *peersvizinhos_** armazenam os apontadores para os nós nos quadrantes vizinhos em cada nível.

O método *search* é utilizado para encontrar o nó responsável por uma determinada chave. Os apontadores para os nós nos quadrantes vizinhos de cada nível são obtidos através do método *atualizarPonteirosPeer*. Sempre que um nó entra ou deixa a rede (utilizando os métodos *join* e *deletePeer*, respectivamente), alguns nós precisam ter seus apontadores reconfigurados, o que é feito através de chamadas aos métodos *atualizarPonteirosVizinhos**. Algumas operações são executadas constantemente por diversos métodos, entre as quais podemos citar: (1) calcular as coordenadas iniciais do quadrado de um determinado nível que contenha um dado ponto do espaço, o que é feito através do método *calcxiyiQuad*; (2) calcular o quadrante ao qual um ponto pertence, o que é feito através do método *numQuadrante*; (3) determinar, em uma operação de busca, o quadrante onde deve ser escolhido o próximo nó a ser visitado, o que é feito através do método *choiceNextQuadrante*.

Embora existam diversas outras classes na implementação do SGrid, na figura 6.3 foi mostrado apenas o esquema da classe *Peer* devido ao fato de que essa é a classe principal do sistema, pois concentra as operações mais importantes da rede e implementa as funcionalidades necessárias para a utilização real do sistema. A maior parte das outras classes se referem à construção do ambiente para interação do usuário e a geração dos dados estatísticos a respeito de determinadas operações.

6.2 Módulo NATal para o Kernel do Linux

Atualmente é uma prática comum a utilização de máquinas Linux atuando como roteadores devido a alta confiabilidade, desempenho e recursos disponíveis nesses sistemas. Esses recursos consistem em mecanismos auxiliares à tarefa de roteamento, como, por exemplo, controle de banda, análise de tráfego e principalmente filtragem de pacotes. Dessa forma, é natural que qualquer novo mecanismo relacionado a roteamento, como é o caso da proposta apresentada nesse trabalho, forneça uma implementação para esse sistema operacional.

Nessa seção é apresentado inicialmente o suporte oferecido pelo Linux para o desenvolvimento de módulos do *kernel* que interagem com o código de roteamento do sistema. Posteriormente é mostrado como esse suporte foi utilizado para a implementação dos protocolos NATal e SGrid.

6.2.1 A Arquitetura do Netfilter

A arquitetura de roteamento do Linux, chamada Netfilter [63], é bastante flexível e permite o desenvolvimento de módulos para o kernel do sistema operacional que podem manipular completamente os pacotes recebidos pela máquina e assumir, se desejado, parcial ou totalmente a tarefa de roteamento do mesmo. Para isso, o código do módulo consiste de uma ou mais funções que são registradas no kernel e recebem os pacotes gerados ou recebidos pela máquina. Para simplificar esse processo o Netfilter define cinco pontos, chamados *hooks*, ao longo do caminho percorrido por um pacote dentro dessa arquitetura de roteamento. Desse modo, o usuário pode registrar funções em um ou vários desses *hooks*. A Figura 6.4 mostra os cinco *hooks* existentes, e seus nomes simbólicos: NF_IP_PRE_ROUTING, NF_IP_LOCAL_IN, NF_IP_FORWARD, NF_IP_POST_ROUTING e NF_IP_LOCAL_OUT.

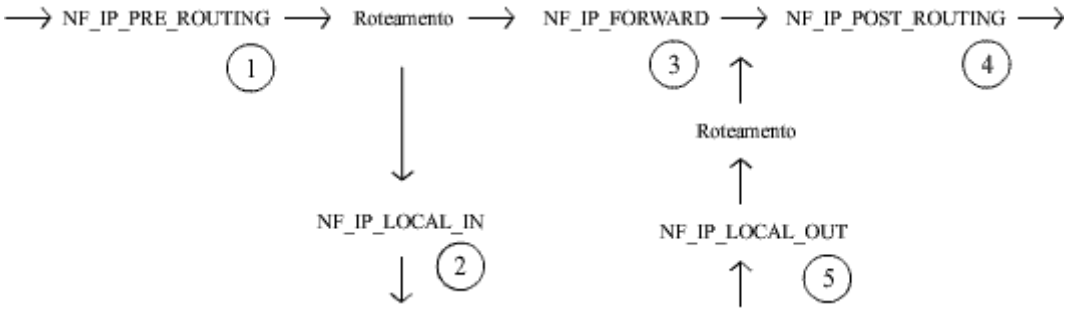


Figura 6-4. Arquitetura do Netfilter do Linux com os seus cinco *hooks*.

As setas na Figura 6.4 indicam a direção do fluxo dos pacotes, de modo que, por exemplo, pacotes destinados à máquina passariam pelos *hooks* 1 e 2 e conseqüentemente apenas as funções registradas para os mesmos seriam chamadas. Da mesma forma, pacotes sendo roteados, passariam pelos *hooks* 1, 3 e 4, e conseqüentemente apenas as funções registradas para os mesmos seriam chamadas. A Tabela 6.1 descreve o significado de cada *hook*.

Tabela 6-1. Os cinco *hooks* do Netfilter.

Nome simbólico do *Hook*	Descrição
NF_IP_PRE_ROUTING	Logo ao receber o pacote e antes das decisões de roteamento.
NF_IP_LOCAL_IN	Logo após as decisões de roteamento, mas apenas se o pacote for destinado a esta máquina.
NF_IP_FORWARD	Logo após as decisões de roteamento, mas apenas se o pacote é destinado a uma outra interface diferente da que foi recebido.
NF_IP_POST_ROUTING	Para todos os pacotes deixando a máquina e depois das decisões de roteamento.
NF_IP_LOCAL_OUT	Para pacotes gerados por algum processo local da máquina, mas antes das decisões de roteamento.

A função registrada em um *hook* recebe o pacote em um buffer e pode manipulá-lo como desejar, inclusive fazendo alterações, tanto nos cabeçalhos quanto na parte de dados do mesmo. Evidentemente, dependendo da alteração realizada campos de *checksum*, como o do cabeçalho IP, podem precisar ser recalculados e isso deve ser feito pela própria função. Ao terminar suas operações uma função *hook* deve indicar ao Netfilter como o pacote deve ser tratado desse ponto em diante, podendo solicitar, por exemplo, que o pacote seja descartado, continue seu caminho normalmente, ou inclusive assumir a responsabilidade total sobre o roteamento do pacote. A Tabela 6.2 mostra os possíveis códigos de retorno das funções *hook* e conseqüentemente a ação que ela representa.

Tabela 6-2. Códigos de retorno das funções *hook*.

Código de retorno	Significado
NF_ACCEPT	O pacote deve continuar seu caminho dentro do Netfilter normalmente, ou seja, deve ser analisado pelo próximo *hook*.
NF_DROP	O pacote deve ser descartado.
NF_STOLEN	O Netfilter deve ignorar o pacote, pois a própria função *hook* assumirá a responsabilidade sobre o mesmo. Esse pacote não será analisado por mais nenhum *hook*. É utilizado quando a própria função deseja realizar o roteamento e o encaminhamento do pacote.
NF_QUEUE	O pacote é enviado para uma área de memória onde pode ser capturado por programas no espaço do usuário.
NF_REPEAT	Chama a função *hook* novamente.

O protótipo das funções a serem registradas é mostrado na Figura 6.5, onde *nf_hookfn* consiste no nome da função sendo registrada para receber os pacotes e *hooknum* é o nome simbólico do *hook* (ver tabela 6.1) para o qual o registro é feito.

```
typedef unsigned int nf_hookfn(unsigned int hooknum,
                                        struct sk_buff **skb,
                                        const struct net_device *in,
                                        const struct net_device *out,
                                        int (*okfn)(struct sk_buff
*));
```

Figura 6-5. Protótipo de uma função *hook* do Netfilter.

Os parâmetros *in* e *out* representam os dispositivos por onde o pacote foi recebido e por onde será enviado, respectivamente. Evidentemente, nem sempre os dois estão definidos. Para pacotes sendo gerados na própria máquina, por exemplo, apenas *out* é definido. O parâmetro *skb* representa o *buffer* para onde o pacote é copiado, de modo a ser manipulado pela função. Essa estrutura é bastante complexa e contém campos relativos aos diversos tipos de protocolos suportados, desde a camada de enlace até a de transporte. Apenas os campos relacionados com o tipo de pacote recebido (ou enviado) são definidos. Para um pacote TCP recebido em uma rede Ethernet, por exemplo, são definidos os campos do frame Ethernet, os campos do cabeçalho IP e os campos do cabeçalho TCP. Além dos cabeçalhos existe um ponteiro para a parte de dados do pacote. Qualquer alteração nos campos dessa estrutura representa uma alteração no pacote a ser enviado ou recebido. Para realizar NAT, por exemplo, basta alterar o campo referente ao endereço IP (e recalcular o *checksum*). O último parâmeto, *okfn*, é um ponteiro para uma função, mas raramente é utilizado.

O valor de retorno é um dos valores indicados na Tabela 6.2. Portanto, se a função desejar que o Netfilter continue com o processamento do pacote deve retornar NF_ACCEPT, entretanto, se ela mesma desejar assumir o papel de encaminhar o pacote, ela pode chamar as funções necessárias para realizar essa tarefa e depois retornar NF_STOLEN.

6.2.2 O Módulo SGrid/NATal

O módulo SGrid/NATal desenvolvido para o kernel do Linux registra duas funções no Netfilter, uma chamada *hook_func_pre_routing*, que é registrada no *hook* NF_IP_PRE_ROUTING, e outra chamada *hook_func_post_routing*, que é registrada no *hook* NF_IP_POST_ROUTING. Enquanto a função *hook_func_pre_routing* trata tanto de mensagens do protocolo NATal quando de mensagens do protocolo SGrid, a função *hook_func_post_routing* manipula apenas mensagens deste último protocolo.

Como procedimentos diferentes precisam ser executados caso os pacotes tenham sido gerados na rede interna ou na Internet, e mensagens NATal são aceitas apenas se forem recebidas pela interface interna, o módulo precisa que lhe seja informado qual é a interface interna e qual a externa. Feito isso, o módulo pode facilmente verificar a qual

delas cada pacote está relacionado através dos parâmetros *in* e *out* que são recebidos pelas funções *hook*.

A tarefa principal das funções *hook_func_pre_routing* e *hook_func_post_routing* é identificar o tipo de pacote recebido e chamar a função quer realiza os procedimentos referentes ao mesmo. Identificar o tipo de pacote se refere a determinar o protocolo de aplicação (SGrid ou NATal), o tipo de mensagem (consultas/respostas para o protocolo SGrid, registros de nós para o protocolo NATal, etc.) e o sentido da mensagem (rede interna para Internet, Internet para rede interna, etc).

A Figura 6.6 ilustra o algoritmo da função *hook_func_pre_routing*, e mostra que, após algumas verificações iniciais para identificar a interface, o protocolo e a porta, outra função é chamada para realizar o processamento efetivo do pacote. Quando o pacote é do tipo SGrid, duas outras funções podem ser chamadas, que são: *nat_do_from_ext* e *nat_do_from_int*. A principal tarefa de cada uma delas é calcular o novo endereço de destino do pacote de acordo com as chaves contidas no mesmo. Essa tarefa é realizada analisando a tabela de roteamento P2P e a tabela de registro dos nós internos e pode implicar na realização de NAT no endereço IP de destino, ou no endereço IP de origem, ou em ambos. Quando os pacotes são referentes ao protocolo NATal a função *hook_func_pre_routing* chamará a função relativa ao código de operação contido no pacote. Se esse código for, por exemplo, NATAL_REG_PEER, que indica uma mensagem de registro de um nó da rede interna, a função *regPeer* será chamada. Após chamar qualquer uma dessas funções auxiliares, tanto as relacionadas ao protocolo SGrid quanto as relacionados ao protocolo NATal, *hook_func_pre_routing* retorna o código NF_STOLEN. Isto significa que essas funções encarregam-se de transmitir o pacote, de modo que o Netfilter não precisa mais realizar nenhuma operação referente ao mesmo.

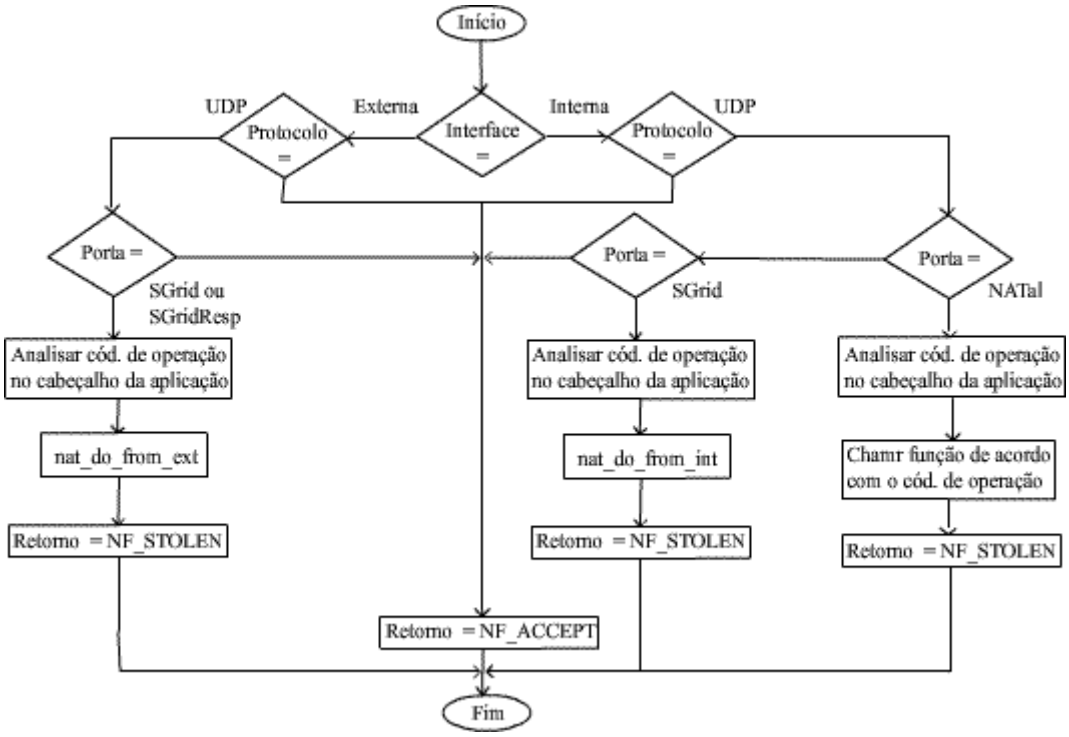


Figura 6-6. Fluxograma da função *hook* registrada no NF_IP_PRE _ROUTING.

A Figura 6.7 ilustra o algoritmo da função *hook_func_post_routing*. A implementação dessa função é bem mais simples que a *hook_func_pre_routing*, uma vez que a sua única tarefa é realizar NAT no endereço de origem dos pacotes destinados à rede externa. Após verificações iniciais de identificação da interface, do protocolo e da porta, a função *snat_do* é chamada para alterar o endereço IP de origem para o endereço utilizado nas operações de NAT, que é um IP do próprio roteador. Posteriormente, o código NF_ACCEPT é retornado, indicando que o Netfilter deve continuar o processamento do pacote.

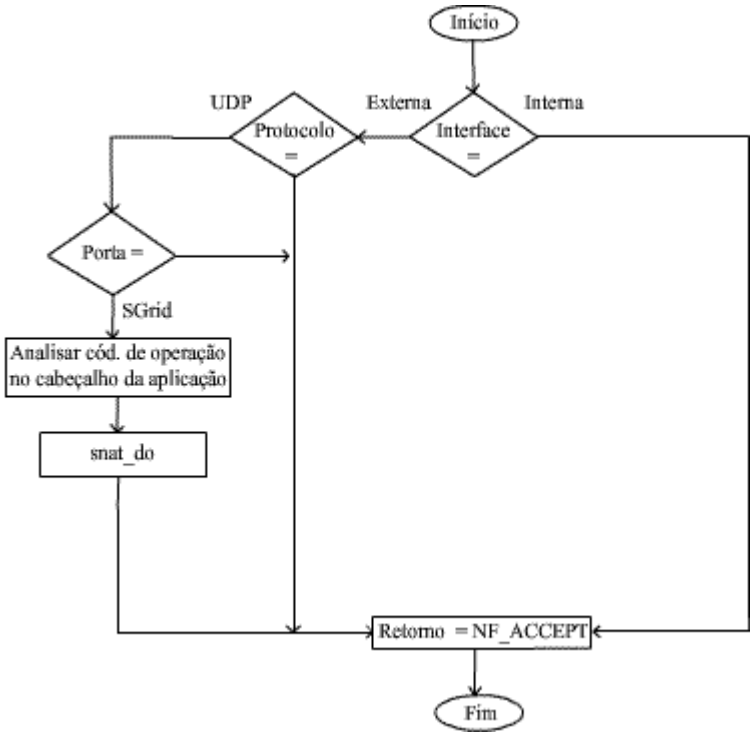


Figura 6-7. Fluxograma da função *hook* registrada no NF_IP_POST _ROUTING.

6.3 Aplicação P2P

Para que o módulo NATal desenvolvido para o kernel do Linux pudesse ser testado foram desenvolvidas duas pequenas aplicações, utilizando a linguagem C [54], que permitem a interação com o módulo e a realização de pesquisas por chaves. A aplicação para interação com o módulo permite a realização, ainda que manualmente, de todas as operações referentes ao protocolo NATal, como, por exemplo, registro dos nós e suas regiões de chaves, definição do LSP e a criação e manutenção da tabela de roteamento P2P. A outra aplicação implementa tanto um cliente quanto um servidor, de modo que é possível ao nó enviar e responder pesquisas por chaves.

7 Trabalhos Relacionados

Nesse capítulo faremos uma análise comparativa entre as arquiteturas descritas no capítulo 2 e a proposta apresentada nesse trabalho. Inicialmente são tratadas as arquiteturas básicas, sem o conceito de *super-peers*, e posteriormente é feita uma comparação do SGrid com o modelo de *super-peers*.

7.1 GnuTella

Conforme explicado na seção 2.4.1, nas redes GnuTella não existe uma topologia definida e cada nó propaga as mensagens para todos os nós vizinhos. Essa flexibilidade em termos da topologia permite que cada nó mantenha conexões com apenas um grupo de nós vizinhos fazendo com que o número de nós que precisam ser reconfigurados quando nós entram ou deixam a rede seja pequeno e inferior ao do SGrid. Além disso, como o conteúdo das mensagens não influencia as operações de roteamento, uma vez que essa tarefa é realizada pelo método de inundação, as pesquisas podem ser feitas usando expressões complexas, ou seja, qualquer linguagem de consulta, desde que seja suportada pelos outros nós. No SGrid, como em todas as outras redes DHT, o dado sendo pesquisado é que determina como o roteamento é realizado, e isso é feito de uma forma que apenas buscas por informações exatas são suportadas diretamente. Portanto, para que SGrid suporte pesquisas por palavras-chave ou faixas de valores, por exemplo, mecanismos adicionais precisam ser desenvolvidos.

Embora essa flexibilidade nas operações de pesquisa e a simplicidade para a manutenção da estrutura da rede sejam características interessantes, as conseqüências decorrentes dessa abordagem são bastante graves e comprometem o desempenho da rede. Como o número de vezes que cada mensagem é propagada é limitado por um campo chamado TTL, não existe garantia de que as informações pesquisadas serão encontradas, mesmo que existam na rede. O SGrid não apenas garante que se a informação pesquisada existe na rede, ela será encontrada, como determina o número máximo de nós visitados durante essa tarefa. Além disso, a estratégia de roteamento por inundação gera um alto tráfego de rede, enquanto que o tráfego gerado no SGrid é bastante reduzido, uma vez que cada nó propaga as mensagens para apenas um de seus vizinhos.

Como todos os nós realizam roteamento e, além disso, essa tarefa é realizada através de inundação o GnuTella é bastante susceptível a ataques, principalmente de usuários enviando um número muito elevado de pesquisas. No SGrid o fato de que cada mensagem é roteada para apenas um nó vizinho, fornece um mecanismo natural de proteção contra esse tipo de ataque. Porém, como todos os nós realizam roteamento no GnuTella existe muito tráfego de roteamento circulando desnecessariamente por dentro das redes das organizações, diferentemente do SGrid que resolve esse problema através dos roteadores NATal.

Para lidar com a heterogeneidade dos nós o GnuTella utiliza um campo nas mensagens de pesquisa que informa os recursos mínimos que um nó deve possuir para que possa responder a pesquisa. Desse modo, mesmo que um nó possua a informação procurada, se não atender aos requisitos de desempenho solicitados ele apenas realiza o roteamento da mensagem e não responde a pesquisa. No SGrid, a arquitetura baseada em LSPs permite reduzir a carga em cada um dos nós além de deixar que o roteamento seja feito apenas pelos roteadores NATal, evitando assim que nós com poucos recursos comprometam o desempenho da rede.

7.2 Chord

As redes Chord caracterizam-se por definir uma arquitetura simples e eficiente, conforme discutido na seção 2.4.2. Entretanto, por utilizar funções *hash* para determinar tanto a localização dos dados quanto a posição dos nós, ou seja, a topologia da rede, não existe nenhuma relação entre as distâncias dos nós na rede P2P e suas distâncias na rede IP. Desse modo, conforme discutido anteriormente, as medidas de desempenho da rede P2P podem não corresponder ao comportamento real da rede. O SGrid, embora utilize uma função *hash* para determinar a localização dos dados, determina a posição dos nós baseado na hierarquia do endereçamento IP, que considera, portanto, as distâncias dos nós na rede física.

Tanto no Chord quanto no SGrid as operações de pesquisa garantem a localização do dado pesquisado, caso esse dado exista na rede, além de determinarem o número máximo de nós que precisam ser visitados para a realização dessa operação. Nas redes Chord o número máximo de nós visitados em uma operação de pesquisa é O(log N), onde N é o número de nós na rede. Embora no SGrid esse número também

seja limitado a O(log N), como apenas os LSPs participam da manutenção da estrutura da rede, N representa a quantidade de LSPs e não o número total de nós. Evidentemente, como existe apenas um LSP por rede o valor de N no SGrid é bastante inferior ao seu valor no Chord e, portanto, as buscas no SGrid são mais eficientes pois visitam um número inferior de nós.

Uma consideração importante decorrente da arquitetura em grade utilizada pelo SGrid, mas que não pode ser observada analisando-se apenas o valor O(log N) é que para pesquisas por chaves próximas ao nó realizando a busca, o SGrid pode apresentar um desempenho ainda melhor em relação a estrutura em anel do Chord. Isso deve- se ao fato de que a tabela de apontadores do Chord é construída considerando apenas um sentido do anel, por exemplo, o sentido horário. Desse modo, cada nó possui apontadores para vizinhos apenas nesse sentido. No outro sentido, o anti-horário, é mantido apenas um apontador para o vizinho imediato. Desse modo, se uma pesquisa é feita por uma chave próxima do nó, porém no sentido anti-horário, ou percorre-se os nós um a um nesse sentido, o que poderia ser ineficiente, ou utiliza-se a tabela de apontadores para os nós vizinhos no sentido horário. Nesse caso, entretanto, todo o anel teria que ser percorrido. Em termos do número de nós visitados isso não vem a ser um problema pois, como cada nó mantém um apontador para o vizinho 2^{n-1}, apenas dois nós precisam ser visitados para se dar uma volta no anel. Porém, como no Chord não existe nenhum conhecimento da localização física dos nós, os nós visitados poderiam estar muito distantes na rede IP. No Sgrid, por outro lado, como cada nó mantém apontadores para vizinhos horizontais e verticais, além de existir uma relação entre a posição dos nós na rede P2P e na rede IP, as pesquisas por chaves próximas são realizadas de forma mais eficiente, pois visitam apenas nós próximos na rede IP.

Em relação ao tamanho da tabela de apontadores, ou seja, ao número de conexões com outros nós que cada nó possui, o SGrid e o Chord mantêm O(log N) entradas nas suas tabelas. Entretanto, consideração semelhante à feita sobre os LSP para as operações de pesquisa se aplica aqui. Desse modo, como apenas os LSP participam da manutenção da estrutura da rede, N possui um valor bastante inferior no SGrid em relação ao Chord. Além disso, apenas os LSPs (e os roteadores NATal) mantêm tabelas de apontadores, uma vez que os demais nós de cada rede enviam suas pesquisas diretamente ao roteador NATal.

Nas redes Chord cada novo nó que entra na rede, ou cada nó que deixa a rede, causa uma mudança de topologia na rede P2P e requer que, no máximo, outros O($\log^2$

N) tenham suas tabelas de apontadores reconfiguradas. No SGrid como as chaves de todos os nós de uma rede particular são vistas na rede P2P como estando sob a responsabilidade do LSP e como o IP do LSP conhecido na Internet é o IP do roteador NATal, enquanto existir pelo menos um nó de uma rede particular pertencendo à rede P2P global, outros nós dessa rede particular podem entrar ou sair da rede P2P sem causar nenhuma reconfiguração. Desse modo, o primeiro nó de uma rede particular a entrar na rede P2P global, ou o último a deixá-la, causam a reconfiguração de no máximo $O(\log^2 N)$ outros nós, onde N é o número de LSPs na rede global.

Como no Chord todos os nós participam do roteamento de mensagens, a rede apresenta os diversos problemas decorrentes dessa abordagem, já citados anteriormente: vulnerabilidade a usuários mal-intencionados realizando o roteamento de forma incorreta ou propagando informações também incorretas sobre as tabelas de roteamento; tráfego desnecessário passando por dentro das redes internas das organizações; dificuldade de tratar com a heterogeneidade dos nós. No SGrid como apenas os roteadores NATal realizam roteamento, esses problemas são resolvidos ou minimizados, conforme foi discutido ao longo desse texto.

A distribuição homogênea das chaves nos nós, de modo que cada nó fique responsável por aproximadamente a mesma quantidade de chaves, é uma característica importante das redes P2P. Embora o SGrid utilize um mecanismo de mapeamento dos nós para pontos da grade que torna impossível dois LSPs mapearem para o mesmo local, a utilização da função *hash* no modelo Chord ainda apresenta uma distribuição mais homogênea das chaves aos nós.

7.3 CAN

As redes CAN utilizam uma arquitetura em grade n-dimensional, porém, diferentemente do SGrid, não seguem um modelo hierárquico. A forma como as chaves são alocadas aos nós é bastante semelhante nas duas arquiteturas, onde o novo nó entrando na rede determina a coordenada da grade para a qual ficará associado e solicita ao nó responsável pela mesma que divida a sua região de chaves em duas partes iguais. Apesar dessa similaridade, no CAN o ponto para o qual cada nó é associado é escolhido aleatoriamente e, portanto, não existe nenhuma relação entre as posições dos nós na rede P2P e suas posições na rede IP. No SGrid, a posição de cada nó na grade é feita através de uma função de mapeamento do seu endereço IP que considera a proximidade

dos nós na rede IP. Embora existam variações da arquitetura que procuram considerar as distâncias IP, os métodos utilizados baseiam-se em medições dos tempos de acesso a algumas máquinas bem conhecidas na Internet, com os servidores de nome raiz do DNS. Essas técnicas não apenas estão sujeitas a variações nesses tempos devido à variação natural nos *links* de comunicação da Internet, como requerem a troca de informações dinâmicas. O mecanismo utilizado no SGrid é estático e, portanto, fornece sempre os mesmos resultados sem necessitar da troca de informações pela rede.

O mecanismo de roteamento utilizado no CAN requer que cada nó mantenha conexões apenas com seus vizinhos imediatos, repassando as mensagens para o que está mais próximo da chave pesquisada. Embora essa abordagem reduza o tamanho da tabela de apontadores para apenas O(número de dimensões) e, consequentemente o número de nós reconfigurados após mudanças de topologia, ela impossibilita que chaves distantes sejam alcançadas com poucos saltos. O modelo hierárquico do SGrid permite que qualquer região do espaço de chaves seja alcançada rapidamente, utilizando para isso os apontadores dos níveis mais altos da hierarquia. Para equiparar os desempenhos das operações de pesquisa seria necessário utilizar um recurso da rede CAN chamado "múltiplas realidades", porém isso gera um efeito colateral de aumentar a quantidade de informações armazenadas em cada nó.

Nas redes CAN todos os nós participam do roteamento de mensagens. Desse modo, a rede apresenta os diversos problemas decorrentes dessa abordagem, já citados anteriormente, quais sejam: vulnerabilidade a usuários mal-intencionados que realizam o roteamento de forma incorreta ou propagam informações também incorretas sobre as tabelas de roteamento; tráfego desnecessário passando por dentro das redes internas das organizações; dificuldade de tratar com a heterogeneidade dos nós. No SGrid como apenas os roteadores NATal realizam roteamento, esses problemas são resolvidos ou minimizados, conforme foi discutido ao longo desse texto.

Por associar os nós a pontos do espaço de forma aleatória o CAN obtém uma melhor distribuição das chaves aos nós que o SGrid.

7.4 Super-Peers

As arquiteturas de *super-peers* existentes atualmente caracterizam-se por serem redes onde apenas os nós que são designados como *super-peers* participam da rede diretamente. Essa participação direta refere-se a manter a tabela de apontadores para nós

vizinhos e realizar o roteamento de mensagens. Os demais nós da rede conectam-se em algum *super-peer* e enviam suas pesquisas e recebem as respostas através deles, não sendo vistos na rede global. Embora existam várias arquiteturas de *super-peers* [39] [64], todas seguem esse modelo, diferenciando-se apenas por questões como o número de nós que cada *super-peer* suporta, a quantidade de *super-peers* existentes, a topologia da rede formada pelos *super-peers*, o mecanismo de pesquisa utilizado, entre outras.

A idéia básica da utilização dos *super-peers* é resolver o problema da heterogeneidade dos nós, uma vez que nós com maior capacidade computacional assumem a responsabilidade sobre os recursos dos nós com menor capacidade. Essa estratégia, contudo, sobrecarrega os *super-peers* pois, além de realizarem as funções relacionadas à manutenção da topologia da rede e do roteamento de mensagens, ainda têm que realizar as pesquisas solicitadas pelos nós comuns ligados a ele e responder as pesquisas endereçadas aos recursos desses nós. Ao introduzir o LSP, o SGrid reduz as tarefas que um *super-peer* precisa realizar para manutenção da topologia da rede, uma vez que o roteamento de mensagens é realizado pelos roteadores NATal e cada nó responde as pesquisas destinadas às chaves sob a sua responsabilidade.

Como nas arquiteturas de *super-peers* os nós comuns não participam da topologia da rede, a entrada e saída desses nós, não requer reconfigurações em outros nós da rede. Entretanto, conectar muitos nós a um único *super-peer* gera uma sobrecarga no mesmo, de modo que aumentar o número de *super-peers* melhora o desempenho da rede, pois reduz a carga individual em cada um. Por outro lado, aumentar o número de *super-peers* implica em reconfigurações na topologia da rede a cada novo *super-peer* que entra ou deixa a rede. Esse fato limita o número de máquinas que participam da rede diretamente, subutilizando, portanto, a capacidade computacional de diversos nós. Como no SGrid apenas a primeira máquina de uma organização a entrar na rede causa uma reconfiguração na rede P2P e, além disso, todos os nós realizam e respondem pesquisas, os recursos computacionais de todas as máquinas são aproveitados.

Diante da possibilidade de qualquer nó tornar-se um *super-peer* independente de sua posição física dentro da rede da organização, e do fato que os *super-peers* realizam roteamento, as arquiteturas de *super-peers* atuais geram tráfego circulando dentro das redes internas das empresas que não é destinado a nenhum de seus nós. Como no SGrid o roteamento é realizado pelos roteadores NATal apenas mensagens destinadas a algum dos nós de uma rede particular são repassados para dentro dessa rede.

7.5 Análise Comparativa

Essa seção apresenta um resumo das características discutidas nas sessões anteriores desse capítulo, sobre as redes Gnutella, Chord, CAN e das arquiteturas com *Super-Peers*, conforme pode ser observado na Tabela 7.1.

Tabela 7-1. Análise comparativa entre arquiteturas P2P.

	Sgrid/NATal	Gnutella	Chord	CAN	Super-Peers
Relação com a rede física	Sim	Não	Não	Não	-
Nós participando do roteamento	Roteadores Natal	Todos	Todos	Todos	Super-peers
Nós mantendo a tabela de roteamento	LSP	Todos	Todos	Todos	Super-peers
Tamanho da tabela de roteamento	O(log N)	-	O(log N), N=número de nós	2d, d=dimensões	-
Nós que causam reconfiguração	Primeiro e último nó de cada rede	Todos	Todos	Todos	Super-peers
Número de nós visitados em uma busca	O(log N), N=número de LSPs	Indeterminado - TTL	O(log N), N=número de nós	$(d/4)(1^{n/d})$, n=número de nós, e d=dimensões	-
Nós que recebem pesquisas.	Todos	Todos	Todos	Todos	Super-peers

Na Tabela 7.1 alguns campos relacionados à arquitetura com *super-peers* não foram preenchidos porque não podem ser aplicados em todos os modelo de *super-peers*. Para as redes Gnutella o campo "tamanho da tabela de roteamento" também não foi preenchido porque esse valor é configurado pela aplicação. Entretanto normalmente são utilizados valores abaixo de 10.

Pela Tabela 7.1 pode-se observar que nas redes Gnutella, Chord e CAN todas as máquinas participam do roteamento, enquanto nas redes Sgrid/NATal apenas os roteadores desempenham essa tarefa. Além disso, embora nas redes com *super-peers* apenas esses nós realizem roteamento a tabela mostra que a entrada ou saída de um *super-peer* causa reconfigurações na rede. Enquanto que nas redes SGrid/NATal apenas a entrada do primeiro nó de cada rede, ou a saída do último, causa reconfigurações na rede P2P global. Nas redes Gnutella, Chord e CAN qualquer mudança de topologia causa reconfigurações na rede P2P global. Ainda pela Tabela 7.1 pode-se observar que ao contrário das redes com *super-peers* onde apenas esses nós recebem as pesquisas a arquitetura Sgrid/NATal se comporta da mesma forma que as redes Gnutella, Chord e CAN, ou seja, todos os nós participam diretamente da rede recebendo pesquisas.

8 Conclusões e Trabalhos Futuros

Nesse trabalho identificamos alguns problemas existentes nas redes P2P atuais e propomos uma nova arquitetura de rede para solucioná-los. Entre os principais problemas identificados podemos citar: sobrecarga nos nós devido à realização do roteamento de mensagens, número elevado de nós reconfigurados devido a mudanças de topologia da rede e a existência de tráfego de roteamento dentro das redes das organizações que não é destinado a nenhuma de suas máquinas. Além disso, muitas redes não consideram as distâncias existentes entre os nós nas redes IP, de modo que nós próximos na rede P2P podem ser distantes na rede física, e as que o fazem, utilizam mecanismos que requerem a troca de informações dinamicamente entre os nós para determinar essas distâncias.

Para resolver os problemas citados a arquitetura proposta é baseada em três características principais. A primeira característica consiste em utilizar um esquema de mapeamento dos endereços IP para pontos da grade de modo que nós próximos na rede IP são próximos na rede P2P, sem precisar de troca de informações dinamicamente. A segunda característica consiste em distribuir as chaves entre os nós de modo que nós de uma mesma organização são responsáveis por regiões adjacentes do espaço de chaves. A terceira característica consiste na transferência das funções de roteamento de mensagens para um roteador. Essas três características juntas permitem que todas as máquinas de uma organização sejam vistas como uma única máquina na rede P2P, e esse fato é o principal responsável pelas melhoras obtidas pela arquitetura, como por exemplo, fazer com que a entrada ou saída de máquinas de uma organização na rede não causem modificações na rede P2P global.

Foi definida uma arquitetura P2P básica, chamada SGrid, que possui as duas primeiras características citadas no parágrafo anterior, ou seja, manter relação dos nós na rede P2P com suas posições na rede IP e atribuir regiões de chaves adjacentes para nós de uma mesma organização. A principal vantagem do esquema de mapeamento dos nós definido nesse trabalho, além de não necessitar da troca de informações entre os nós, é ser um método determinístico, ou seja, cada endereço IP mapeia sempre para o mesmo ponto da grade. Essa característica é essencial para que se possa agrupar as regiões de chaves dos nós de uma organização como uma única região. Essa arquitetura básica foi estendida para suportar a terceira característica citada no parágrafo anterior,

através da criação do protocolo NATal, de modo que a tarefa de roteamento de mensagens é transferida para roteadores com suporte a esse protocolo. Finalmente, para atingir o objetivo de fazer com que todas as máquinas de uma rede particular sejam vistas na rede P2P como uma única máquina foi criado um tipo especial de nó, chamado LSP, que é o único nó de cada rede particular a participar da rede P2P global e possui, como única função na rede P2P, a manutenção da tabela de roteamento P2P. Entretanto, o LSP não realiza roteamento, ele apenas propaga sua tabela para o roteador NATal, que é o responsável por essa tarefa.

Essa arquitetura, entretanto, não pode ser implementada com o modelo de roteamento e NAT atuais, uma vez que essas operações são realizadas considerando apenas os cabeçalhos das camadas de rede e de transporte, e a arquitetura proposta determina o nó de destino de cada mensagem baseado no valor das chaves. Desse modo, o protocolo NATal define um novo modelo de roteamento que utiliza também o cabeçalho da camada de aplicação para o encaminhamento dos pacotes.

No geral, a abordagem de realizar o roteamento baseado em dados da camada de aplicação mostrou-se bastante eficiente e flexível. Embora esse tipo de roteamento não seja suportado nos roteadores atuais, sua implementação no Linux constatou que a arquitetura de roteamento desse sistema, o Netfilter, é bastante adequada a esse tipo de extensão e não compromete o desempenho, uma vez que todas as tarefas são realizadas no próprio kernel sem a necessidade de comunicação com processos de usuário. Além disso, esse modelo não apenas simplifica bastante a realização de operações de NAT, pois não requer a criação de tabelas de conexões dinâmicas, como também resolve o problema para a utilização das aplicações P2P quando as máquinas em comunicação estão ambas em redes privadas. Problema esse que é conhecido como transversal NAT. Entretanto, essas considerações sobre o NAT aplicam-se apenas ao protocolo UDP, que é o protocolo de transporte utilizado pelo SGrid e pelo NATal.

8.1 Contribuições da Tese

As contribuições desse trabalho são:

- Proposta de uma arquitetura de rede P2P que apresenta os seguintes aspectos como diferencial das arquiteturas existentes atualmente:

- Definição de um mecanismo simples e eficiente para a criação da topologia da rede, que considera as distâncias entre dois nós nas redes IP, de modo que esses nós estejam próximos na rede P2P, sem necessitar da troca de informações dinâmicas.
- Redução significativa das mudanças de topologia na rede P2P.
- Eliminação do tráfego de dados que passa por dentro das redes das organizações, mas que não é destinado a nenhuma de suas máquinas.
- Redução significativa do processamento realizado por cada nó da rede, e, conseqüentemente, das exigências necessárias em termos de recursos computacionais que um nó precisa atender para participar eficientemente da rede P2P.
- Aumento no nível de confiabilidade da rede, uma vez que nós comuns não podem (ou dificilmente conseguirão) prejudicar o desempenho da rede como um todo.
- Utilização de uma estrutura em grade com níveis hierárquicos, que proporciona um ambiente propício a construção de protocolos de busca utilizando técnicas que podem se beneficiar dessa estrutura, como, por exemplo, *bloom filters* [53] e árvores *quadtree* [62].
- Solução para o problema da comunicação direta entre máquinas em redes privadas diferentes, conhecido como transversal NAT. Entretanto, o modelo proposto aplica-se apenas ao protocolo UDP.

- Desenvolvimento de um protótipo da rede P2P que permite simulações para análise do comportamento do modelo proposto.
- Especificação dos protocolos NATal e SGrid.
- Desenvolvimento de um módulo para o kernel do Linux, utilizando a arquitetura do NetFilter, que implementa o protocolo NATal e permite, portanto, que máquinas com esse sistema operacional sejam utilizadas como roteadores NATal.

8.2 Trabalhos Futuros

Como extensões desse trabalho podemos citar os seguintes pontos:

- Suporte a múltiplas aplicações P2P, uma vez que a implementação atual suporta apenas a criação de uma rede *overlay*. Embora a utilização de várias aplicações sobrecarregue o roteador, pois requer uma tabela de roteamento para cada aplicação, essa extensão é indispensável para ambientes reais. A arquitetura atual já foi projetada visando essa extensão, de modo que as modificações necessárias consistem basicamente em adicionar um campo nas mensagens para indicar a aplicação e modificar o código do módulo de roteamento para associar tabelas diferentes a cada um desses códigos, ou seja, a cada uma das aplicações.
- Definição de um esquema de propagação das tabelas de roteamento. Embora a arquitetura atual evite que o tráfego de roteamento passe por dentro da rede das organizações, se o roteador NATal utilizado for da própria organização, as mensagens passam pelo *link* de acesso à Internet para chegar até o roteador. Desse modo, é interessante definir um esquema de propagação das tabelas de roteamento P2P entre roteadores NATal, de modo que um provedor de *backbone* que atende a diversas empresas possa receber as tabelas dos roteadores NATal de seus clientes e só repassar para os mesmos as mensagens referentes às chaves da sua organização.
- Definição de um mecanismo para autenticação dos nós. Atualmente não existe nenhum esquema de validação das mensagens, o que compromete a segurança do sistema.
- Desenvolvimento de um mecanismo de cache hierárquico, aproveitando o modelo de níveis hierárquicos do SGrid, para permitir que as pesquisas sejam atendidas por nós mais próximos do nó que faz a pesquisa. Possivelmente esse mecanismo de cache será construído utilizando as idéias apresentadas em [67].
- Análise da viabilidade da utilização dos roteadores NATal e dos LSPs, em outras arquiteturas, como as redes Chord, por exemplo.

- Definição de um esquema de verificação de roteamento para identificar nós mal-intencionados que realizem essa operação de forma incorreta. Como cada nó conhece a chave pesquisada e o endereço IP de quem repassou o pacote, portanto é possível calcular o caminho, em termos de coordenadas da grade, que a mensagem deve percorrer. Isso permite que cada nó realize várias validações de segurança, como, por exemplo, ao receber uma mensagem verificar se esse nó realmente era o próximo destino da mensagem.
- Desenvolvimento de um protocolo de localização de serviços utilizando a arquitetura do SGrid. Uma das propostas consiste em utilizar *Bloom Filters* hierárquicos, onde os níveis dos *Bloom Filters* correspondem aos níveis do SGrid, e o *bloom filter* de cada nível seria propagado para o nó vizinho naquele nível. Desse modo, as mensagens de propagação desses filtros poderiam ser enviadas anexadas às próprias mensagens circulando na rede.
- Desenvolvimento de uma API para ser utilizada pelas aplicações oferecendo apenas as seguintes funções: *Join*, *Leave* e *Search*, para inserir o nó na rede P2P, remover o nó da rede P2P e pesquisar pelo nó responsável por uma chave, respectivamente. A função *search* deve possuir como parâmetros as coordenadas x e y da chave a ser pesquisada e as outras duas funções não terão nenhum parâmetro.
- Desenvolvimento de uma aplicação P2P completa que permita a publicação e pesquisa por nomes de arquivos. Além do desenvolvimento de uma aplicação de monitoramento e controle remoto que permita verificar os nós na rede e executar operações remotamente nesses nós.

9 Referências Bibliográficas

[1] Litzkow, M., Livny, M. and Mutka, M. Condor - a hunter of idle workstations. In Proceedings of the 8th International Conference on Distributed Computing Systems (ICDCS), pp.104-111, June1988.

[2] Buyya, R. High Performance Cluster Computing: Architectures and Systems. Prentice Hall. 1st, ed. Vol. 1. 1999.

[3] Foster, I., Kesselman, C., and Tuecke, S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. In Int. J. Supercomputer Applications, pp. 15-18, 2001.

[4] Foster, I. What Is the Grid? A Three Point Checklist. Grid Today, Vol. 1, No. 6, July 2002.

[5] Werthimer, D., Cobb, J., Lebofsky, M., Anderson, D., and Korpela, E. SETI@HOME—massively distributed computing for SETI, Computing in Science and Engineering, vol.3 n.1, pp.78-83, January 2001.

[6] Foster, I. and Kesselman, C. The Globus Project: a Status Report. In Proc. of Seventh Heterogeneous Computing Workshop (HCW 98), IEEE Computer Society Press, pp. 4-18, March 1998.

[7] Andrade, N., Cirne, W., Brasileiro, F. and Roisenberg, P. OurGrid: An Approach to Easily Assemble Grids with Equitable Resource Sharing. Proceedings of the 9th Workshop on Job Scheduling Strategies for Parallel Processing, pp. 61-86, June 2003.

[8] Foster, I., Gieraltowski, J., and Gose, S. The Grid2003 Production Grid: Principles and Practice. Presented at 13th Intl. Symposium on High Performance Distributed Computing (HPDC13), pp. 236-245, Honolulu, Hawaii, 2004.

[9] Oram, A., Peer-to-Peer: harnessing the benefits of a disruptive technology. 1st ed. 2001, Beijing; Sebastopol, CA: O'Reilly. xv, 432.

[10] Aberer, K., Alima, L.O., Ghodsi, A., Girdzijauskas, S., Hauswirth, M.and Haridi, S. The essence of P2P: A reference architecture for overlay networks. The Fifth IEEE International Conference on Peer-to-Peer Computing (P2P'05), pp. 11-20, Konstanz, Germany, 2005.

[11] Yang, B. and Molina, H.G. Designing a Super-Peer Network. Proceedings of the 19th International Conference on Data Engineering (ICDE), pp. 49-60, Bangalore, India, 2003.

[12] Napster. http://www.napster.com/

[13] Napster protocol specification. http://opennap.sourceforge.net/napster.txt, April 2000.

[14] The Gnutella Protocol Specification v0.4 (Document Revision 1.2), June 15 2001. http://www9.limewire.com/developer/gnutella protocol 0.4.pdf.

[15] Ratnasamy, S., Shenker, S., and Stoica, I. Routing algorithms for DHTs: Some open questions. Presented at International Workshop on Peer-to-Peer Systems (IPTPS), pp. 45-52, Cambridge, USA, March 2002.

[16] Stoica, I., Robert I., Morris, R., Karger, D., Kaashoek, M. F. and Balakrishnan, H. Chord: A scalable peer-to-peer lookup service for internet applications. In Proceedings of the ACM SIGCOMM, pp. 149-160, 2001.

[17] Ratnasamy, S., Francis, P., Handley, M., Karp, R. and Shenker, S. A Scalable Content-addressable Network. In Proceedings of the ACM SIGCOMM '01 Conference, pp. 161-172, San Diego, California, August 2001.

[18] Rowstron, A., Druschel, P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. Presented at International Conference on Distributed Systems Platforms (Middleware), pp. 329-350, November 2001.

[19] Zhao, B. Y., Kubiatowicz, J. D. and Joseph, A. D. Tapestry: An infrastructure for fault-resilient wide-area location and routing. U.C. Berkeley, Berkeley Technical Report UCBCSD-01-1141, April 2001.

[20] Krishna P., Gummadi, R., Gribble, S., Ratnasamy, S. Shenker, S. and Stoica, I. The impact of DHT routing geometry on resilience and proximity. In Proceedings of SIGCOMM, pp. 381-394, Karlsruhe, Germany, September, 2003.

[21] Rhea, S., Roscoe, T. and Kubiatowicz, J. Structured peer-to-peer overlays need application-driven benchmarks. In Proceedings of the Second International Workshop on Peer-to-Peer Systems (IPTPS), pp. 56-67, February, 2003.

[22] Xu, J. On the fundamental tradeoffs between routing table size and network diameter in peer-to-peer networks. In Proceedings of the IEEE Infocom, March 2003.

[23] Mockapetris, P. and Dunlap, K. J. Development of the Domain Name System. In Proceedings of the ACM SIGCOMM, pp. 123-133, Stanford, CA, 1988.

[24] Rocha, J., Domingues, M. A., Callado, A., Souto, E., Silvestre G., Kamienski, C. A., and Sadok, D. Peer-to-Peer: Computação Colaborativa na Internet. Minicursos SBRC2004 (capítulo de livro), pp. 3-46, Maio 2004.

[25] Wang, C. and Li, B. Peer-to-peer overlay networks: A survey. Technical Report, Department of Computer Science, HKUST, February 2003.

[26] Stevens, R. TCP/IP Illustrated, Volume 1: The Protocols, Addison-Wesley, 1994.

[27] Schoder, D., Fischbach, K. and Schmitt, C. Core Concepts in Peer-to-Peer Networking, in: Subramanian, R.; Goodman, B. (Hrsg.): The Evolution of a Disruptive Technology, Idea Group Inc, Hershey, 2005.

[28] L. Gong. JXTA: A network programming environment. IEEE Internet Computing, vol. 5, pp. 88-95, 2001.

[29] Waterhouse, S. JXTA Search: Distributed Search for Distributed Networks. white paper, Sun Microsystems, Palo Alto, Calif., 2001.

[30] Bray, T., Paoli, J., Sperberg-Mcqueen, C. M., and Maler, E. Extensible Markup Language (XML) 1.0 (Second Edition) W3C Recommendation 6, October, 2000.

[31] Dabek, F., Zhao, B., Druschel, P., Kubiatowicz, J. and Stoica, I. Towards a Common API for Structured Peer-to-Peer Overlays. In Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS), pp. 33-44, February 2003.

[32] Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N. and Weerawarana, S. Unraveling the Web Services Web: an Introduction to SOAP, WSDL, and UDDI. In: IEEE Internet Computing, Vol. 6, Issue 2, pp. 86-93, 2002.

[33] Groove. Http://www.groove.net.

[34] Edwards, J. Peer-to-Peer Programming on Groove. Indianapolis: Addison-Wesley. 2002.

[35] AOL Instant Messenger. http://www.aim.com.

[36] Microsoft Messenger. http://messenger.msn.com.

[37] Yahoo! Messenger. http://messenger.yahoo.com.

[38] Clarke, I., Sandberg, O., Wiley, B. and Hong, T. W. Freenet: A distributed anonymous information storage and retrieval system. Lecture Notes In Computer Science, vol. 2009, pp.46-66, 2001.

[39] The Kazaa web site. http://www.kazaa.com.

[40] Leibowitz, N. Ripeanu, M. and Wierzbicki, A. Deconstructing the Kazaa Network. 3rd IEEE Workshop on Internet Applications (WIAPP'03), pp.112-119, San Jose, CA, June 2003.

[41] Becker, D., et al. Beowulf: A ParallelWorkstation for Scientific Computation. 24nd International Conference on Parallel Processing (ICPP), pp.11-14, Wisconsin, U.S.A., August 1995.

[42] Distributed.net. http://www.distributed.net/.

[43] Anderson, D. P., Cobb, J., Korpela, E., Lebofsky, M., and Werthimer, D. SETI@home: An Experminent in Public-Resource Computing. Communications of the ACM, vol. 45, No. 11, pp. 56-61, 2002.

[44] SETI@home: Search for Extraterrestrial Intelligence at Home. Space Science Laboratory, University of California, Berkley, 2002, http://setiathome.ssl.berkeley.edu/.

[45] Dabek, F., Kaashoek, F., M., Karger, D., Morris, R. and Stoica, I. Wide-area cooperative storage with CFS. In the Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01), pp. 202-215, Banff, Canada, October 2001.

[46] Fightaids@HOME. http://fightaidsathome.scripps.edu/

[47] Aberer, K. P-Grid: A Self-Organizing Access Structure for P2P Information Systems. Sixth International Conference on Cooperative Information Systems (CoopIS), pp. 179–194, Trento, Italy 2001.

[48] Kleinberg, J. The Small-World Phenomenon: An Algorithmic Perspective. In Proceedings of the 32nd ACM Symposium on Theory of Computing, pp. 163-170, 2000.

[49] Aberer, K., Datta, A. and Hauswirth, M. Route maintenance overheads in DHT overlays. In 6th Workshop on Distributed Data and Structures (WDAS), Lausanne, Switzerland, July 2004.

[50] Karger, D., Lehman, E., Leighton, F., Levine, M., Lewin, D., and Panigrahy, R. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In Proceedings of the 29th Annual ACM Symposium on Theory *of Computing*, pp. 654–663, El Paso, TX, May 1997.

[51] Lim, H., Hou, J. C. and. Choi, C.H. Constructing Internet coordinate system based on delay measurement. IEEE/ACM Transactions on Networking, vol. 13, no.3, pp.513-525, June 2005.

[52] Ng, T. E. and Zhang, H. Predicting Internet network distance with coordinates-based approaches. In Proceedings IEEE Annu. Joint Conf. IEEE Computer and Communications Societies (INFOCOM'02), pp. 170-179, New York, USA, June 2002.

[53] Mitzenmacher, M. Compressed Bloom Filters. In Twentieth ACM Symposium on Principles of Distributed Computing, pp. 144-150, August 2001.

[54] Stevens, R. UNIX Network Programming. Prentice Hall, 1990.

[55] Hightower, J. Borriella, G. Location systems for ubiquitous computing. IEEE Computer, 34(8):57-66. 2001.

[56] SHA1 Secure Hash Algorithm. http://www.w3.org/PICS/DSig/SHA1_1_0.html.

[57] Singh, S., Ramabhadran, S., Baboescu, F. and Snoeren, A. The case for service provider deployment of super-peers in peer-to-peer networks, June 2003.

[58] Srisuresh, P. and Holdrege, M. IP network address translator (NAT) terminology and considerations, August 1999. RFC 2663.

[59] Ford, B. Peer-to-peer (P2P) communication across middleboxes, October 2003. Internet Draft draft-ford-midcom-p2p-01.txt. http://midcom-p2p.sourceforge.net/draft-ford-midcom-p2p-01.txt.

[60] Droms,R. and Lemon, T. The DHCP Handbook, Sams, 2002.

[61] Guttman, E. Service location protocol: Automatic discovery of IP network services. Internet Computing, pp. 71-80, July/August 1999.

[62] Samet, H. Spatial Data Structures: Quadtrees, Octrees, and Other Hierarchical Methods. Addison-Wesley, Reading, MA,1989.

[63] The Netfilter Project. http://www.netfilter.org.

[64] Nejdl, W., Wolf, B., Qu, C., Decker, S., Sintek, M., Naeve, A., Nilsson, M., Palmer, M. and T. Risch. EDUTELLA: a P2P Networking Infrastructure based on RDF. In Proceedings of the 11th International Conference on World Wide Web. Hawaii, USA, May 2002.

[65] Lassila, O. and Swick, R., R. Resource Description Framework (RDF) Model and Syntax Specification, World Wide Web Consortium, Recommendation REC-rdf-syntax-19990222, February 1999.

[66] Heberlein, L. T. and Bishop, M. Attack Class: Address Spoofing. In 1996 National Information Systems Security Conference, pp. 371–378, Baltimore, MD, October 1996.

[67] Madruga, M., Batista, T. and Guedes, L. A Distributed Proxy Architecture for Service Discovery in Peer-to-Peer Networks. Proceedings of the 2005 IFIP International Conference on Intelligence in Communication Systems (INTELLCOMM'05), pp 201-210, Montréal, Canada, October 2005.

[68] Balazinska, M., Balakrishnan, H. and Karger, D. INS/Twine: A scalable peer-to-peer architecture for intentional resource discovery. In Proceedings of the First International Conference on Pervasive Computing, pp. 195–210, Zurich, Switzerland, August 2002.