

Fundamentos, Tecnologias e Tendências rumo a Redes P2P Seguras

Marinho P. Barcellos (marinho@acm.org)
Luciano P. Gasparly (paschoal@inf.ufrgs.br)

Abstract

This chapter discusses two very current themes and that have received substantial attention from both academy and industry: peer-to-peer (P2P) networks and security. In terms of research, this is due to the amount of technical challenges to be solved, whereas in industry it is important because of the huge popularization of P2P and the great concern with information security and its socioeconomic impact. We present an overview of research results and technologies related to security in peer-to-peer networks, based on examples taken from the vast spectrum of P2P systems in operation nowadays.

Resumo

Este capítulo versa sobre dois temas bastante atuais e que tem recebido significativa atenção tanto da comunidade científica como da indústria: Redes Peer-to-Peer (P2P) e Segurança. Na pesquisa, tal deve-se à grande quantidade de desafios a serem resolvidos. Na indústria, deve-se à enorme popularização de P2P, à constante preocupação com segurança da informação e ao impacto sócio-econômico decorrente de seu uso. Apresentamos um apanhado geral de resultados de pesquisa e tecnologias relacionadas à segurança em redes P2P, amparando-nos em exemplos provenientes da vasta gama de sistemas P2P disponíveis e em operação atualmente.

4.1. Introdução

Aplicações *peer-to-peer* (P2P) não estão mais limitadas a usuários domésticos, e começam a ser aceitas em ambientes acadêmicos e corporativos. Embora compartilhamento de arquivos e aplicações de mensagens instantâneas sejam os exemplos mais tradicionais, eles já não são os únicos a se beneficiar das vantagens potenciais de redes P2P. Por exemplo, sistemas de armazenamento de arquivos em rede, de transmissão de dados, de computação distribuída e de colaboração também têm tirado proveito dessas redes.

As razões pelas quais este modelo de computação é atrativo se desdobram em três. Primeiro, redes P2P são *escaláveis*, ou seja, lidam bem (eficientemente) tanto com grupos pequenos quanto com grupos grandes de participantes. Segundo, é possível *depend* (de *dependability*) mais do funcionamento

dessas redes, já que não possuem ponto central de falhas e resistem melhor a ataques intencionais como os de negação de serviço. Terceiro, redes P2P oferecem *autonomia* a seus participantes, possibilitando que *entrem e saiam* da rede de acordo com seu interesse e disponibilidade, bem como tomem suas decisões sem depender de entidades externas.

Embora redes P2P possam contribuir para o compartilhamento de recursos e a colaboração em larga escala, em ambientes geograficamente distribuídos, com controle descentralizado e acoplamento fraco, sua diversificação e disseminação são dificultadas pela atual falta de segurança. Segundo [Wallach, 2002], trata-se de um desafio tornar essas redes seguras.

Ao almejar que redes P2P sejam amplamente adotadas, elas precisam estar protegidas contra a ação de nodos maliciosos. Esses podem fornecer, propositalmente, respostas incorretas a requisições tanto no nível de aplicação quanto no de rede. No primeiro caso, retornando informações não verdadeiras em resposta a uma busca, na tentativa de censurar o acesso a determinados objetos. No segundo, fornecendo informações falsas sobre rotas, visando particionar a rede. Além dessas, atacantes podem realizar outras atividades maliciosas tais como análise de tráfego (inclusive em sistemas que buscam oferecer anonimidade) e censura naqueles que desejam prover alta disponibilidade.

Outro tipo de comportamento indesejado, manifestado por muitos usuários, consiste em tentar ganhar mais da rede P2P do que oferecer em troca. Essa disparidade pode ser expressa, por exemplo, na utilização de espaço em disco, quando o atacante deseja armazenar dados nos nodos da rede em quantidade bastante superior ao que ele próprio disponibiliza ao sistema. Situação semelhante ocorre quando o nodo malicioso se nega a utilizar sua largura de banda restrita para transmitir um objeto, forçando o requisitante a recuperá-lo de alguma outra réplica.

Problemas como os recém enumerados fazem da área de *segurança* um dos principais campos de estudo em redes P2P. Nesse contexto, os aspectos a serem explorados são dependentes do tipo de aplicação e do grau de segurança exigido. Os principais aspectos investigados (definidos de forma bastante geral) são os seguintes:

- **disponibilidade:** garante que uma entidade está pronta para uso quando necessária;
- **autenticidade:** determina se alguém (ou algo) é, de fato, quem (ou o que) afirma ser;
- **confidencialidade:** protege dados contra observação por entidades não autorizadas;
- **integridade:** protege dados contra modificação, seja ela maliciosa ou acidental;
- **autorização:** restringe, com base em direitos, o acesso a recursos;
- **reputação:** determina grau de confiança nas demais entidades de um sistema;

- **anonimidade:** mantém desconhecida a identidade de uma entidade;
- **negabilidade:** ofusca dados para permitir à entidade que os detém “negar” responsabilidade pelo seu armazenamento; e
- **não-repúdio:** evita que uma entidade negue responsabilidade por ações executadas.

Este capítulo aborda redes P2P e segurança, temas bastante atuais e que têm recebido significativa atenção tanto da comunidade científica quanto da indústria. Na pesquisa, tal deve-se à quantidade de desafios a serem resolvidos; na indústria, deve-se à grande popularização desse tipo de aplicação, à constante preocupação com segurança da informação e ao impacto sócio-econômico decorrente de seu uso. O capítulo apresenta um apanhado geral de resultados de pesquisa e tecnologias relacionadas à segurança em redes P2P, amparado em exemplos provenientes da vasta gama de sistemas disponíveis e em operação atualmente.

O restante do capítulo está organizado como segue. Na Seção 4.2 são revisados conceitos de redes P2P, incluindo suas características chave, tipos de aplicações e arquiteturas. Na Seção 4.3 são abordados os principais aspectos de segurança e as implicações de seu emprego em redes P2P. Para cada aspecto são introduzidos sua definição no contexto de P2P, vulnerabilidades associadas e mecanismos existentes para eliminá-las. Já na Seção 4.4 é descrito, agora de forma mais detalhada, um conjunto de propostas para solucionar os principais problemas de segurança identificados em redes P2P. Por fim, na Seção 4.5 são apresentadas as considerações finais e uma amostra dos desafios de pesquisa a serem superados.

4.2. Fundamentos de P2P

A literatura recente apresenta excelentes revisões sobre P2P, como por exemplo [Theotokis and Spinellis, 2004], [Lua et al., 2005] e [Rocha et al., 2004], que definem, categorizam e exemplificam sistemas P2P. Nesta seção, revisamos apenas os pontos principais de P2P ou que tenham reflexo na segurança dos mesmos, tornando o texto auto-contido e claro em relação à terminologia adotada, sem no entanto ser repetitivo em relação à literatura recente na área.

4.2.1. Características Chave e Definição

Não existe consenso na literatura sobre o que exatamente são sistemas P2P ou quais são as características imprescindíveis de tais sistemas. Originalmente, P2P se refere a um estilo de arquitetura distribuída que contrasta com a cliente/servidor: sistemas distribuídos completamente descentralizados, em que todos os nodos são equivalentes em termos de funcionalidade e tarefas que executam. Esta definição é purista e exclui diversas aplicações aceitas hoje em dia como P2P. Mais recentemente, P2P passou a ser associado a uma classe de aplicações que aproveita recursos como disco e CPU presentes nas bordas da Internet.

Segundo [Theotokis and Spinellis, 2004], as duas características chave de P2P são:

- compartilhamento direto de recursos entre nodos, sem a intermediação de um servidor centralizado (embora se admita o uso de servidores em tarefas com menor demanda computacional ou de comunicação);
- capacidade de auto-organização tolerante a falhas, assumindo conectividade variável e população transiente de nodos como norma, automaticamente adaptando-se a falhas tanto nas conexões de rede como em computadores.

Outro ponto chave em P2P é a noção de que o sistema é construído com base na colaboração (supostamente voluntária) dos participantes. Baseado nestas características, sistemas P2P podem ser definidos como segue.

“Redes Peer-to-Peer (P2P) são sistemas distribuídos consistindo de nodos interconectados capazes de se auto-organizar em “redes de sobreposição” (overlays) com o objetivo de compartilhar recursos tais como conteúdo (música, vídeos, documentos, etc.), ciclos de CPU, armazenamento e largura de banda, capazes de se adaptar a populações transientes de nodos enquanto mantendo conectividade aceitável e desempenho, sem necessitar da intermediação ou apoio de uma entidade central.”

Em um sistema P2P, nodos estabelecem ligações lógicas e interagem através das mesmas, formando uma “rede de sobreposição” ou *overlay* no nível de aplicação. No restante deste texto, o termo “overlay” é usado para denotar um conjunto de nodos interligados em um sistema, aplicação ou rede P2P. Em contraste, o termo “rede”, a não ser quando disposto em contrário, é usado para denotar a rede física subjacente.

4.2.2. Aplicações P2P

Considerando as características chave e a definição de P2P apresentadas, identificamos as seguintes categorias de sistemas e/ou aplicações P2P:

- **compartilhamento de arquivos** (*file sharing*): uma das mais simples porém mais populares aplicações de P2P, também é referenciada por certos autores como *distribuição de conteúdo*. Seu objetivo é permitir que usuários “publiquem” arquivos, cujo conteúdo permanece imutável e é disseminado para usuários quaisquer, geograficamente espalhados pelo mundo e potencialmente em grande número. Tipicamente, qualquer usuário pode publicar um arquivo no sistema, não havendo também restrições de leitura. Exemplos são Napster [OpenNap, 2006], Gnutella [Gnutella, 2006], KaZaa [Liang et al., 2004] e BitTorrent [Bittorrent, 2006];
- **sistema de armazenamento de arquivos em rede** (*network storage*): diferentemente do caso anterior, o conteúdo dos arquivos podem ser

modificados por usuários (não é imutável) e alterações devem, no caso de replicação, serem consistentemente propagadas a todas as réplicas. É tipicamente necessário controlar e restringir operações de escrita, e potencialmente as de leitura também. Exemplos são PAST [PAST, 2006], OceanStore [OceanStore, 2006], Ivy [Ivy, 2006] e JetFile [JetFile, 2006];

- **transmissão de dados ou overlay multicast:** neste caso, o overlay forma uma infraestrutura de comunicação que supre a ausência de suporte multicast nativo na rede, de forma a permitir que um mesmo conteúdo seja transmitido por um nodo e entregue a um número potencialmente grande de nodos (usuários) geograficamente espalhados pelo globo. Tal tecnologia tem sido usada para transmissão de eventos ao vivo. Um dos principais exemplos, neste caso, é o ESM - End System Multicast ([ESM, 2006]), que já foi usado para transmitir mais de trinta eventos, incluindo simpósios científicos populares como o ACM SIGCOMM e o IEEE INFOCOM (Conference on Computer Communications);
- **computação distribuída:** estas aplicações visam a execução de processamento intensivo através da exploração de capacidade ociosa (*cyber-foraging*) em computadores que fazem parte de sistemas de grade. Exemplos são o OurGrid [Andrade et al., 2004], Seti@Home [SETI, 2006] e Genome@Home [Genome, 2006]. Note-se que Seti@Home e Genome@Home têm sido categorizados como P2P na literatura, mas estão baseados em um modelo mestre/escravo onde um servidor central (mestre) aloca trabalho a computadores pessoais (escravos). Nesses sistemas, o controle de acesso e a gerência de reputação são dois mecanismos fundamentais e que, assim, tem recebido grande atenção da comunidade de pesquisa;
- **colaboração e comunicação entre usuários:** são aplicações que permitem que usuários se comuniquem através de voz (VoIP), mensagens de texto, imagens gráficas e arquivos em geral de forma direta, sem passar por um servidor. Exemplos são Skype [Skype, 2006] e aplicações de Instant Messaging como ICQ [ICQ, 2006], Jabber [Jabber, 2006], MSN Messenger [MSN, 2006] e Yahoo Messenger [Yahoo, 2006].

Além das categorias acima, existem outras aplicações que empregam conceitos de P2P, tais como sistema gerente de bases de dados distribuída, mas que fogem ao escopo deste trabalho. Para fins de terminologia e tratamento do tema, consideramos que um overlay P2P é composto por um conjunto de *nodos (peers)*, conectados através de *ligações* ditadas por protocolos em nível de aplicação, por sua vez baseados em protocolos de transporte. Nodos são identificados unicamente no overlay através de um *identificador* (ocasionalmente abreviado por *id*), podem armazenar e fornecer *objetos* (arquivos, blocos de dados) a outros nodos que originam requisições (*requisitores*), bem como oferecer serviços a outros nodos do overlay, incluindo serviços de comu-

nicação em *Instant Messaging* (IM) e processamento em aplicações de computação intensiva. Além disso, nodos colaboram na execução de protocolos de busca por objetos e serviços originados em outros nodos. Buscas são baseadas em *tabelas de roteamento* e *chaves*, que identificam unicamente um objeto ou serviço no overlay.

4.2.3. Organização do Overlay

Sistemas P2P podem ser classificados em duas categorias principais, quanto à organização do overlay: *estruturado* e *não-estruturado*. A organização de um overlay possui influência significativa em diversos aspectos, incluindo segurança, robustez e desempenho (vide Seção 4.3.1).

Essencialmente, a organização determina as regras, caso exista alguma, para alocação de objetos (ou suas chaves) a nodos, e os algoritmos de busca empregados. Em **sistemas não estruturados**, a topologia é determinada de forma *ad hoc*: a medida que nodos entram (e saem) do overlay, estabelecem ligações com outros nodos arbitrários. O posicionamento de objetos ou de serviços é, neste caso, completamente independente da topologia do overlay. Uma dificuldade associada com esse tipo de overlay é o processo de busca de objetos ou serviços. Métodos primitivos como *inundação* do overlay foram empregados nos primeiros sistemas P2P, tal como o Gnutella original. A ineficiência deste método fomentou a pesquisa e desenvolvimento com ênfase na escalabilidade. Como resultado, surgiram sistemas não estruturados que empregam estratégias mais eficientes em termos de uso de recursos, tal como *caminhada aleatória* [Gkantsidis et al., 2004] ou índices de roteamento [Tsoumakos and Roussopoulos, 2003], além de overlays escaláveis via Tabelas Hash Distribuídas (de DHT, *distributed hash tables*).

Já nos **overlays estruturados**, a topologia do overlay é ditada por um esquema de alocação de chaves a ids de nodos, de forma a associar um determinado objeto ou serviço a um nodo específico de forma determinística e conhecida globalmente no overlay. A DHT funciona como uma tabela de roteamento distribuída, permitindo que um objeto seja encontrado em um pequeno número de passos. No entanto, ao empregar DHTs é necessário uma correspondência perfeita entre a chave buscada e a chave oferecida como parâmetro na busca; em outras palavras, o nodo requisitor precisa conhecer perfeitamente a chave do objeto procurado, e nem sempre isso é possível. Além disso, alguns autores argumentam que a manutenção do overlay em populações altamente transientes é difícil. Segundo [Theotokis and Spinellis, 2004], overlays estruturados podem ser ainda separados em *infraestruturas* e *sistemas*, dependendo se o overlay é apenas uma infraestrutura de roteamento *escalável*¹ em nível de aplicação (tal como Chord, CAN, Pastry) ou trata-se de um sistema completo (tal como OceanStore, PAST e Kademia). No restante deste texto, não fazemos distinção entre infraestrutura e sistema, a não ser caso explicitamente

¹ que pode operar eficientemente independentemente de suas dimensões.

mencionado.

Um overlay estruturado pode ser modelado através de seus principais atributos. Combinando os modelos de [Sit and Morris, 2002] e [Srivatsa and Liu, 2004], tem-se a definição que segue. DHTs são constituídas tipicamente por uma *API de armazenamento* disposta sobre uma *camada de protocolo de busca*. Esta última possui seis propriedades: **P1**: um espaço de *chaves*, sendo uma chave o identificador unívoco de um objeto, tipicamente gerada usando uma função de *hash* como MD5 (*Message-Digest algorithm 5*) ou SHA1 (*Secure Hash Algorithm*); **P2**: um espaço de identificadores de nodos e um esquema de mapeamento; por exemplo, Chord usa espaço circular de ids, enquanto CAN usa espaço de coordenadas de dimensão d . Um id de nodo poderia ser o *hash* do seu IP, por exemplo; **P3**: regras para dividir o espaço de identificadores entre os nodos: uma DHT divide o espaço completo de identificadores de nodos entre os nodos existentes (ativos) em um dado instante; **P4**: regras para associar chaves a nodos particulares, pois cada nodo é responsável por determinadas chaves; **P5**: tabelas de roteamento por nodo, e um esquema de roteamento que preencha as entradas da mesma; **P6**: regras para atualização de tabelas em entradas e saídas de nodos do overlay; quando um nodo entra, ele assume responsabilidade de uma parte do espaço de ids pertencentes a outros nodos (e conseqüentemente, do espaço de chaves).

4.2.4. Principais Overlays Não Estruturados

Os principais exemplos de infraestruturas P2P que possuem uma organização não estruturada são descritos a seguir.

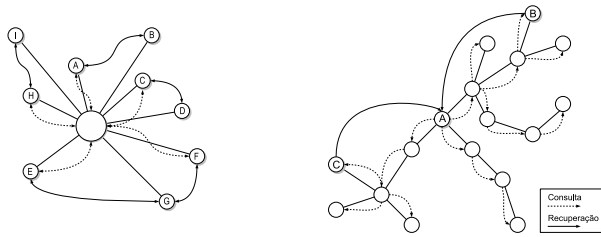
Napster. Precursor em termos de compartilhamento de arquivos, o Napster foi a “aplicação matadora” para disseminação de cultura P2P. Apesar disso, o Napster vai contra os princípios de P2P, porque depende de um servidor central para seu funcionamento. O Napster obteve enorme sucesso ao permitir o compartilhamento de arquivos de música. Predominantemente, o conteúdo publicado no Napster era protegido por *copyright* e não poderia ser copiado por outros usuários; hoje, o conteúdo disponibilizado pelo Napster é legal, porém o mesmo não possui mais tanto apelo. Usuários que ingressam na rede Napster oferecem conteúdo enviando informações sobre arquivos locais ao servidor central; operações de busca são resolvidas no servidor, que retorna ao nodo requisitor uma lista de endereços de nodos disponibilizando o arquivo procurado. O *download* do arquivo então se dá diretamente entre os nodos envolvidos, sem sobrecarregar o servidor. A arquitetura do Napster está exemplificada na Figura 4.1(a): computadores de usuários buscam informações sobre arquivos no diretório central e então se comunicam com outros nodos diretamente para obter os arquivos desejados. De acordo com estudos em [Comscore, 2006], Napster chegou a contar com mais de 26 milhões de usuários em 2001. Maiores informações sobre o Napster podem ser obtidas em [OpenNap, 2006].

Gnutella. É um sistema de compartilhamento de arquivos de topologia *ad hoc*.

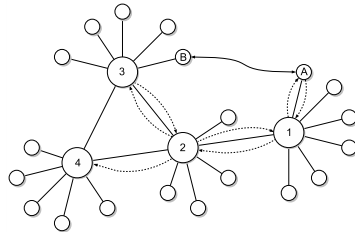
Todos os nodos são funcionalmente idênticos, ditos *servents*, porque são *servers* e *clients* ao mesmo tempo. Buscas por arquivos são realizadas através de uma inundação de escopo limitado (chamado “horizonte”). Nodos em que há um casamento entre o nome do arquivo especificado e o conjunto de arquivos publicados pelo nodo enviam uma resposta positiva, pelo caminho reverso no overlay. O nodo requisitor então escolhe um dos nodos que retornaram resposta e faz o *download* diretamente deste nodo. Não há garantia que um arquivo será localizado, mas o desempenho de buscas é bom para conteúdo popular. A Figura 4.1(b) mostra um exemplo de arquitetura Gnutella, onde um nodo faz uma inundação para localizar um arquivo, e uma vez encontrado, faz o *download* diretamente de um dos nodos que responderam (positivamente). Melhorias foram realizadas em versões mais recentes, através de uma hierarquia de dois níveis, com *supernodos* responsáveis por indexar informações de outros nodos, tal como ilustrado na Figura 4.1(c). Além disso, o esquema de buscas foi modificado de forma a diminuir o grau de inundação da rede. Segundo [Slyck, 2006], em abril de 2006 havia aproximadamente 2.219.539 de usuários na rede Gnutella. Maiores informações sobre o mesmo podem ser obtidas em [Gnutella, 2006].

FastTrack/KaZaa. É um sistema de compartilhamento de arquivos que emprega uma arquitetura de dois níveis, com nodos normais e *supernodos*. Os nodos normais se conectam a um supernodo, e supernodos se conectam entre si. Um nodo normal mantém uma lista com endereços de até 200 supernodos, enquanto um supernodo pode manter uma lista com milhares de endereços de supernodos. Quando se conecta à rede, um nodo envia a seu supernodo uma lista com a descrição dos arquivos que está disponibilizando. Um nodo envia uma busca a seu supernodo, que ou responde diretamente (quando sabe da localização do arquivo desejado) ou então executa uma busca enviando mensagens aos outros supernodos. Existe um certo grau de garantia na busca de arquivos, a medida que buscas são encaminhadas para os supernodos, oferecendo bom desempenho para conteúdo popular [Lua et al., 2005]. Em caso de falha de um supernodo, os nodos órfãos são passados a outros supernodos. A arquitetura do FastTrack está exemplificada na Figura 4.1(c) e se assemelha à versão atual do Gnutella: um nodo normal consulta seu supernodo sobre a localização de um arquivo, e caso existente, então solicita o arquivo desejado diretamente a outro nodo. Segundo estatísticas disponíveis em [Slyck, 2006], em abril de 2006 havia aproximadamente 3.144.691 de usuários concorrentes na rede FastTrack. Maiores informações sobre o FastTrack/KaZaa podem ser obtidas em [Liang et al., 2004].

BitTorrent. Sistema de compartilhamento de arquivos baseado em “enxames” (*swarms*) de nodos, que trocam diretamente blocos de arquivos mas são



(a) Napster & BitTorrent: nodos A, B, (b) Gnutella: nodo A faz uma busca C... consultam uma entidade central por inundação, encontra recurso por inundação, e após interage diretamente entre si.



(c) Gnutella/Overnet/eDonkey2000: nodo A consulta supernodo 1 sobre um dado recurso, que através de inundação é encontrado no nodo B conectado ao supernodo 3, e então solicita recurso diretamente ao nodo A.

Figura 4.1. Exemplos de P2P não estruturado

coordenados por um nodo central, o *tracker*. A lista de arquivos e suas propriedades (incluindo um *hash* de cada arquivo), bem como o endereço do *tracker* responsável pelo enxame, são especificados através de um arquivo denominado *torrent*. Este arquivo é preparado por um usuário que deseja publicar conteúdo, e disponibilizado em sites web especializados; um *torrent* precisa ser antes localizado por usuários interessados, o que ocorre procurando-se em site com *torrents* ou via máquinas de busca na web. Uma vez carregado o arquivo *torrent*, o usuário fornece-o ao software cliente, que se conecta ao *tracker*. O nodo informa ao *tracker* sobre seu interesse naquele *torrent*, e este responde com uma lista aleatória de nodos presentes no enxame. O nodo então

contacta múltiplos nodos, solicitando blocos de 512 KB do arquivo. O BitTorrent usa uma política de incentivo baseada em “olho-por-olho” (*tit-for-tat*), conforme descrito em [Cohen, 2003]: um nodo possui um número de vizinhos no enxame, e em princípio todos são “sufocados” (*choked*); então um nodo escolhe um número (por *default*, 4) de nodos entre seus parceiros de enxame, para os quais irá fazer *upload*. Na terminologia BitTorrent, diz-se que estes nodos serão *unchoked*. A escolha se baseia na taxa de *download* obtida dos vizinhos com quem o nodo interage, além de um *unchoking otimista* em que um nodo é escolhido periodicamente e de forma aleatória. O *tracker* monitora a disponibilidade dos nodos e dos pedaços do objeto nos mesmos; nodos são testados periodicamente, e um nodo que não responde é sucessivamente trocado. A arquitetura de um enxame BitTorrent está ilustrada na Figura 4.1(a): o elemento central representa o *tracker* do enxame, e a ele se conectam até várias centenas de nodos usuários que trocam dados entre si. Note-se que não há apenas um único ponto central, mas sim um número arbitrário de *trackers* espalhados pela Internet e que dividem a responsabilidade por milhares de *torrents*. Maiores informações sobre o BitTorrent podem ser obtidas em [Bittorrent, 2006, Jun and Ahamad, 2005].

Overnet/eDonkey2000. É uma arquitetura híbrida de duas camadas, composta por nodos “clientes” e nodos “servidores”, estes responsáveis por indexar informações sobre arquivos e participar das operações de busca. Ambos cliente e servidor são executados por usuários quaisquer. A Figura 4.1(c) apresenta um exemplo de arquitetura Overnet. As versões recentes de eDonkey implementam o protocolo Kamdelia [Maymounkov and Mazieres, 2002], típico de overlays estruturados. Segundo [Slyck, 2006], em abril de 2006 havia 3.736.358 usuários concorrentes nessa rede, porém o site do eDonkey [eDonkey, 2006] indicava na mesma ocasião apenas 920.387 usuários. Maiores informações sobre o Overnet/eDonkey2000 podem ser obtidas em [eDonkey, 2006].

Freenet. É um sistema para compartilhamento de arquivos com garantia de anonimato. Baseada fracamente em DHT, usa palavras-chave e texto descritivo para identificar objetos. Buscas são efetuadas de nodo em nodo através de chaves ou cadeias de caracteres contendo texto descritivo. O roteamento garante localizar objetos usando uma chave até que requisições excedam os limites de *hops-to-live*. Não existe uma hierarquia ou ponto central de falha. O Freenet é descrito em maior detalhe na Seção 4.4.4.

4.2.5. Principais Overlays Estruturados

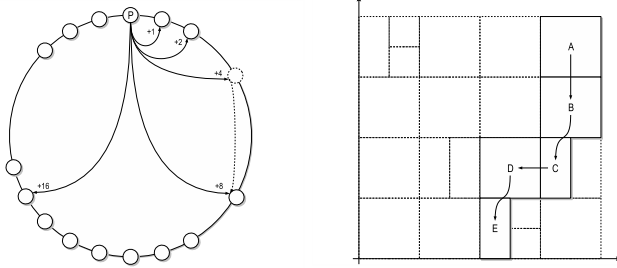
Os principais exemplos de infraestruturas P2P que possuem uma organização estruturada são descritos a seguir.

Chord. Infraestrutura de roteamento que usa *hashing* consistente SHA-1 para associar chaves de objetos a nodos em um espaço de ids (de nodos)

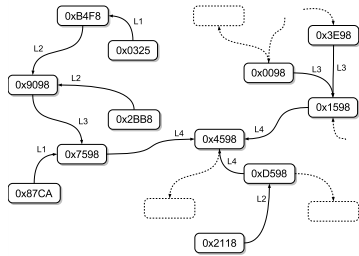
circular de m bits, ou seja, de 2^m identificadores. Identificadores de nodos são obtidos fazendo-se um *hash* do endereço IP, enquanto chaves de objetos são obtidas fazendo-se um *hash* da descrição do objeto. Uma chave k é associada ao nodo de identificador igual a k , ou se o mesmo não existe, o próximo nodo no anel (dito “nodo sucessor”). *Hashing* consistente permite a nodos entrarem e saírem da rede causando pouco estresse ao overlay P2P; quando um nodo ingressa na rede após o nodo n , ele assume a responsabilidade sobre uma parcela das chaves que estavam com n . Cada nodo mantém um apontador para os N nodos imediatamente sucessores e uma *finger table* com até m apontadores para outros nodos (logaritmicamente espalhados no anel). O roteamento de buscas é unidirecional ao longo do anel e pode ser *recursivo* ou *iterativo*. No modo recursivo, a mensagem vai sendo encaminhada de nodo em nodo e se aproximando do predecessor do objeto; quando chega no nodo com o objeto, a busca volta recursivamente ao origem. No iterativo, o nodo requisitante vai perguntando a nodos que ficam cada vez mais próximos do nodo com o objeto; quando o nodo com o objeto é perguntado, ele responde com os dados. A falha de nodos não causa uma falha global, podendo haver replicação de objetos em nodos consecutivos. A Figura 4.2(a) ilustra um exemplo de topologia Chord e uma operação de busca bem sucedida no anel. Um dos exemplos de uso do Chord é o Cooperative File System, CFS [Dabek et al., 2001b]. Maiores informações sobre o Chord podem ser obtidas em [Dabek et al., 2001a, Stoica et al., 2003, Chord, 2006].

CAN. *Content-Addressable Network* é uma infraestrutura descentralizada cujo princípio básico é o uso de um espaço de coordenadas cartesianas virtual de n dimensões em um multi-torus. O espaço de coordenadas é inteiramente lógico e serve para implementar a identificação de nodos e sua localização via tabelas de roteamento distribuído. Cada nodo é responsável por uma zona do espaço, que é dinamicamente determinada, e mantém uma tabela de roteamento com o endereço IP e as coordenadas de cada um de seus vizinhos no espaço. O protocolo de busca emprega pares $\langle \text{chave}, \text{objeto} \rangle$ para mapear um ponto P no espaço de coordenadas usando uma função *hash* uniforme, e coloca estas coordenadas nas mensagens. Uma mensagem é roteada em direção ao destino usando um encaminhamento simples ao nodo que está mais próximo das coordenadas. A falha de nodos não causa uma falha global; múltiplos nodos são responsáveis por um objeto, e quando há falha, a aplicação faz uma retentativa. A Figura 4.2(b) ilustra o espaço de coordenadas de uma rede CAN e o roteamento de uma mensagem em direção ao nodo destino, que é responsável pela chave procurada. Maiores informações sobre o CAN podem ser obtidas em [Ratnasamy et al., 2001].

Tapestry. Tapestry é uma infraestrutura P2P que permite o roteamento de mensagens a objetos (ou a cópia mais próxima a eles, se mais de uma



(a) Chord: um nodo P mantém apon- (b) CAN: busca através de coordena-
 tadores para outros nodos espalha- das em direção ao ponto x, y associ-
 dos no anel, permitindo uma busca ado ao recurso procurado, e que se
 eficiente e em tempo logarítmico. encontra abrigado na região sob res-
 ponsabilidade do nodo E.



(c) Tapestry: exemplo de uma fração de topologia Plaxton com os nodos e seus identificadores, onde L_n reflete o nível de coincidência da direita para esquerda.

Figura 4.2. Exemplos de P2P estruturado

cópia existe) de uma forma distribuída, auto-administrada e tolerante a falhas. As informações de roteamento e localização são distribuídas entre nodos da rede. A consistência da topologia é verificada dinamicamente e pode ser reconstruída em caso de perda. Tapestry é baseado nos mecanismos de localização e roteamento em malha propostos em [Plaxton et al., 1997]. Esta estrutura distribuída permite que nodos localizem objetos em uma rede de tamanho arbitrário usando mapas de roteamento pequenos e de tamanho constante. Na malha Plaxton original, os nodos podem assumir o papel de servidores (que armazenam objetos), roteadores (que encaminham mensagens) e clientes (que originam requisições). Cada nodo mantém um mapa de vizinhos; cada

mapa possui múltiplos níveis, cada nível n contendo apontadores para nodos cujo id deve casar em n dígitos. Cada entrada no mapa de vizinhos corresponde a um apontador para o nodo mais próximo na rede cujo id confere com o número no mapa, até uma posição de dígitos. Mensagens são incrementalmente roteadas através dos nodos dígito por dígito, da direita para a esquerda. Por exemplo, uma mensagem do nodo 67493 para o nodo 34567 poderia passar pelos seguintes nodos: $xxx7- \rightarrow xxx67- \rightarrow xx567- \rightarrow x4567- \rightarrow 34567$. A malha Plaxton usa um *nodo raiz* para cada objeto, que serve como uma garantia a partir do qual um objeto pode ser localizado. Quando um objeto o é inserido na rede no nodo ns , um nodo raiz nr é associado ao objeto usando um algoritmo determinístico global. Uma mensagem é então roteada de ns para nr , armazenando dados na forma de um mapeamento $\langle o, ns \rangle$ em todos os nodos ao longo do caminho. Durante uma operação de busca, mensagens destinadas a o são inicialmente roteadas com destino nr , até que um nodo seja encontrado contendo o mapeamento $\langle o, ns \rangle$. A Figura 4.2(c) demonstra um exemplo de roteamento de mensagem, extraído de [Zhao et al., 2001]. Maiores informações sobre o Tapestry podem ser obtidas em [ChimeraTapestry, 2006].

Pastry. Assim como o Tapestry, é uma infraestrutura de roteamento baseada em malha do estilo Plaxton. A diferença principal reside na abordagem para se obter localidade de rede e replicação de objetos. Pastry é empregado pelo sistema de armazenamento persistente de larga escala PAST [PAST, 2006, Rowstron and Druschel, 2001, Druschel and Rowstron, 2001] e no Scribe [Scribe, 2006], um sistema de comunicação em grupo e de comunicação de eventos de larga escala. Maiores informações sobre o Pastry podem ser obtidas em [Pastry, 2006].

Kademlia. Infraestrutura de roteamento que usa um mecanismo inovador para roteamento de mensagens e busca de objetos segundo uma métrica de distância entre identificadores de nodos (não de proximidade de rede) baseada em xor. A topologia tem a propriedade que toda a mensagem trocada carrega ou reforça informações úteis de contato. O sistema explora essa informação para enviar mensagens de busca assíncronas e paralelas que toleram falhas de nodos sem impor atrasos e *timeouts* a usuários. Diversas aplicações de P2P estão empregando o algoritmo Kademlia: Overnet, eDonkey e eMule, além de BitTorrent, que emprega Kademlia para permitir o uso de *torrents* sem um *tracker*. Maiores informações sobre Kademlia podem ser obtidas em [Maymounkov and Mazieres, 2002].

4.3. Segurança em P2P

Nesta seção são abordados os principais aspectos de segurança mencionados anteriormente: disponibilidade, autenticidade, reputação (e confiança),

autorização, integridade, anonimidade e negabilidade. Para cada um deles, são apresentados conceitos fundamentais e os desafios/implicações de seu uso em overlays P2P. São identificados e caracterizados aspectos de segurança associados particularmente a overlays P2P, ignorando quando possível questões mais amplas sobre segurança. Primeiramente, a seção apresenta considerações mais gerais sobre segurança em P2P, como o impacto da estrutura do overlay e considerações gerais sobre atacantes, após então tratando individualmente cada um dos aspectos de segurança citados.

4.3.1. Impacto da estrutura e topologia na segurança

A estrutura de um overlay P2P é questão chave de projeto e impacta sobremaneira no desempenho de operações comuns como busca de um objeto, inserção de novos objetos e manutenção do overlay (especialmente quando da entrada e saída de nodos). A estrutura do sistema P2P também é de grande importância para a segurança do mesmo. Não há um esquema que seja claramente superior a outro em relação à segurança; portanto, ao invés de indicar um esquema melhor, apontamos as principais questões envolvidas.

- Em overlays não estruturados, como é a topologia? A conectividade dos nodos obedece uma Lei de Potência, com poucos nodos muito conectados e muitos nodos pouco conectados? As ligações entre nodos seguem o fenômeno de “Pequeno Mundo”, com qualquer nodo sendo alcançável em no máximo alguns poucos saltos?
- Em overlays não estruturados, como são feitas as buscas, por inundação ou por caminhada aleatória? A inundação permite que um nodo gere consultas com identidade falsa, intercepte uma busca, e responda falsamente indicando um outro nodo como detentor de um objeto?
- O overlay possui algum tipo de hierarquia, com supernodos? Assumindo que supernodos concentram responsabilidade, indexando dados dos nodos de menor capacidade, um supernodo malicioso não tem uma capacidade de ataque maior? Pode a escolha de supernodos ser determinada com base em informações de desempenho falsificadas pelo nodo malicioso?
- Em um overlay estruturado, pode um nodo escolher seu identificador? Caso positivo, ele pode obter controle sobre uma chave desejada, ou seja, sobre um objeto que deseja atacar?

Em [Dumitriu et al., 2005] é analisado o impacto da topologia quanto à resistência a certos tipos de ataques. Segundo o autor, sistemas estruturados são mais resistentes a ataques que sistemas hierárquicos, porque os nodos do primeiro nível oferecem grande vulnerabilidade a ataques de negação de serviço. Mais precisamente:

- hierarquia de dois níveis: a existência de supernodos confere grande vulnerabilidade a ataques caso nodos maliciosos obtenham papel de su-

período (atualmente com Gnutella um nó pode se decidir supernó anunciando um enlace de grande capacidade);

- grafos *k-regulares* não hierárquicos: este tipo de estrutura está presente em diversas infraestruturas de P2P, como CAN, Chord, Pastry, Tapestry e Kademlia. Oferecem grande resistência a ataques pois os pontos de “colapso” para tais grafos tipicamente ocorrem apenas com caminhos de comprimento muito longo (por exemplo, maiores que 10), que ocorrem ou em sistemas de escala muito grande ou em redes que roteiam por caminhos bastante longos propositalmente, como por exemplo para obter anonimidade (vide Seção 4.4.2);
- grafos de Lei de Potência, que ocorrem em uma série de situações, incluindo Freenet: representam uma vulnerabilidade aguda a negação de serviço caso os nós maliciosos possam tomar controle sobre um dos nós com alto grau no grafo (ditos *hubs*).

4.3.2. Ações de adversários e premissas

Estudos sobre a segurança de overlays P2P usualmente criam um *modelo de ataque*, e listam premissas sobre o ambiente e quanto aos poderes do atacante. Um atacante participa (ou deseja participar) de um overlay P2P conectando ao mesmo um ou mais nós maliciosos, sob seu controle (note-se que isso pode ocorrer de forma alheia à vontade de um usuário, através de uma contaminação do software P2P). Tais nós, ao contrário dos corretos, deixam de seguir os protocolos de roteamento e manutenção do overlay. Tipicamente, assume-se que nós maliciosos possam fazer *conluíus* para atacar um overlay P2P em conjunto. É possível que tais nós estejam geograficamente espalhados. Tal pode ser obtido, por exemplo, através da distribuição de software para acesso P2P contendo um cavalo de tróia.

Em certos casos, o sistema de autenticação é fraco e um nó malicioso é capaz de obter para si múltiplos identificadores, permitindo a um único nó se passar por diferentes nós. Estes nós virtuais são chamados de *Sybilis*, e este tipo de ataque é explorado na Seção 4.3.4.

Sobre a camada de rede subjacente e o sistema operacional, pode-se assumir ou não que as mesmas sejam seguras, no sentido que um nó não pode observar pacotes que não lhe são endereçados.

A seguir, discutimos os aspectos mais relevantes sobre segurança em P2P. O primeiro aspecto é da **disponibilidade**, e como a mesma pode ser afetada através de ataques de **negação de serviço** e ao sistema de **roteamento do overlay**. O impacto de tais ataques é alto, pois tem o poder de comprometer não apenas o funcionamento de determinados nós, mas do sistema como um todo. Ataques de roteamento estão ligados à **autenticidade**, tratada na Seção 4.3.4, pois determinadas estratégias dependem de um nó malicioso assumir múltiplas identidades. A seguir, discutimos a questão de **reputação e confiança** e como um nó pode julgar o grau de confiança em outro de acordo com o histórico de ações do mesmo. A Seção 4.3.6 trata do conceito

de **autorização** e como o mesmo é usado para o provimento de mecanismos de controle de acesso entre nodos. A seguir, discutimos como **integridade** pode ser obtida em sistemas P2P, através de mecanismos de criptografia e a técnica de *dispersão de dados*. Por fim, tratamos de duas questões interrelacionadas e de fundamental importância em P2P: **anonimidade** e **negabilidade**; o primeiro se refere à capacidade de publicar e buscar conteúdo ou solicitar serviços de forma anônima, enquanto o segundo à ausência de responsabilidade pelo conteúdo armazenado ou colaboração no funcionamento do overlay.

4.3.3. Disponibilidade

No contexto de P2P, disponibilidade está relacionada à parcela de tempo em que um objeto ou serviço está acessível aos demais nodos. No compartilhamento de arquivos e armazenamento em rede, é relativa ao sucesso nas operações de leitura e escrita de dados. Para computação distribuída, nodos que participam de uma determinada computação devem estar disponíveis de forma a executar o que se espera dos mesmos.

A técnica básica para se obter disponibilidade é a replicação. A seguir, são exploradas vulnerabilidades e ataques relativos à disponibilidade: a negação de serviço e os ataques de roteamento.

4.3.3.1. Negação de Serviço

Ataques DoS podem ocorrer em nível de rede e em nível de aplicação, conforme descrito em [Daswani and Molina, 2002]. No nível de rede, consiste em executar um DoS convencional sobre uma máquina ou enlace de acesso de uma máquina que executa um nodo P2P. Já no nível de aplicação, existe uma série de ataques ao overlay P2P; o tipo de ataque varia com a arquitetura P2P adotada, estruturada ou não estruturada.

Uma forma de ataque DoS é fazer que um ou mais nodos maliciosos enviem muitas mensagens de busca, sobrecarregando o overlay P2P. Um exemplo disso são os ataques de DoS em nível de aplicação a overlays P2P Gnutella (com supernodos), descritos em [Daswani and Molina, 2002].

Em [Sit and Morris, 2002] são citados diversos tipos de ataques, comentados a seguir. Os três primeiros são diretamente relacionados à negação de serviço. O primeiro consiste em um nodo malicioso funcionar corretamente para buscas, mas quando solicitado, negar o serviço, ou seja, a existência de um objeto sob sua responsabilidade (ou se recusar a enviar uma resposta). Este tipo de ataque, que é trivialmente detectável por nodos corretos, pode ocorrer tanto em overlays estruturados como não estruturados. Como defesa, um overlay P2P pode implementar replicação (na camada de armazenamento). Em geral, deve-se evitar pontos únicos de responsabilidade, e replicação pode permitir que não exista um único nodo responsável pela replicação ou acesso às réplicas.

O segundo tipo de ataque descrito é a sobrecarga de nodos específicos,

através de um ataque DoS convencional. Neste caso, um nodo correto sob ataque ficaria incomunicável, fazendo com que o mesmo fosse eliminado do overlay por outros nodos. Este ataque deve ser combatido através da alocação aleatória de identificadores, uso de réplicas de objetos e serviços, e sua dispersão física na rede.

O terceiro tipo de ataque relacionado em [Sit and Morris, 2002] é o de entrada e saída acelerada de nodos (*churn* excessivo). Em sistemas de alta disponibilidade, que buscam manter seus objetos sempre disponíveis apesar da ausência de certos nodos, é necessário que objetos sejam copiados, ou do nodo que deixa o overlay em caso de saída proposital, ou de réplicas em caso de falha. Como estas operações possuem custo associado, um ataque que provoca a entrada e saída rápida de nodos tem o potencial de sobrecarregar determinados nodos ou um segmento da rede e causar uma negação de serviço aos demais nodos do overlay P2P. Entretanto, se os nodos maliciosos precisam se envolver nestas operações, então seus próprios recursos seriam exauridos, o que reduziria bastante o poder de um atacante.

Em [Sit and Morris, 2002] descreve-se ainda o ataque de mensagens não solicitadas, onde um nodo malicioso engenha uma situação em que envia mensagens de resposta não solicitadas, interferindo por exemplo em buscas. A melhor defesa para este tipo de ataque seria empregar técnicas padrão de autenticação, tal como assinaturas digitais ou códigos de autenticação de mensagens (MACs). Como assinaturas digitais são dispendiosas computacionalmente e MACs requerem chaves compartilhadas, *nonces*² podem ser adicionados a mensagens como forma de defesa, exigindo-se que o mesmo valor fornecido na mensagem de requisição seja incluído na mensagem de resposta.

Por fim, [Dumitriu et al., 2005] descreve o ataque de nodo lento (*slow node attack*), no qual um nodo malicioso intercepta respostas e modifica as mesmas indicando um nodo não-malicioso mas de pouca capacidade, e aumenta falsamente a capacidade daquele nodo. Adicionalmente, um nodo malicioso pode descartar mensagens de nodos rápidos. Este ataque de DoS faz com que requisições sejam atrasadas.

4.3.3.2. Ataques de Roteamento

Essa classe de ataques são anomalias de roteamento em que mensagens transmitidas através do overlay são desviadas para longe de seu destino, para nodos maliciosos, ou descartadas. Este tipo de ataque aumenta a chance de falhas na busca, e possui impacto negativo no custo e desempenho do sistema. Para que este tipo de ataque ocorra, as informações de roteamento ou sobre outros nodos em nodos corretos são “envenenadas” por nodos maliciosos. Isto ocorre, por exemplo, quando nodos maliciosos respondem com rotas incorretas

² derivado de *number used once*, indica número usualmente aleatório empregado para evitar que uma mensagem prévia possa ser usada em um ataque.

a mensagens de busca.

Em [Sit and Morris, 2002] são identificadas três categorias de ataque de roteamento, como segue. Primeiro, o roteamento incorreto de buscas, quando um nodo malicioso encaminha mensagens de busca a um nodo incorreto ou inexistente. Segundo, atualizações incorretas de rotas provocadas por nodos maliciosos que fornecem informações falsas sobre rotas a nodos corretos. O terceiro tipo de ataque de roteamento é o particionamento do overlay, que pode ocorrer quando um nodo correto, ao entrar no overlay, carrega sua tabela de rotas de um nodo de inicialização (dito nodo de *bootstrap*) malicioso, criando potencialmente uma partição separada controlada pelos nodos maliciosos, mas contendo nodos corretos.

Em [Castro et al., 2002] e [Wallach, 2002] são apresentados ataques a mecanismos de roteamento que buscam assumir o controle da comunicação em um overlay P2P. Em [Singh et al., 2004], este tipo de ataque é batizado como "Eclipse": o atacante possui um grande número de nodos (ou de identificadores) no overlay P2P de forma que os corretos são "eclipsados" pelos maliciosos e o overlay deixa de funcionar corretamente. Mais recentemente, em [Singh et al., 2006] há uma análise mais detalhada desse tipo de ataque e uma avaliação de desempenho considerando possíveis contra-medidas.

Um ataque Eclipse pode ser feito tanto contra overlays estruturados como não-estruturados. Os trabalhos [Sit and Morris, 2002], [Wallach, 2002] e [Castro et al., 2002] foram os primeiros a abordar questões de segurança em overlays estruturados. Vulnerabilidades de roteamento estruturado aparecem também em [Srivatsa and Liu, 2004]. Em [Xuan et al., 2003] é proposto o RChord, acrescentando ligações em sentido contrário no anel do Chord, e evitando ataques de roteamento que levam as mensagens para longe do destino. Em [Danezis et al., 2005] sugere-se uma estratégia contra nodos virtuais falsos (*Sybils*) que assegura que buscas usem um conjunto diverso de nodos, assim evitando a concentração de mensagens em nodos maliciosos e diminuindo-se o impacto deste tipo de ataque.

Já em overlays não-estruturados, o ataque de Eclipse ocorre de forma diferente, pois nestes overlays o roteamento é feito através de inundação de mensagens ou caminhadas aleatórias. Quando um nodo correto envia mensagens, ele o faz através de seus vizinhos. Um ataque de roteamento neste caso pode ser obtido fazendo com que um nodo correto tenha ligações apenas com nodos maliciosos. Estas informações são fornecidas por outros nodos, no momento da entrada do nodo no overlay (processo denominado *bootstrap*) bem como em função da entrada e saída de nodos. No Gnutella, um nodo mantém uma lista de vizinhos, denominada *pong cache*, que é usada também para descoberta de novos nodos no overlay. O envenenamento da *pong cache* é a contaminação com identificadores de nodos adversários. [Daswani and Molina, 2004] mostra como ataques coordenados a nodos no Gnutella podem fazer com que as *pong caches* sejam preenchidas com valores incorretos, afetando disponibilidade e autenticidade.

[Sieka et al., 2004] trata do mesmo problema, com ênfase nos protocolos de *polling* que são empregados por nodos no Gnutella para descoberta de outros nodos (ou seja, para o preenchimento da *pong cache*).

Existem duas formas de efetuar um ataque do tipo Eclipse. A primeira ocorre através da manipulação do algoritmo de manutenção do overlay. Diversos trabalhos exploram essa abordagem. A segunda maneira consiste em, através de um ataque Sybil [Douceur, 2002], se obter um grande número de identificadores que falsamente indicam um pequeno número de nodos maliciosos (a questão autenticidade é aprofundada na próxima seção). Em ambos os casos, nodos corretos são levados a encaminhar mensagens a nodos maliciosos.

4.3.4. Autenticidade

Um nodo possui três fontes principais de informação sobre a identidade de outros nodos [Douceur, 2002]: uma agência de confiança, o próprio nodo, ou outros nodos (que não são confiáveis). Um nodo pode aceitar as identidades apenas de nodos que ele já validou diretamente (de alguma forma) em uma ocasião anterior, ou então também aceitar identidades de nodos “apoiadas” (*vouched*) por outros nodos que ele já validou.

Em [Douceur, 2002] é identificado o *ataque Sybil*: a falsificação de múltiplas identidades em um sistema. O artigo mostra que em um sistema P2P de larga escala não é possível evitar, pragmaticamente, que um nodo malicioso gere múltiplas identidades. A única forma realmente segura de se gerenciar identidades de nodos é através de uma autoridade certificadora de confiança. Esta autoridade pode ser explícita, tal como uma Autoridade de Certificação (por exemplo, VerySign), ou pode ser implícita, tal como ocorre no Farsite ([Farsite, 2006]).

Se um mesmo nodo pode assumir múltiplas identidades, então os mecanismos de segurança e confiabilidade baseados em replicação ficam comprometidos, pois as múltiplas réplicas de um objeto podem ficar sob controle de um nodo malicioso. São muitos os exemplos de sistemas P2P que se baseiam na replicação de tarefas computacionais ou de armazenamento, ou na fragmentação de tarefas, entre diversos nodos. Em [Srivatsa and Liu, 2004] os resultados de [Douceur, 2002] são estendidos e mostra-se que se a geração arbitrária de identificadores (*pseudo-spoofing*) não é controlada pelo sistema, então um atacante pode facilmente alvejar um objeto.

O problema da autenticação em P2P pode ser resolvido através de uma Agência Certificadora ou outro elemento centralizado em que se confie na atribuição de identidades. No entanto, tal esquema não é desejável em um sistema distribuído, particularmente em um overlay P2P com potencial para lidar com milhões de nodos, por ser um ponto central de falha e um potencial “gargalo”. Existem diferentes alternativas, tal como usar uma *identidade fraca*: fazer um *hash* de um identificador externo do nodo, tal como o seu endereço IP, ao gerar o id (como no CFS [Dabek et al., 2001b]).

Uma vulnerabilidade do esquema com identificadores externos, conforme [Srivatsa and Liu, 2004], é que se um atacante determina seu próprio identificador (e não uma autoridade), o nodo malicioso pode engenhosamente alocar o id do nodo responsável pela chave associada ao objeto alvo, como explicado a seguir. Dado um objeto com chave k como alvo, o ataque consiste em encontrar um id para o nodo (se o id interno é gerado em função do id externo, então encontrar o id externo) de forma que k seja mapeada para o id escolhido. Inverter uma função de *hash* é muito caro computacionalmente, mas o nodo adversário, denotado como p , pode encontrar o id do nodo responsável por k , denotado como q , através da função usual de busca. De posse de q , então o nodo malicioso encontra o id externo que mapeia para q . Com isso, o atacante pode entrar no overlay usando o id externo de q , e acaba por dividir a responsabilidade por objetos com q , se apoderando de parcela das chaves do mesmo. Existe uma boa chance de o nodo atacante p receber a responsabilidade por k .

Outra alternativa é o uso de um servidor de *bootstrap*, responsável por determinar, de forma aleatória, um identificador para cada nodo que deseje ingressar no overlay. Diferentemente do caso anterior, um nodo não pode alocar seu próprio id, evitando o ataque acima. Por outro lado, ainda é possível que um atacante obtenha, através de requisições consecutivas ao servidor de *bootstrap* (fingindo diferentes endereços IP para enganar o servidor), um conjunto de ids válidos. Portanto, mesmo que a obtenção de ids seja mais demorada, por exemplo obrigando o solicitante a resolver uma charada criptográfica (*crypto-puzzle*) para limitar a taxa de alocação de ids [Castro et al., 2002], a vulnerabilidade continua a existir porque o atacante poderia obter um grande número de ids (dominando o espaço de ids), e mais cedo ou mais tarde obtendo acesso à chave k .

4.3.5. Reputação e Confiança

O funcionamento eficiente e correto de um sistema P2P depende da participação voluntária de seus usuários. Quando os nodos de um sistema P2P não colaboram, as conseqüências são sérias e podem causar danos individuais a usuários ou mesmo levar o sistema ao colapso. Sistemas tradicionais assumem que usuários são “obedientes”, que aderem a um protocolo especificado sem questionar a utilidade do mesmo para si. Esta obediência não é uma premissa realística em sistemas P2P [Feldman and Chuang, 2005]. É necessário, portanto, que o sistema controle de certa maneira os nodos, responsabilizando-os pelas suas ações quando eles se recusam a colaborar, e recompensando-os quando colaboram adequadamente. Popularmente, nodos que usam muito mais recursos do que oferecem (quando oferecem) são conhecidos como *free-riders* ou “nodos carona”. Estudos em [Adar and Huberman, 2000, Saroiu et al., 2002] mostram que *free-riders* são comuns em diversas aplicações P2P de compartilhamento de arquivos, e a explicação para tal fenômeno está relacionada à “Tragédia dos Comuns”

[Harding, 1968], que argumenta que as pessoas tendem a abusar do uso de certos recursos se elas não têm que pagar por eles de alguma forma.

Segundo [Marti and Garcia-Molina, 2006], nodos de um overlay P2P que não colaboram podem ser divididos em *egoístas* e *maliciosos*. O objetivo de egoístas é usufruir o máximo do sistema P2P contribuindo o mínimo de recursos para o mesmo. Em contraste, o objetivo dos maliciosos é causar mal a um dos nodos ou ao sistema como um todo, como explorado nas demais subseções deste capítulo. Nodos maliciosos estão dispostos a empregar recursos no ataque, tal como injetar arquivos corrompidos, o que não ocorre com nodos egoístas.

Existem diferentes abordagens para se incentivar nodos a colaborarem. [Theotokis and Spinellis, 2004] identifica duas classes gerais de mecanismos de incentivo: baseados em confiança (*trust*) e reputação, e baseados em comércio (*trade*), que inclui mecanismos de micro-pagamento (em que um nodo do overlay P2P que oferece um serviço para outro é explicitamente renumerado) e esquemas de troca de recursos. Uma terceira categoria é mencionada em [Feldman and Chuang, 2005]: generosidade inerente, onde usuários decidem se contribuem para o sistema ou não baseados no nível global de contribuição dos demais usuários.

Um exemplo de esquema baseado em comércio é o MojoNation [Mojo, 2006], em que usuários que oferecem recursos como tempo de processamento e espaço em disco podem acumular unidades de *mojo*, e posteriormente, gastá-las. Outro é a “rede de favores” [Andrade et al., 2004] proposta para o sistema de grade P2P OurGrid, onde a decisão de acolher ou não uma tarefa de um nodo remoto é tomada segundo o histórico anterior entre os nodos envolvidos.

Os esquemas de confiança e reputação estão amparados na *reciprocidade* [Feldman and Chuang, 2005]: cada nodo possui uma reputação associada, que parte de um estado inicial e é construída ao longo da vida de um nodo, com base em suas interações com outros nodos. Essa informação de reputação pode influenciar a decisão de um nodo sobre os parceiros de sua próxima interação, buscando reciprocidade. Por exemplo, um nodo pode preferir solicitar um serviço a um nodo que tem lhe executado correta e eficientemente as últimas solicitações, ou deixar de considerar certos nodos que têm retornado arquivos com conteúdo corrompido. Reciprocidade é obtida em um sistema P2P através de um **sistema (de gerência) de reputação**.

O termos reputação e confiança (*trust*) estão intimamente ligados. Há variação considerável nas definições de confiança na literatura, não havendo consenso sobre seu significado. Segundo [Grandison and Sloman, 2000], confiança pode ser definida como a “firme crença na competência de uma entidade em agir de forma confiável e segura, em que se possa depender, dentro de um contexto especificado.” Ainda segundo Grandison, a confiança é usualmente especificada em termos de uma relação entre uma entidade que confia, o fiador (*trustor*), e outra que é confiada, o depositário (*trustee*). Confiança é a

base para permitir que um depositário use ou manipule recursos de um fiador, ou pode afetar a decisão de um fiador em usar um serviço de um depositário. O grau de confiança em uma operação tem relação inversamente proporcional ao seu risco. Sobre a relação entre os termos, um sistema de gerência de reputação determina a reputação de nodos baseado no histórico das ações dos mesmos e permite que sejam formadas opiniões sobre o grau de confiança a cerca de outros nodos.

As relações de confiança podem ser de um para outro (confiar em um nodo para execução de um serviço), de um para vários (confiar em um conjunto de nodos com quem informações podem ser seguramente trocadas), de vários para um (tal como confiar em um líder), e de vários para vários (quando um grupo confia em outro). Uma das propriedades importantes de confiança é a transitividade: se A confia em B e B confia em C , então é possível que A confie (pelo menos limitadamente) em C . Este nível de confiança pode ser expresso em escalas discretas ou contínuas. No primeiro caso, os valores seriam como “baixo”, “médio” e “alto”, enquanto no segundo poderiam ser valores no intervalo contínuo $[0, 1]$. Mas neste último caso, como representar ignorância, considerando que falta de conhecimento e falta de confiança são coisas bem diferentes? Uma forma, expressa no Modelo de Opinião de Josang [Josang, 1998], é empregar uma tripla de valores, c , d e i , que correspondem à “crença”, “descrença” e “ignorância”, com $c + d + i = 1$ ($1 \geq c, d, i \geq 0$).

Um dos principais desafios na área de P2P é o projeto de um sistema de reputação que consiga determinar, precisa e eficientemente, valores de confiança adequados para nodos de um sistema descentralizado de larga escala, e em que nodos entram e saem autonomamente e a todo o momento do sistema, potencialmente sob identidades diferentes, e podendo forjar mensagens e identidades. A confiança de um nodo em outro é baseada, naturalmente, na **visão** do nodo sobre a reputação do outro. Nodos adquirem boa reputação através de interações bem sucedidas, que recebem uma avaliação positiva por parte dos nodos envolvidos; por essa razão, esses sistemas são ditos baseados em *feedback* (retroalimentação).

Muitos sistemas de gerência de reputação foram propostos buscando resolver esse desafio. A literatura é particularmente rica em propostas; para ilustrar, existem mais de cinquenta artigos publicados sobre confiança e reputação em P2P desde 2001, muitos com propostas similares ou parcialmente sobrepostas. Segundo [Marti and Garcia-Molina, 2006], sistemas de reputação têm em comum três partes principais: coleta de informações, determinação de escore e ranqueamento, e ações de resposta.

A coleta de informações está relacionada ao esquema de identificação usado, às fontes de informação, à agregação de informações e à política adotada para nodos que ingressam no sistema sem histórico associado (tais nodos são ditos *estranhos*). No extremo de precaução, um nodo confia como fonte de informação apenas em suas informações locais. Um usuário cauteloso pode aumentar as fontes de informação perguntando a opinião de outros em que

confia com base em uma relação externa prévia, ou um nodo pode perguntar a outros nodos (vizinhos ou nodos com quem já interagiu com sucesso) sobre um determinado nodo. Se insuficiente, um nodo A pode solicitar a um nodo B (em que confia), que pergunte aos nodos em que B confia, sobre um nodo C, de forma recursiva e transitiva. Aqui há um *trade-off* entre segurança e desempenho: aumentar o número de nodos perguntados (buscando qualificar as informações sobre outros nodos) pode afetar negativamente o desempenho do sistema.

O *survey* sobre confiança em [Marti and Garcia-Molina, 2006] discute fontes de informação que um nodo pode buscar sobre um outro nodo, bem como as estratégias que um nodo pode usar. As seguintes fontes são elencadas: relação de confiança *a priori* com outro nodo; fontes externas que sejam de confiança; nodos que sejam de confiança e que estejam a um nodo de distância; nodos que sejam de confiança e que estejam a vários nodos de distância; e um sistema global de reputação.

As informações obtidas podem ser aplicadas por um nodo com as seguintes estratégias de confiança: otimisticamente assumir que todos os estranhos são de confiança até prova em contrário; pessimisticamente ignorar todos estranhos até que sejam provados de confiança; investigar um estranho perguntando a nodos de confiança; transitivamente propagar a investigação através de amigos de amigos; ou usar um sistema de reputação centralizado.

Um dos problemas fundamentais de sistemas de reputação é garantir a validade das informações prestadas por outros nodos. Portanto, é natural que na determinação do escore de reputação de um estranho, experiências anteriores do próprio nodo sejam valorizadas em relação à opinião de outros nodos. Uma abordagem comum nesse sentido é a aplicação de pesos: a reputação de uma informação dada por um nodo é proporcional à reputação desse nodo. Informações coletadas através de confiança transitiva podem ser pesadas de acordo com a reputação do nodo de menor reputação na cadeia de confiança; alternativamente, se os valores de confiança situam-se em $[0, 1]$, então o valor resultaria da multiplicação dos escores de reputação de cada um dos nodos.

Sobre gerência dos escores de reputação, [Gupta et al., 2003] apresenta duas abordagens possíveis para mapear ações de nodos em um escore não-negativo de reputação: uma é baseada em crédito e débito (para nodos que prestam ou consomem um conteúdo/serviço, respectivamente), e a outra apenas em crédito, mas onde créditos expiram naturalmente com o passar do tempo.

Em termos de ação de resposta, um sistema de gerência de reputação fornece como serviço um valor de reputação sobre outros nodos, que pode ser usado então em diversos contextos em um sistema P2P. Por exemplo, ao buscar um nodo que execute de maneira confiável um determinado serviço, ele pode usar a informação de reputação para inferir qual a probabilidade de o serviço ser executado corretamente. Outro exemplo é, ao buscar um objeto (como um arquivo) e encontrar múltiplos nodos como candidatos a fonte, usar

o valor de reputação (aliado ao desempenho) na escolha do nodo ou nodos a contactar.

Segundo [Feldman and Chuang, 2005], existem duas questões principais que têm sido tratadas em diversos trabalhos na literatura:

- como nodos novos, ditos estranhos, devem ser tratados em esquemas de reciprocidade (ou seja, em esquemas de reputação)?
- estratégias que são baseadas em reciprocidade indireta são vulneráveis a comportamento de conluio?

Essas questões são importantes em termos de vulnerabilidades do sistema de gerência de reputação. Existem diferentes formas de ataque a um sistema de reputação; os três principais ataques são discutidos a seguir.

O primeiro ataque é conhecido como *whitewashing* e apenas ocorre quando nodos podem trocar sua identidade facilmente (o que é o caso de muitos sistemas P2P). Um nodo pode deixar o sistema e voltar em seguida com uma nova identidade, em uma tentativa de se livrar de qualquer reputação ruim que ele tenha acumulado. Se a política dos nodos em relação a estranhos é permissiva, nodos podem “usar” a reputação inicial, deixar o sistema e reingressar com nova reputação inicial. Se um nodo não consegue distinguir um nodo novo correto de um antigo, então *whitewashers* podem causar o colapso do sistema se nenhuma contramedida for tomada [Feldman et al., 2004].

O segundo tipo de ataque é o de conluio contra sistemas de reputação. Este tipo de ataque é frequentemente efetivo porque em sistemas de reputação típicos um nodo deve consultar outros nodos sobre a reputação de um terceiro. Se muitos nodos estão comprometidos, então nodos podem prover falso testemunho, no sentido de aumentar a reputação de um nodo malicioso, ou de atacar um nodo correto diminuindo a sua reputação. Em princípio, o atacante deveria dispor de recursos em massa, fazendo com que boa parte dos nodos do overlay fossem seus; entretanto, em muitos sistemas P2P não existe um esquema seguro de autenticação, possibilitando que nodos adquiram múltiplas identidades falsas (criando nodos Sybil) com um único nodo físico, conforme explorado em [Cheng and Friedman, 2005].

O terceiro tipo de ataque é o de nodo *traidor* [Marti and Garcia-Molina, 2006]. Em tal ataque, um nodo se comporta adequadamente por um tempo de forma a construir uma boa reputação, e então explora o sistema valendo-se da mesma. Este ataque é especialmente efetivo quando os nodos ganham privilégios a medida que conquistam reputação. Um exemplo do ataque de traidor é quando um usuário no eBay [eBay, 2006] constrói uma reputação com muitas transações de pequeno valor, e então lesa alguém em uma transação de grande valor. Em termos sistêmicos, um nodo traidor pode surgir não de uma mudança comportamental de um usuário, mas de uma mudança no ambiente: por exemplo, uma máquina cliente perfeitamente correta pode ser infectada com um vírus estilo Cavalo de Tróia, que então poderia aleatoriamente abusar da boa reputação do nodo. A

resistência a esse tipo de ataque pode ser aumentada usando a análise da história recente de um nodo [Feldman et al., 2004].

Por fim, conforme anteriormente comentado, a reputação de um nodo é tipicamente usada na política de seleção de um nodo com quem interagir. Em [Dumitriu et al., 2005] é caracterizado o impacto de políticas de seleção entre respostas adotadas por nodos P2P que originam buscas. São identificadas diversas políticas, dentre as quais incluem-se escolher o nodo que anuncia a melhor capacidade, ou seja, com o menor atraso, que é calculado por cada nodo em função da capacidade de *upload* e o número atual de *uploads* simultâneos; aleatório, onde o cliente seleciona um nodo aleatoriamente; e *file chunking*: o cliente divide o arquivo em múltiplos blocos, e faz *download* simultâneo de um pedaço de cada nodo que anunciou o arquivo.

Sob ponto de vista de ataques, a pior política de escolha de um nodo é aquela que seleciona o nodo que se anuncia como o melhor nodo. Por exemplo, se uma informação de desempenho é facilmente falsificável, uma busca só terá sucesso quando **nenhuma** resposta for de um nodo malicioso, pois um nodo malicioso sempre será escolhido. Segundo [Dumitriu et al., 2005], sistemas de reputação **não** conseguem resolver esse problema mesmo quando os erros do sistema de reputação são mínimos. Técnicas baseadas em aleatoriedade são efetivas para aumentar a resistência de um sistema P2P a ataques. Entretanto, aleatoriedade impacta negativamente no desempenho quando atacantes não estão presentes.

4.3.6. Autorização

Para que aplicações P2P possam ser empregadas em ambientes além do popular compartilhamento de arquivos, torna-se indispensável integrar a elas mecanismos de *autorização*. No contexto P2P, o conceito de *autorização* está ligado ao provimento de mecanismos de controle de acesso entre nodos. Boa parte das aplicações existentes concentra-se em compartilhar recursos abertamente ao invés de definir regras que permitam definir *quem, quando* e *o que* pode ser acessado.

A ainda baixa popularidade de mecanismos de controle de acesso em aplicações P2P pode ser explicada, em parte, pelos requisitos a serem satisfeitos por potenciais soluções. O primeiro consiste em definir uma solução que **não** comprometa a escalabilidade da aplicação; a maioria das propostas para controle de acesso em ambientes mais tradicionais faz uso de serviços centralizados para prover esse tipo de funcionalidade. O segundo requisito consiste em lidar, satisfatoriamente, com o anonimato característico das redes P2P. Ao contrário de sistemas cliente/servidor, em que há um acoplamento forte entre os envolvidos na comunicação, nodos em um sistema P2P típico são fracamente acoplados e fornecem pouca informação sobre sua identidade. O terceiro requisito está relacionado a encontrar uma forma de manter o incentivo ao compartilhamento, apesar das restrições impostas pelas políticas de acesso associadas ao mecanismo de *autorização*.

Ciente desses desafios, em [Tran et al., 2005] é proposto um *framework* para controle de acesso em aplicações de compartilhamento de arquivos P2P. A solução mescla aspectos de modelos de reputação e recomendação com esquemas de justiça (*fairness*) e controle de acesso. O nodo é considerado como um sistema *standalone* em que os arquivos compartilhados são tratados como objetos que precisam ser protegidos, e os nodos que solicitam tais objetos são sujeitos que possuem ou precisam ganhar direitos de acesso. Os arquivos disponíveis são categorizados de acordo com seu tamanho e conteúdo, e têm associado a si dois limiares, que determinam dois aspectos relacionados ao seu acesso. Um nodo que solicita a recuperação de um objeto precisa ter valores equivalentes ou superiores aos limiares associados a esse objeto. Esses valores são computados com base em quatro escores: *direct trust*, *indirect trust*, *direct contribution* e *indirect contribution*. Cabe ao próprio nodo requisitante coletar – junto a outros nodos – recomendações que o habilitem a calcular seus valores perante um determinado nodo. Após o término de cada transação, os valores de *direct trust* e *direct contribution* são atualizados de acordo com o grau de satisfação da transação. Esses novos valores afetarão as avaliações de controle de acesso em futuras comunicações entre esses dois nodos.

Já [Park and Hwang, 2003] propõe uma abordagem, baseada no modelo RBAC (*Role-based Access Control*), para controle de acesso em aplicações P2P de colaboração. Os autores definem um *middleware*, transparente para a aplicação, que atua como um *broker* na comunicação. Ao contrário do trabalho recém descrito, uma entidade central (o servidor de políticas) é utilizada para armazenar políticas de acesso aos recursos (em nome dos nodos). As políticas são transferidas periodicamente desses servidores para os nodos, permitindo que a decisões sejam tomadas autonomamente por eles.

4.3.7. Integridade

As aplicações mais comuns de redes P2P, como já mencionado anteriormente, são as de armazenamento e compartilhamento de arquivos. Nessas, um dos aspectos de segurança mais demandados é a *integridade*, indispensável para a identificação de objetos corrompidos. A adulteração de objetos pode ocorrer no momento de seu armazenamento ou no trânsito entre nodos.

Diversos mecanismos baseados em *criptografia* vêm sendo empregados para prover integridade a aplicações P2P. Em [Dabek et al., 2001b] é proposto um mecanismo que permite que a integridade de um objeto possa ser verificada pelo nodo que o recuperou. Para tal, o nodo que deseja inserir um objeto na rede calcula o *hash* de seu conteúdo (usando uma função conhecida) para produzir a chave. Quando um nodo recupera um objeto, ele calcula o *hash* do objeto e o compara com a chave usada na busca. Em sendo iguais, o objeto pode ser considerado íntegro.

Outro mecanismo adotado por certas aplicações (por exemplo o Free Haven) é o algoritmo para dispersão de informação [Rabin, 1989]. Um objeto a ser armazenado é dividido em m blocos, de modo que quaisquer n deles são sufi-

cientes para reconstituir o objeto original (com $m < n$). Dependendo do grau de redundância empregado na composição dos blocos, o mecanismo será mais ou menos resistente (*resilient*) a perdas. Um esquema semelhante, proposto em [Shamir, 1979], é o de compartilhamento seguro. Nesse esquema, o nodo que publica cifra o objeto com uma chave K . Em seguida, divide K em l *shares*, de modo que quaisquer k desses *shares* permitam reconstruir a chave, mas $k - 1$ não oferecem pistas suficientes sobre K . Cada nodo, então, cifra um dos *shares* da chave junto com o bloco do objeto. Para que um objeto fique indisponível, pelo menos $l - k - 1$ nodos que armazenem sua chave devem ser “derrubados”.

Já que a violação dos mecanismos baseados em criptografia é bastante custosa computacionalmente, os ataques à integridade mais típicos são os seguintes:

- ataque de poluição a arquivos (*file-targeted DoS attack*): um nodo malicioso anuncia uma cópia corrompida de um arquivo e, então, a distribui quando solicitado por outro nodo. Evidências indicam que isso está sendo realizado pela indústria fonográfica, e existem companhias que vendem esse serviço de ataque publicamente na Internet, como por exemplo overpeer (<http://www.overpeer.com/>);
- ataque de resposta falsa (*false attack reply*): um nodo malicioso encaminha normalmente mensagens de busca; entretanto, identifica e intercepta mensagens de resposta (para qualquer arquivo), e modifica a resposta indicando ele mesmo como detentor de uma cópia e com baixíssimo atraso; se selecionado pelo requisitor, o nodo malicioso fornece uma cópia corrompida do arquivo.

Um estudo em [Dumitriu et al., 2005] faz uma análise sobre a resistência de sistemas de compartilhamento de arquivos P2P contra esses tipos de ataque. Ele mostra ainda que em um ambiente cooperativo um ataque de poluição de arquivos não consegue prevenir o espalhamento de cópias corretas de um arquivo. Sem cooperação entre os nodos, entretanto, fatores como (a) remoção lenta ou incompleta de cópias corrompidas, (b) pouca inclinação ao compartilhamento de arquivos obtidos e (c) falta de persistência ao recuperar arquivos quando o sistema se encontra sob ataque evitam que cópias corretas se espalhem pelo sistema.

4.3.8. Anonimidade e Negabilidade

Anonimidade constitui um dos aspectos de segurança mais almejados em aplicações P2P típicas, tais como as de compartilhamento de arquivos e as de armazenamento distribuído, que desejam oferecer a seus usuários privacidade, confidencialidade e resistência à censura. No contexto dessas aplicações, anonimidade consiste em não permitir a identificação: (a) do autor ou aquele que publica o objeto; (b) da identidade de um nodo que está armazenando determinado objeto; (c) da identidade e do conteúdo do objeto; e (d) dos detalhes de uma requisição para recuperar determinado objeto [Dingledine et al., 2001].

A busca por soluções para prover anonimidade em aplicações P2P tem sido motivada, em parte, como uma reação à censura a que as primeiras aplicações de compartilhamento de arquivos como Napster foram alvo há alguns anos [Danezis and Anderson, 2005]. A arquitetura centralizada dessas primeiras aplicações possibilitou que seus proprietários fossem responsabilizados legalmente pelo conteúdo armazenado. Além disso, foram forçados a revelar a identidade de usuários e suprimir determinados tipos de conteúdo do sistema.

Diversas soluções têm sido propostas para prover anonimidade a aplicações Internet, incluindo as baseadas em P2P [Reed et al., 1998, Freedman and Morris, 2002, Dingleline et al., 2004, Clarke et al., 2001, Dingleline et al., 2001]. Na técnica básica empregada pela maioria dessas soluções, dois nodos que desejem se comunicar o fazem através de um conjunto de *relays*, que transmitem as mensagens entre si até que um deles, definido arbitrariamente para cada conexão, encaminha a mensagem ao destinatário. Essa técnica – conhecida como redes *mix* de Chaum – foi apresentada pela primeira vez em [Chaum, 1981]. Detalhes sobre as contribuições oferecidas pelas propostas recém referidas serão apresentadas na Seção 4.4.

Além de soluções para manter anônimos os usuários de certas aplicações, outros trabalhos vêm sendo realizados nesta área. Em [Singh and Liu, 2003] é proposto um protocolo para atualizar e recuperar, de modo anônimo e seguro, informações sobre confiança (*trust*). Os autores argumentam que a distribuição e o acesso a essas informações devem ser realizados de maneira mais cuidadosa do que, por exemplo, a requisição e a recuperação de um objeto em uma aplicação P2P. Já Zhuang et al. descrevem em [Cashmere, 2006] um sistema de roteamento anônimo resistente *resilient* a falhas denominado Cashmere. Ao invés de selecionar nodos individuais para atuarem como *relays*, o sistema seleciona regiões do *overlay*. Qualquer nodo presente em uma região pode desempenhar a função de *relay*, reduzindo sobremaneira a probabilidade de falhas no sistema.

Os principais ataques a que soluções de anonimidade estão vulneráveis são os *passivos*, em especial os de escuta e análise de tráfego. O objetivo desses ataques é desvendar a identidade dos elementos envolvidos nas comunicações estabelecidas. No caso particular das aplicações P2P, há interesse – ainda – em revelar quem publicou, quem está armazenando e quem solicitou determinados objetos. Trabalhos como [Murdoch and Danezis, 2005] analisam as fragilidades de soluções frente a essa natureza de ataque.

A *negabilidade* em aplicações P2P pode ser entendida como um componente do aspecto *anonimidade*, referindo-se à habilidade de um nodo em negar conhecer o conteúdo de objetos por ele armazenados [Theotokis and Spinellis, 2004]. Em sistemas com suporte à negabilidade, usuários não podem ser responsabilizados pelo conteúdo armazenado em seus nodos, tampouco pelas ações executadas pelos nodos como parte de sua operação na rede P2P.

Sistemas P2P estruturados possuem dificuldades em prover negabilidade aos nodos, uma vez que as chaves dos objetos são associadas aos endereços dos nodos que os armazenam. Assim, se um determinado objeto está disponível no sistema, é possível determinar sua localização e o nodo que está de posse do mesmo. Por outro lado, o proprietário de um nodo não pode controlar os objetos que vai armazenar e, portanto, não poderia ser responsabilizado.

4.4. Propostas de Soluções

Nesta seção descrevemos um apanhado de propostas para solucionar os principais problemas de segurança identificados em overlays P2P. Destacamos, em um primeiro momento, as propostas de segurança para roteamento seguro e anônimo de mensagens em P2P, nas Seções 4.4.1 e 4.4.2, respectivamente. Na sequência, apresentamos Sistemas de Gerência de Reputação, na Seção 4.4.3. Em seguida, abordamos os aspectos de segurança apresentando dois exemplos de sistemas completos, o Freenet e o Free Haven, nas Seções 4.4.4 e 4.4.5. Escolhemos dois sistemas que envolvem anonimidade por contemplarem um bom leque de requisitos de segurança. Por fim, na Seção 4.4.6 descrevemos P2PSL, um *middleware* – proposto por nosso grupo de pesquisa – para incorporação flexível de aspectos de segurança em aplicações peer-to-peer.

4.4.1. Roteamento Seguro em overlays P2P

Na Seção 4.3.3.2, foram apresentados ataques de roteamento. Nesta seção, discutimos propostas e contra-medidas em relação a esses ataques, tanto para sistemas estruturados como não estruturados.

Em [Daswani and Molina, 2004] é tratado o problema de ataques coordenados aos protocolos de descoberta de recursos em overlays não estruturados (afetando disponibilidade e autenticidade). Ele foca no envenenamento da cache de pong (ou *Pong-cache poisoning*) no protocolo GUESS. Conforme citado anteriormente, a cache pong é a lista de nodos conhecidos ao nodo; esta lista é usada para descoberta de novos nodos no overlay, e seu envenenamento é causado pela contaminação com identificadores de nodos adversários. No *bootstrap*, o nodo ingressante deve preencher sua cache pong; os autores analisam diferentes políticas. É proposto o uso de um protocolo de apresentação de um nodo ingressante a outros nodos, de forma que o mesmo passe a constar de outras caches. Um nodo envia mensagens de ping para outros nodos, que devem responder com uma lista de nodos supostamente vivos. O tamanho dessa lista é pré-definido e os autores avaliam diferentes políticas para escolha de quais nodos enviar na lista, assim como políticas para escolha de quais nodos substituir na cache.

Segundo [Sit and Morris, 2002], ataques de roteamento podem ser combatidos fazendo com que nodos corretos detectem a violação de invariantes no roteamento P2P, como por exemplo o fato de uma busca ficar cada passo mais próxima do nodo destino. Isto é passível de implementação se o nodo origem puder observar o progresso da sua busca. Ataques a atualizações incorretas

de rotas são combatidos verificando as informações de rota. Os erros patentes podem ser facilmente verificados, enquanto atualizações das tabelas devem ser feitas apenas após a verificação do nodo destino. O particionamento é evitado através de um *bootstrap* seguro, como por exemplo com o auxílio de chaves públicas. Um nodo pode usar endereços de nodos acessados com sucesso no passado, embora esta estratégia possa ser minada pelo amplo uso de DHCP para atribuição de IPs dinâmicos a nodos. Os autores sugerem a detecção de partições através de consultas aleatórias a nodos que foram usados com sucesso no passado, comparando com buscas encaminhadas a nodos anteriormente desconhecidos.

O Roteamento Seguro para overlays P2P é descrito em [Castro et al., 2002] como uma primitiva que assegura que quando um nodo não falho envia uma mensagem para uma chave k , com alta probabilidade a mensagem chega a todos nodos não falhos no conjunto de nodos que mantém réplicas. Depende de três mecanismos: atribuição segura de ids de nodos, manutenção segura da tabela de roteamento e encaminhamento seguro de mensagens, conforme descrito a seguir.

O primeiro mecanismo limita a escolha de identificadores via Autoridade de Certificação, evitando que um atacante possa particionar um overlay, isolar um nodo específico, ou vir a controlar determinados objetos em um overlay. A Autoridade atribui e assina certificados de identificadores de nodos, associando um id aleatório a uma chave pública. O certificado e o nodo estão associados ao IP (que se alterado requer novo certificado). A Autoridade assegura que identificadores sejam escolhidos aleatoriamente e não sejam forçados. A atribuição distribuída de identificadores de nodos possui limitações de segurança intrínsecas, mas uma alternativa nessa linha é obrigar um nodo a resolver um *crypto-puzzle* ao receber um id, limitando portanto a taxa com que ids podem ser obtidos.

O segundo mecanismo faz a associação de IPs a ids de nodos para evitar que nodos maliciosos enganem, com sucesso, nodos corretos quanto à proximidade de outros nodos, em algoritmos de roteamento que usam proximidade de rede. Além disso, evita o ataque de atualização ilegítima de rotas através de uma Tabela de Roteamento Restrita, que impõe condições para inserção de ids em entradas da tabela, como CAN e Chord, que precisam ser os ids mais próximos a um ponto no espaço. A proposta de solução contempla duas tabelas, uma com proximidade de rede e a outra restrita, a ser usada quando se detecta que a normal deixou de funcionar. Ataques no *bootstrap* são resolvidos contactando-se um número suficientemente grande de nodos, para aumentar a chance que pelo menos um nodo seja correto.

O terceiro mecanismo que sustenta o Roteamento Seguro de [Castro et al., 2002] é o encaminhamento seguro de mensagens. Qualquer um dos nodos maliciosos que exista em uma rota pode não encaminhar mensagens que sejam enviadas a uma chave. A solução passa pela replicação dos objetos, aumentando a chance de pelo menos uma mensagem chegar

a uma réplica. Falhas no roteamento são detectadas solicitando o conjunto de nodos réplicas ao detentor do objeto, e em caso de falha no teste usar roteamento redundante; a detecção é baseada na análise da densidade de nodos. Com roteamento redundante, mensagens são enviadas à chave k mas encaminhadas por diferentes vizinhos do origem. O roteamento é feito de forma iterativa e com verificação a cada passo, tal como proposto por [Sit and Morris, 2002]: o origem pergunta ao seu vizinho qual o nodo seguinte na rota, e então pergunta ao próximo nodo qual o nodo seguinte, e assim por diante, potencialmente escolhendo um novo nodo em caso de falha (ou de nodo malicioso). A sobrecarga do mecanismo de roteamento seguro proposto é substancial. Dados auto-certificáveis podem ser usados para diminuir o uso de roteamento seguro, pois é mais fácil detectar problemas neste caso.

Em [Srivatsa and Liu, 2004] é mostrada a importância do uso de caminhos alternativos (sub-ótimos) e independentes em overlays com a presença de nodos maliciosos, e o impacto dos mesmos em termos de robustez e desempenho. Mais precisamente, caminhos independentes entre dois nodos são aqueles que não compartilham nenhum nodo intermediário entre eles.

Em um overlay baseado em Chord, mensagens são roteadas de forma unidirecional. Um atacante pode fazer com que mensagens de busca sejam desviadas via nodos maliciosos, aumentando bastante o comprimento de uma busca (em número de nodos), com impacto negativo no desempenho. Em [Xuan et al., 2003] é proposta uma variante do Chord, denominada **RChord**, que acrescenta arestas reversas e permite o roteamento bi-direcional. Isto faz com que RChord seja resistente a ataques de roteamento que desviem mensagens de busca para nodos distantes. No entanto, RChord é limitado, pois é vulnerável a ataques que descartem mensagens ou alterem tabelas de roteamento. O modelo adotado, para permitir a análise de desempenho, é simplista.

Em [Danezis et al., 2005] é proposta uma estratégia de roteamento no Chord cuja idéia básica é fazer com que os nodos virtuais falsos (os Sybils) se tornem *trust bottlenecks*, assegurando que buscas usem um conjunto amplo de nodos. O roteamento iterativo do Chord é modificado de forma a fazer com que um nodo não responda apenas com o id do próximo nodo, mas sim uma série de informações: sua tabela inteira de roteamento, seus nodos sucessores e seu grafo de *bootstrap*. O nodo que origina a busca pode assim escolher qual o próximo nodo em uma busca, baseado nas informações retornadas até o momento. O princípio é nem sempre escolher como próximo nodo um que seja mais próximo ao destino; ao invés disso, nodos podem ser escolhidos levando em conta a quantidade de confiança que tem sido colocada em um nodo. Contra-intuitivamente, os autores propõem que não seja escolhido aquele nodo em que se julga confiar mais, mas sim tentar usar todos os nodos, baseado na premissa que a informação de *trust* pode ser falsa. Os autores definem sua técnica como Roteamento Diverso. Como o Roteamento Diverso puro não faz com que a busca se aproxime do nodo destino, é proposto o uso do Roteamento Zig-Zag como alternativa de roteamento misto. A mesma

consiste em alternar-se entre nodos mais próximos ao destino e nodos menos utilizados até o momento (de forma a dispersar o roteamento).

Em [Condie et al., 2006] é apresentada uma defesa para ataque de roteamento contra sistemas estruturados cuja filosofia básica é permanentemente causar instabilidade no overlay, roubando do atacante a oportunidade para planejar um ataque, se posicionar e amplificar a contaminação de uma tabela. Mais especificamente, a proposta se baseia no trabalho sobre Roteamento Restrito [Castro et al., 2002], em que há duas tabelas, uma otimizada e outra não, e um detector de defeitos para selecionar qual tabela é usada. Condie propõe um *reset* periódico da tabela otimizada com o conteúdo da tabela restrita. Isto só é efetivo se o nível de envenenamento cresce suavemente, o que nem sempre é verdade. Como resposta, os autores então propõem uma limitação na taxa de atualização da tabela otimizada. Se um atacante pode prever quando os *resets* serão feitos, ataques podem ser realizados logo antes de um *reset*. Portanto, o trabalho sugere que após cada *reset*, cada nodo obtenha um novo id aleatório e seja portanto movido na topologia. Nodos fazem *churn* em taxas diferentes, para aumentar a imprevisibilidade e distribuir o tráfego de *churn*. Os autores implementaram uma extensão segura ao Bamboo, cunhada Maelstrom. A demanda de taxa sobre o servidor de aleatoriedade é da ordem de 512kbps para 50.000 nodos. Um aspecto único do sistema é sua proatividade de segurança, uma vez que busca prevenir ataques. Isto é um *trade-off*, pois força os nodos corretos a mudar de identificador, força o descarte de otimizações de cache, etc. mesmo na ausência de ataques. Outras limitações que identificamos na abordagem são: lenta ou não convergência para conectividade ótima; necessidade de um servidor aleatório, que deve permanecer disponível, seguro e alcançável por todos os nodos e com baixa latência. Por fim, a mudança periódica de identificador em nodos impede que o esquema seja empregado em certas aplicações, por exemplo o armazenamento em rede, pois provocaria excessiva migração de dados.

4.4.2. Roteamento Anônimo em overlays P2P

Roteamento Cebola (*Onion Routing*) [Reed et al., 1998] é um sistema que tem por objetivo prover *anonimidade* para uma grande variedade de aplicações, protegendo-as contra escuta (*eavesdropping*) e análise de tráfego. Uma aplicação cliente se conecta a uma determinada aplicação servidora por intermédio de um conjunto de nodos intermediários denominados roteadores cebola (*onion routers*). Essa rede de roteamento em nível de aplicação permite que a conexão entre requisitante e fornecedor permaneça anônima.

A Figura 4.3 ilustra o funcionamento básico do sistema. A aplicação cliente conecta-se a um *proxy* de aplicação, que ajusta as informações dessa e de mensagens subseqüentes para um formato apropriado para transitar pela rede de roteadores cebola. Em seguida, esse *proxy* se conecta ao *proxy* cebola (*onion proxy*), com o objetivo de definir uma rota pela qual as mensagens vão transitar. Essa rota é representada através de uma estrutura de dados em

camadas denominada *cebola*. A cebola é passada, então, para um túnel de entrada.

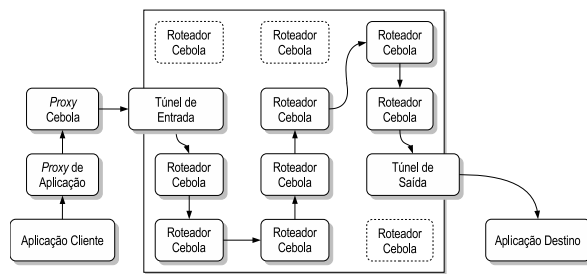


Figura 4.3. Fluxo de mensagens no sistema Roteamento Cebola.

Quando um túnel de entrada ou roteador cebola recebe uma cebola, ele a decifra, revelando uma nova camada da cebola. Essa camada conterá informações sobre o próximo salto na rota estabelecida pelo proxy cebola. A camada é removida e a cebola resultante é encaminhada a esse próximo salto. Mais cedo ou mais tarde, a cebola chega a um túnel de saída. O pacote decifrado nesse ponto – idêntico ao que foi produzido pelo proxy de aplicação no início da conexão – será enviado à aplicação destino.

Para dificultar a análise de tráfego, todas as mensagens que circulam entre os roteadores cebola possuem o mesmo tamanho. Para tal, as mensagens (células) são preenchidas com conteúdo aleatório em cada roteador cebola.

Roteamento Cebola se utiliza de criptografia de chave pública, permitindo que as diferentes camadas da cebola sejam cifradas pelo proxy cebola, de modo que apenas os destinatários de cada camada possam decifrá-las com suas respectivas chaves privadas.

Quando o destinatário envia uma resposta à mensagem recebida, o túnel de saída a converte para o protocolo genérico, a criptografa com sua própria chave privada, e o envia usando o mesmo caminho (inverso) da mensagem inicial. Cada roteador cebola no caminho criptografa a cebola de resposta com sua chave privada. Quando a cebola chega ao proxy cebola, o mesmo a decifrará usando as chaves públicas dos roteadores que fizeram parte da rota, até revelar a mensagem original.

Um conjunto de ataques pode ser realizado com o intuito de comprometer a anonimidade no sistema Roteamento Cebola:

- ataque de marcador (*marker attack*): como as mensagens são transportadas em células de tamanho fixo que mudam seu formato em cada roteador cebola visitado, acompanhar a sua movimentação se restringe a identificar marcadores que indicam quando uma célula começa. Em um ataque de marcador, o atacante identifica o conjunto de células saindo

de um roteador cebola através de um prefixo (ou marca) que não muda ao longo da rota. Ao relacionar informações coletadas em diferentes roteadores cebola, é possível identificar, através desses marcadores, o conjunto de possíveis próximos saltos. Esse ataque é bastante simplificado se houver roteadores cebola comprometidos no sistema.

- ataque de temporização (*timing attack*): o atacante registra uma assinatura baseada no tempo para uma determinada sessão, correlacionando taxa/frequência de envio de dados ao longo do tempo. Essa assinatura muito provavelmente não muda ao longo da rota, permitindo que atacantes (analisando o tráfego em pontos distintos) consigam distinguir determinadas sessões.
- ataque de volume (*volume attack*): roteadores cebola comprometidos podem contabilizar o número total de células que uma determinada conexão anônima possui e encaminhar essa informação a outros roteadores cebola comprometidos. Pela observação de conexões anônimas com volume semelhante de células passando por esses roteadores (em instante próximo), é possível identificar o caminho percorrido pela conexão anônima.

Tarzan [Freedman and Morris, 2002] é um overlay em nível de rede para, a exemplo do sistema Roteamento Cebola, prover *anonimidade* a aplicações na Internet. Por ser uma solução em nível de rede, Tarzan pode ser usado, de forma transparente, por um grande conjunto de aplicações.

A anonimidade é obtida usando o mesmo esquema adotado pelo sistema Roteamento Cebola: criptografia em camadas e roteamento baseado em múltiplos saltos. Como uma evolução àquele sistema, as contribuições introduzidas por Tarzan, segundo os próprios autores, se desdobram em duas. A primeira reside na extensão da solução seguindo um *design* P2P, possibilitando que nodos possam se comunicar por intermédio de *relays* selecionados a partir de um conjunto aberto de nodos voluntários, sem a presença de nenhum componente centralizado. A segunda contribuição consiste na introdução de uma técnica para inserir no *overlay* tráfego artificial com o propósito de evitar que, através de análise, seja possível identificar quem inicia determinadas comunicações. Ainda, para reduzir os riscos de ataques ao overlay (apesar de sacrificar o desempenho) Tarzan implementa uma política na definição de rotas que prevê a seleção de nodos presentes em diferentes organizações e domínios administrativos.

De acordo com [Freedman and Morris, 2002], as novidades introduzidas no Tarzan visam eliminar algumas das fraquezas observadas no sistema Roteamento Cebola. Por empregar um conjunto reduzido e fixo de *relays*, este sistema deixa seus usuários vulneráveis mesmo na presença de apenas um roteador cebola malicioso. Por exemplo, se um roteador cebola comprometido recebe tráfego de um roteador cebola de borda, ele pode identificar que esse roteador de borda é o remetente do tráfego observado. Ademais, os poucos

roteadores cebola que atuam na entrada e na saída da rede (*túneis*) podem realizar análise temporal e determinar remetente e destinatário das mensagens observadas.

A experiência acumulada no projeto Roteamento Cebola revelou uma série de fragilidades no *design* do sistema. Além disso, sua implantação em ambiente de produção foi bastante limitada em função de diversos problemas. **Tor** [Dingledine et al., 2004] surgiu nesse contexto, como uma segunda geração do sistema, buscando solucionar os problemas de projeto e implantação há muito identificados, mas que não haviam sido abordados desde então. Alguns dos avanços introduzidos por Tor são enumerados a seguir.

Na versão original do sistema Roteamento Cebola, como ficava a cargo dos proxies produzir as cebolas usando a chave pública (estática) dos roteadores envolvidos, um único roteador cebola malicioso podia armazenar o tráfego encaminhado através dele e, posteriormente, reproduzi-lo na rede. A motivação, nesse caso, era realizar um ataque de negação de serviço, forçando os roteadores cebola subseqüentes a consumirem desnecessariamente seus recursos para decifrar as cebolas reproduzidas repetidas vezes. No sistema Tor, esta limitação é superada com a utilização de um protocolo para estabelecimento incremental de rotas, em que a aplicação negocia chaves de sessão com cada *relay* escolhido para compor a rota.

Outros avanços do sistema Tor consistem: (a) na multiplexação de múltiplos fluxos TCP em um único circuito, aprimorando a eficiência do sistema e o anonimato de seus usuários; (b) na possibilidade de (através de sinalização) as mensagens enviadas deixarem o circuito no meio da rota, frustrando ataques de temporização e volume; (c) no provimento de um mecanismo para controle de congestionamento fim-a-fim que preserva a anonimidade do sistema; (d) na divulgação de *relays* e seu estado através de *servidores de diretórios*, ao contrário do complexo e custoso esquema de inundação adotado por seu antecessor; (e) na disponibilização de um mecanismo para que cada nodo possa tornar pública sua política em relação a *hosts* e portas a que está disposto a se conectar; (f) na verificação de integridade fim-a-fim das mensagens encaminhadas; e (g) no emprego de um mecanismo baseado em servidores protegidos e *rendezvous points* para prover anonimato à aplicação servidora.

Tor tem sido amplamente utilizado [Tor, 2006], inclusive para anonimizar o uso de aplicações P2P como BitTorrent [Azu, 2006]. Nesse exemplo em particular, é aconselhável que apenas as comunicações com o *tracker* sejam anônimas, uma vez que a rede Tor não está preparada para fazer circular o grande volume de tráfego gerado pela transferência de objetos entre nodos.

4.4.3. Sistemas de Gerência de Reputação

Nesta subseção descrevemos brevemente exemplos representativos de sistemas de gerência de reputação, construindo sobre os conceitos apresentados na Seção 4.3.5.

4.4.3.1. XRep

[Damiani et al., 2002] propõe um esquema para gerência de reputação para Gnutella. Diferentemente de outras abordagens, a reputação de nodos é combinada com a reputação de objetos, aumentando a resistência contra ataques do tipo Sybil. Reputações são cooperativamente gerenciadas através de um algoritmo distribuído de polling de maneira a refletir a visão da comunidade sobre o risco de *download* e uso de um objeto.

O protocolo proposto pelos autores estende o protocolo de busca do Gnutella com passos e mensagens adicionais, facilitando a atribuição, o compartilhamento e a combinação de reputações de nodos e recursos.

O cerne do esquema é o protocolo XRep, uma extensão ao protocolo de busca do Gnutella. No Gnutella, um nodo inicia uma busca enviando mensagens (QUERY) aos seus vizinhos; todos os nodos que encontram o objeto requisitado retornam mensagens de resposta através do mesmo caminho pelo qual vieram. Após receber múltiplas respostas positivas (mensagens QUERYHIT), ele pergunta a outros nodos opiniões sobre estes nodos que ofereceram o objeto buscado.

As reputações são binárias, com (+) ou (-), mas os valores podem ser tanto discretos como contínuos. O protocolo de *polling* é composto de cinco fases:

1. **busca de recursos:** nesta fase, nas mensagens QUERYHIT que são retornadas pelos nodos que detém um ou mais objetos que satisfazem a busca, acrescenta-se um *digest* para cada objeto referenciado na mensagem;
2. **seleção de recurso e voto por *polling*:** o nodo requisitor escolhe o melhor nodo dentre aqueles que parecem satisfazer sua busca (ou seja, que responderam). Para tal, o nodo envia uma mensagem a seus pares contendo uma requisição de voto sobre a reputação dos objetos oferecidos e dos nodos que os oferecem. Estas mensagens são implementadas usando mensagens convencionais QUERY, e contém uma chave pública a ser usada na cifragem da resposta para proteger a integridade e a confidencialidade das respostas. Nodos que recebem a pergunta checam seus *repositórios de experiência* e respondem;
3. **avaliação dos votos:** o nodo descarta mensagens adulteradas (os autores indicam que um nodo faz também um agrupamento e combinação de votos que são oriundos de um mesmo nodo para evitar ataques Sybil, mas não demonstram como isso poderia ser feito a não ser com endereço IP), seleciona um conjunto de votantes e envia uma outra mensagem de poll (TRUEVOTE) direto a cada um, para que respondam confirmando seus votos (este passo obriga nodos atacantes a usarem IPs reais);
4. **verificação de melhor nodo:** o nodo mais confiável é contactado para verificar se ele realmente exporta aquele objeto;

5. **download do objeto:** o nodo contacta outro e solicita o *download* do objeto, após o qual ele verifica a integridade do objeto através do *digest* e atualiza seu repositório de experiências.

4.4.3.2. EigenTrust

EigenTrust [Kamvar et al., 2003] é um algoritmo para gerência de reputação para sistemas de compartilhamento de arquivos. Cada nodo possui associada uma reputação global, que é baseada no histórico de uploads de arquivos. A reputação global de um nodo i é calculada com base nos índices de reputação atribuídos localmente a i por cada nodo j, k, l , etc. e pesados de acordo com a própria reputação daqueles nodos. No estudo realizado, a abordagem ajudou a reduzir o número de arquivos espúrios publicados.

Os valores de confiança atribuídos por um nodo são normalizados. Isso é necessário para evitar que um nodo subverta o sistema atribuindo reputações arbitrariamente altas a outros nodos maliciosos, influenciando o valor de reputação global em um ataque de conluio. Em um nodo i , a reputação do nodo j é normalizada dividindo-se o valor de reputação de j em i pelo somatório de todos os valores de reputação que i mantém. Ou seja, todos os valores de reputação atribuídos por um nodo situam-se entre 0 e 1. As desvantagens dessa normalização são duas. Primeiro, a não distinção entre ignorância e má reputação. Segundo, os valores são relativos, e portanto não podem ser interpretados de forma absoluta; por exemplo, se em i dois nodos j e k possuem o mesmo valor de reputação r , então sabe-se que aos olhos de i , j e k são igualmente reputáveis, mas não se sabe se ambos tem boa ou má reputação.

Para agregar os valores normalizados computados por cada nodo, um nodo i pergunta aos seus nodos “amigos” j sobre os valores de reputação que eles atribuíram a um nodo k , e usa uma média ponderada com as reputações deles para calcular a confiança de i em k . Para aumentar o conhecimento, um nodo pode pedir a opinião dos amigos dos amigos, e assim por diante, recursivamente, até atingir a rede inteira.

Não é possível permitir, naturalmente, que cada nodo seja responsável por calcular e reportar sua própria reputação. Portanto, a reputação de um nodo é calculada por mais de um nodo na rede, e armazenada em outro nodo. Múltiplos nodos computam o score de um nodo, e uma DHT é usada para encontrar tais nodos. O algoritmo proposto evita que um nodo saiba a identidade do nodo para o qual ele está calculando a confiança, de forma que um nodo malicioso não possa aumentar artificialmente a reputação de outro nodo malicioso. Nodos que entram no sistema não podem escolher qual a posição em que entram no espaço de ids, evitando que um nodo re-entrasse exatamente na posição do nodo responsável por calcular a sua reputação.

Valores globais de reputação podem ser usados então para isolamento de nodos maliciosos. Um nodo usa a reputação dos candidatos como política de seleção de quem irá baixar um arquivo. Entretanto, uma escolha desse

tipo concentra as requisições nos nodos com reputação mais alta e não permite que outros nodos corretos adquiram reputação. A proposta é usar um esquema onde o nodo é selecionado de forma semi-aleatória, com influência da reputação.

Segundo [Marti and Garcia-Molina, 2006], EigenTrust oferece uma solução puramente descentralizada, mas usa uma identidade fraca, tornando suscetível a ataques de *whitewashing*. Além disso, [Cheng and Friedman, 2005] indica que EigenTrust é suscetível ao ataque Sybil, uma vez que um nodo malicioso pode criar uma subporção inteira do grafo.

4.4.3.3. PeerTrust

PeerTrust [Xiong and Liu, 2004] é um framework de reputação que inclui um modelo de confiança adaptativo para quantificar e comparar a confiança de nodos baseado em um sistema de transações com *feedback*, e uma implementação descentralizada de tal modelo em uma rede P2P. As duas características principais de PeerTrust são a definição de três parâmetros básicos de confiança e dois fatores adaptativos na computação do grau de confiança em um nodo, e a definição de uma métrica geral de confiança para combinar esses parâmetros. Os fatores que um nodo leva em consideração no cálculo de nível de confiança no PeerTrust são:

- *feedback* recebido de outros nodos, na forma de um valor;
- escopo do *feedback*, tal como o número de transações que um nodo possui com outro;
- fator de credibilidade para a nodo que forneceu o *feedback*, diferenciando a qualidade do *feedback* recebido de outros nodos de acordo com a confiança nos mesmos;
- fator de contexto de transação para diferenciar as mais importantes das menos importantes, associando pesos a transações, como por exemplo levando em conta o valor de uma transação;
- fator de contexto da comunidade para tratar características relacionadas à comunidade e vulnerabilidades peculiares, como por exemplo criar um incentivo à submissão de *feedback* sobre outros nodos.

Para implementação desse modelo de confiança, cada nodo possui um gerente de confiança e um localizador de dados. O primeiro é responsável pela submissão de *feedback* e avaliação de confiança através de um banco de dados com um segmento da base global. Já o localizador de dados serve para alocação e localização de dados de confiança no overlay. O gerente executa duas funções principais: (1) submete *feedback* para o overlay através do localizador, que roteia a informação para outros nodos; (2) avalia o nível de confiança de um determinado nodo, o que é executado em dois passos: primeiro, ele coleta informações de confiança sobre o nodo em questão através do localizador, e então computa o valor de confiança.

Dois métodos são propostos para cálculo do nível de confiança: cálculo dinâmico de confiança (DTM, *dynamic trust computation*), que emprega informações “frescas” obtidas sob demanda de todos os outros nodos da rede; e computação aproximada de confiança (ATC, *approximate trust computation*), que é mais eficiente porém menos precisa, ao calcular a confiança com informações presentes em uma *cache*. Cada nodo mantém uma cache contendo os valores de confiança fornecidos por outros nodos que ele recentemente se comunicou; ele só precisa de comunicação quando não acha um nodo na *cache*.

O modelo de confiança usa transações recentes para cálculo da confiança para evitar o problema do traidor (vide Seção 4.3.5). Quando a reputação de um nodo é baseada na média cumulativa de suas transações de seu tempo de vida, e um nodo adquiriu uma sólida reputação, as transações do tempo presente tem pouco impacto na reputação, e portanto um nodo possui menor incentivo para se comportar de forma honesta. Um nodo pode, mesmo assim, oscilar entre um comportamento honesto e desonesto de forma a manter uma reputação razoável apesar de agir incorretamente em determinadas transações. Os autores propõem um algoritmo simples de janela de tempo deslizando. Os valores de confiança são computados de forma global e recente e comparados. A idéia é que uma boa reputação seja difícil de se ganhar, leve tempo para construir, porém possa ser destruída rapidamente após poucas transações incorretas.

Para evitar problemas de segurança relativos ao armazenamento e transmissão das informações de confiança, o PeerTrust emprega criptografia com chaves públicas/privadas. Cada nodo é obrigado a ter um par de chaves e assinar suas mensagens de *feedback* com sua chave privada, e fornecer a chave pública, garantido a integridade e autenticidade. O id de cada nodo é um *digest* de sua chave pública, ou então sua chave pública. Para lidar com ataques de roteamento, [Xiong and Liu, 2004] sugere o uso de replicação, mas não detalha sua proposta.

4.4.3.4. TrustGuard

Os autores em [Srivatsa et al., 2005] discutem três ataques a sistemas de reputação e como os mesmos podem ser contra-atacados com TrustGuard. O primeiro é o ataque de traidor, em que um nodo acumula boa reputação e depois modifica seu comportamento. Outro ataque é o de *shilling*, onde nodos fornecem *feedback* falso e fazem conluio para aumentar sua própria reputação. O terceiro consiste em inundar o sistema com múltiplos *feedbacks* falsos sobre transações inexistentes. Nesse sentido, as contribuições do TrustGuard são:

- introduzir um modelo de confiança que lida eficientemente com oscilações estratégicas no comportamento de nodos maliciosos;
- proposta de um controle de admissão baseado em *feedback* para assegurar que apenas transações com provas seguras sejam contabilizadas

em termos de reputação;

- proposta de algoritmos de credibilidade de *feedback* para efetivamente filtrar *feedbacks* desonestos.

A arquitetura do TrustGuard, em cada nodo, é composta por três componentes principais: uma Máquina de Avaliação de Confiança (*Trust Evaluation Engine*), o Gerente de Transação (*Transaction Manager*) e o Serviço de Armazenamento de Informações de Confiabilidade (*Trust Data Storage Service*).

Antes de um nodo i estabelecer uma transação com um nodo j , ele solicita à Máquina de Avaliação de Confiança que avalie j . A máquina usa um overlay DHT subjacente para contactar outros nodos, coletar *feedback* e agregar o mesmo em um valor de confiança.

O segundo componente, Gerente de Transação, toma como entrada valores produzidos pela máquina e toma decisões sobre confiança. Antes de executar uma transação, o Gerente gera e troca provas da transação; uma vez que a transação é completada, o *feedback* é entrado pelos dois nodos envolvidos. Mensagens com *feedback* são roteadas através do overlay DHT para nodos designados responsáveis pelo armazenamento destes valores.

Os nodos designados invocam então o terceiro componente, o Serviço de Armazenamento, que admitem um *feedback* apenas se o mesmo passa por um teste de controle de admissão para detecção de transações falsas. O Serviço de Armazenamento de informações de confiança é construído sobre o Peer-Trust, discutido na Seção 4.4.3.3.

Os problemas e soluções tratados pelo TrustGuard são os seguintes:

- oscilação estratégica ou problema dos traidores: a solução proposta é incorporar na determinação da confiança em um nodo oscilações no seu comportamento, além do histórico de reputação. Se um nodo oscila sua reputação, então isso afeta negativamente sua reputação;
- detecção de transações falsas: para evitar que um nodo submeta *feedback* de transações que nunca ocorreram, ou positivo de si próprio ou outro nodo malicioso, ou incorretamente negativo sobre um outro nodo a ser atacado, é proposto que os nodos troquem *provas de transação* de maneira a um nodo poder demonstrar que realmente fez uma transação com outro. Isso evita que um nodo invente *feedback* sobre nodos com que não interagiu, mas não evita que um nodo submeta *feedback* incorreto sobre um nodo com quem realizou uma transação.
- *feedback* desonesto: para lidar com o problema citado no item anterior, que é mais grave nas situações de conluio, TrustGuard aplica pesos sobre os valores reportados de acordo com a confiabilidade do nodo que reporta, tal como EigenTrust e outros.

4.4.3.5. FuzzyTrust

[Song et al., 2005] descreve um sistema de gerência de reputação para sistemas P2P de comércio eletrônico. FuzzyTrust usa lógica nebulosa (*fuzzy*) para computar escores locais de confiança e fazer a agregação de escores globais. O sistema usa uma DHT para troca de informações sobre reputação (assim como no caso do TrustGuard).

O FuzzyTrust foi projetado com base em uma extensiva análise executada sobre transações efetuadas no eBay [eBay, 2006]. O padrão segue uma Lei de Potência: há poucas transações de alto valor, porém muitas transações com pequeno valor. O estudo sobre o eBay deu origem a três princípios de projeto:

- o consumo de largura de banda pode ser bastante alto para troca de reputação para os *hotspots* (nodos que se envolvem na maioria das transações) e, portanto, deve considerar o desbalanceamento de transações entre nodos;
- para lidar com o impacto menor de certos usuários, o sistema não deveria aplicar o mesmo ciclo de avaliação a todos os usuários, de forma que os usuários mais freqüentes deveriam ser avaliados mais freqüentemente;
- porque algumas transações concentram a maior parte do valor, faz sentido avaliar as transações maiores mais freqüentemente do que as pequenas.

FuzzyTrust realiza cálculo de reputação localmente e globalmente. Localmente, nodos empregam o mecanismo de inferência *fuzzy* para capturar incertezas e se auto-ajustar à variação de parâmetros locais. A agregação de escores de reputação coletados de todos os nodos é feita para se gerar um escore global para cada nodo. Três pesos de agregação são usados como parâmetros: a reputação de um nodo, a data de transação, e o valor da transação. Como base, cinco regras *fuzzy* são usadas no trabalho descrito (os autores comentam que um número bem maior poderia ser usado em um sistema de maior porte):

1. se o valor da transação é bastante alto e o tempo da transação é recente, então o peso na agregação é bastante alto;
2. se o valor da transação é bastante baixo ou o tempo da transação é bastante antigo, então o peso na agregação é bastante baixo;
3. se a reputação de um nodo é boa e o valor da transação é alto, então o peso na agregação é bastante alto;
4. se a reputação de um nodo é boa e o valor da transação é baixo, então o peso da agregação é médio;
5. se a reputação de um nodo é ruim, então o peso na agregação é bastante pequeno.

São apresentados resultados de uma comparação por simulação entre o FuzzyTrust e EigenTrust usando três métricas: tempo de convergência necessário para estabelecer a reputação global de cada nodo; a taxa de detecção de nodos maliciosos; e a sobrecarga de mensagens envolvida na agregação de reputação global, apresentada tanto individualmente para cada nodo como globalmente. FuzzyTrust e EigenTrust apresentam tempos similares para convergência de reputação global. Na média, com FuzzyTrust há menos mensagens por nodo, e no global também. FuzzyTrust tem uma boa taxa de detecção de nodos maliciosos, situada entre 80% e 98%.

4.4.3.6. Grupos de Confiança

[Gupta and Somani, 2004] propõe um framework para gerência de reputação em sistemas P2P de larga escala onde se assume que todos os nodos são egoístas, usando uma “moeda virtual” para medição de reputação. A reputação dos nodos é decrementada com a passagem do tempo, de forma que nodos precisam continuar colaborando e prestando serviços. A contribuição principal do artigo é o uso de grupos de confiança mútua que atuam de forma conjunta em relação à reputação. Nodos formam comunidades que exibem confiança mútua e cooperam para combater o egoísmo e o comportamento malicioso de outros nodos.

Nodos com maior reputação devem ter acesso facilitado a serviços. Quando dois nodos competem por um serviço, o nodo provedor deve escolher o nodo que tem maior reputação. Para obter-se isso, o sistema faz com que o nodo que serve outro ganhe maior reputação quando o nodo servido é o de maior reputação.

Nodos formam grupos de confiança, em que cada membro confia nos demais e usa esse conhecimento para se defender. A reputação de um nodo é a reputação de seu grupo, que é determinada pela reputação média dos nodos no grupo. A reputação de um nodo só aumenta quando ele presta um serviço a um nodo fora do grupo. A divisão em grupos oferece maior segurança (as informações são mais confiáveis) e escalabilidade.

São tratados os seguintes modelos de ataque:

- um nodo sempre se recusa a cooperar;
- um nodo primeiro coopera e obtém reputação, e depois passa a não cooperar (traidor);
- nodos maliciosos fazem um ataque em conluio objetivando diminuir a reputação de determinados nodos corretos e provocar a expulsão dos mesmos do grupo;
- nodos enviam certificados falsos de satisfação para aumentar a reputação entre si, e oferecem serviço incorreto ou ruim quando solicitados; com reputação alta, podem prevenir a execução de certas tarefas corretas.

4.4.4. Freenet

Freenet [Clarke et al., 2001] é um sistema de armazenamento distribuído, que foi concebido com o objetivo de oferecer: (a) privacidade para nodos que publicam, recuperam e armazenam objetos; (b) resistência à censura; (c) altas disponibilidade e confiabilidade; (d) armazenamento e roteamento eficientes, escaláveis e adaptativos.

Freenet é implementado como uma rede P2P adaptativa, em que os nodos realizam requisições uns aos outros para armazenar e recuperar objetos. Esses objetos são identificados por chaves independentes de localização. Cada nodo mantém seu espaço próprio de armazenamento e o torna disponível à rede para leitura e escrita. Os nodos mantêm, ainda, uma tabela de roteamento dinâmica contendo endereços de outros nodos e as chaves de objetos que eles armazenam.

As chaves no Freenet são calculadas usando operações *hash* SHA-1. Dois tipos de chaves são admitidos:

- *content-hash key*: gerada a partir do *hash* do objeto a ser armazenado, é útil para verificar a integridade do objeto;
- *signed-subspace key*: gerada a partir de operações *hash* e XOR de (a) um texto descritivo do objeto a ser armazenado e (b) a chave pública associada ao espaço de nomes definido pelo usuário que deseja inserir o objeto no sistema.

Recuperação e inserção de objetos. Quando um nodo recebe uma requisição, ele verifica localmente e, se encontra o objeto, o retorna com uma *tag* identificando-se como o detentor do mesmo. Caso contrário, o nodo encaminha a requisição para o nodo, listado em sua tabela, que armazene a chave mais próxima da que foi solicitada. Esse processo se repete até que a requisição chegue ao nodo que possui o objeto. Nesse momento, o objeto é repassado pelo mesmo caminho pelo qual a requisição transitou, fazendo com que cada nodo intermediário atualize sua tabela de roteamento associando o detentor do objeto com a respectiva chave. Para prover anonimato do nodo que está armazenando o objeto, cada nodo ao longo da rota pode decidir alterar a mensagem de retorno, informando que ele próprio ou outro qualquer é a fonte do objeto.

Já para inserir um objeto no sistema, o nodo solicitante atribui ao objeto uma chave e envia uma mensagem INSERT para si próprio. A mensagem possui a chave e um valor de *hops-to-live* que indica o número de cópias do objeto a armazenar. O processo para definir o local onde o objeto será armazenado é semelhante ao de uma busca; a mensagem INSERT percorre o mesmo caminho que uma requisição pela mesma chave percorreria.

Se o valor de *hops-to-live* atinge 0 e nenhuma colisão é detectada, uma mensagem ALL CLEAR é enviada ao nodo que solicitou a inserção. Esse nodo, então, envia o objeto, que vai sendo armazenado pelos nodos ao longo da mesma rota por onde a mensagem INSERT passou previamente. Para fornecer *anonimato* ao nodo que está publicando o objeto, cada nodo ao longo da rota

pode decidir alterar a mensagem de inserção, informando que ele próprio ou outro qualquer é a fonte do objeto.

Por outro lado, se a chave já estiver sendo usada por outro objeto, o nodo retorna o objeto pre-existente como se o nodo que solicitou a inserção tivesse feito uma solicitação pelo mesmo. Desse modo, tentativas maliciosas de substituir objetos legítimos existentes por *lixo* resultarão na disseminação ainda maior dos arquivos legítimos já armazenados.

Para reduzir a quantidade de informações que um nodo malicioso pode obter acessando o valor de *hops-to-live*, as mensagens não deixam de seguir adiante quando *hops-to-live* atinge 1, sendo encaminhadas com uma determinada probabilidade (com *hops-to-live* valendo sempre 1).

Comunicações anônimas. A privacidade no sistema é obtida usando um esquema semelhante às redes *mix* de Chaum para comunicações anônimas [Chaum, 1981]. Ao invés das mensagens serem transportadas diretamente da origem para o destino, elas passam por cadeias de nodo a nodo em que cada canal é cifrado individualmente, até que a mensagem chegue ao destinatário. Como cada nodo na cadeia conhece apenas seus vizinhos imediatos, não há como determinar a identidade dos nodos que publicam, dos nodos que estão armazenando os objetos e dos nodos de onde partem as requisições para recuperar os objetos.

Negação de objetos armazenados. Por razões legais e/ou políticas, visando prover o aspecto *negabilidade* aos nodos, todos os objetos armazenados são cifrados. Os procedimentos de criptografia utilizados não têm por objetivo tornar o conteúdo do objeto confidencial, uma vez que qualquer nodo solicitante deve ser capaz de decifrá-lo ao recuperá-lo. Ao contrário, o objetivo é que o operador do nodo possa *negar* o conhecimento sobre o conteúdo do objeto, já que ele conhece apenas a chave do objeto e não a chave usada para cifrá-lo.

As chaves criptográficas de objetos armazenados com chaves *signed-subspace* só podem ser obtidas revertendo a operação de *hash*. Já as chaves criptográficas para objetos armazenados com chaves *content-hash* são completamente não relacionadas. Assim, apenas um ataque de força bruta permitiria ao operador do nodo ter acesso ao conteúdo dos objetos por ele armazenados.

4.4.5. Free Haven

Free Haven [Dingledine et al., 2001] é um sistema de armazenamento distribuído baseado em uma comunidade de nodos denominados *servnets*. Nessa comunidade, cada nodo hospeda objetos de outros nodos em troca de uma oportunidade para armazenar seus próprios objetos nessa rede posteriormente. A rede composta pelos *servnets* é dinâmica: objetos migram de um nodo a outro com frequência, considerando a relação de confiança (*trust*) que cada nodo mantém com os demais. Os nodos transferem objetos a partir de negociações entre eles (*trading*).

Cada nodo possui uma chave pública e um ou mais blocos de retorno (*reply blocks* [Mazieres and Kaashoek, 1998]), que juntos fornecem comunicação segura, autenticada e pseudônima com o mesmo. Cada nodo na *servnet* possui uma base de dados que contém a chave pública e os blocos de retorno dos demais servidores.

Objetos são divididos em *shares* e armazenados em nodos distintos. Nodos que publicam atribuem uma data de validade aos objetos publicados. Os nodos se comprometem a manter os *shares* de um determinado objeto até que sua validade expire. Para estimular comportamento honesto, alguns nodos verificam se outros nodos descartam seus *shares* antes do combinado e decrementam sua confiança nesses nodos. Essa confiança é monitorada e atualizada através de um sistema de *reputação*. Cada nodo mantém uma base de dados contendo valores de confiança (*reputação*) dos demais nodos.

Recuperação e inserção de objetos. Para inserir um objeto no sistema, o nodo que publica utiliza o algoritmo para dispersão de informação proposto em [Rabin, 1989] para dividir o objeto nos *shares* f_1, \dots, f_n , onde quaisquer k *shares* são suficientes para recriar o objeto. Em seguida, o nodo gera um par de chaves criptográficas (PK_{doc}, SK_{doc}) , seleciona e assina um segmento do objeto para compor cada *share* f_i , e insere esses *shares* resultantes em seu espaço de armazenamento local. Os atributos armazenados junto a cada *share* são: *timestamp*, data de expiração, chave pública usada para assiná-lo (para verificação de integridade), número do *share* e assinatura.

Cada objeto no sistema Free Haven é indexado pela chave $H(PK_{doc})$, que corresponde ao *hash* da chave pública empregada para assinar os *shares* que compõem o objeto. Para realizar a busca por um objeto, o nodo solicitante gera um par de chaves criptográficas $(PK_{client}, SK_{client})$ e um bloco de retorno. Então, envia uma mensagem de requisição via *broadcast* informando a chave $H(PK_{doc})$, a chave pública PK_{client} e o bloco de retorno. A mensagem é recebida por todos os nodos que o nodo solicitante conhece.

Ao receber uma mensagem de requisição, o nodo verifica se está armazenando algum *share* cuja chave corresponda à *hash* de PK_{doc} . Em caso afirmativo, o nodo cifra cada *share* usando a chave pública PK_{client} e o envia através do *remailer* ao bloco de retorno informado na mensagem de requisição. Esses *shares* chegarão ao nodo solicitante; no momento em que k ou mais *shares* tiverem sido recebidos, o nodo poderá recriar o objeto.

Comunicações anônimas. A comunicação entre nodos se dá por intermédio de uma rede para envio e recebimento de e-mails de forma anônima (*remailer network*) [Mazieres and Kaashoek, 1998]. Cada nodo do sistema Free Haven possui associado a si um ou mais blocos de retorno (*reply blocks*) nessa rede. Tais blocos consistem de instruções de roteamento que identificam como chegar a um destinatário. Essas instruções são sucessivamente cifradas para um conjunto de *remailers*, de modo que cada *remailer* só consiga identificar a identidade do próximo salto. As instruções contidas no núcleo do bloco, visíveis apenas pelo último *remailer*, revelam o destino final da mensagem.

Negociação de *shares*. Nodos trocam *shares* uns com os outros periodicamente. As razões para essa negociação se desdobram em quatro. Primeiro, para prover maior anonimato ao nodo que publica; se trocas ocorrem com grande frequência no sistema, não há como assumir que um nodo propondo uma troca seja o nodo que publica o *share* que ele tem a oferecer. Segundo, para permitir a entrada e a saída de nodos; a idéia é que um nodo que queira deixar o sistema negocie com outros nodos visando armazenar localmente apenas *shares* de curta duração para que, após expirarem, possa “sair” suavemente. Terceiro, para permitir o armazenamento de objetos com datas de expiração longas. Quarto, para evitar que alvos estáticos – por exemplo um nodo armazenando determinados *shares* – possam ser atacados visando provocar uma negação de serviço.

A negociação é finalizada com a troca de mensagens de confirmação, acompanhadas de “recibo”, entre os nodos envolvidos. Além disso, cada nodo envia recibos (a) ao *buddy* vinculado ao *share* do qual está se desfazendo e (b) ao *buddy* do *share* que passará a armazenar. O conceito de *buddy* é utilizado para prover reputação ao Free Haven e será explicado a seguir.

Reputação. Nodos maliciosos podem aceitar receber determinados *shares* e, propositalmente, falhar no momento de armazená-los. Para evitar esse tipo de comportamento, o sistema propõe a associação entre pares de *shares* de um mesmo objeto. Cada *share* fica responsável por manter informações sobre a localização do outro *share* ou *buddy*. Periodicamente, o nodo responsável por um determinado *share* envia uma requisição ao nodo que está armazenando o respectivo *buddy* para se certificar de que o mesmo continua existindo. Caso o mesmo não responda, o nodo que fez a requisição anuncia o problema ocorrido, revelando a identificação – no caso um pseudônimo – do nodo que estava responsável pelo armazenamento do *buddy*.

Diante dessa e de outras oportunidades que um nodo tem de levar vantagens sobre os demais, o sistema Free Haven utiliza um mecanismo de reputação que tem por função identificar e contabilizar nodos mal comportados. Cada nodo mantém duas informações sobre os demais: *reputação* e *credibilidade*. A primeira se refere ao grau de confiança de que um nodo está obedecendo a especificação do protocolo Free Haven. Já a segunda representa a crença de que as informações recebidas daquele nodo são verdadeiras.

Os nodos disseminam “referências” aos demais sempre que registrarem o término bem sucedido de uma negociação, suspeitarem que o *buddy* de um determinado *share* foi perdido ou quando os valores de reputação e credibilidade para um certo nodo mudarem substancialmente.

Ataques à anonimidade. Um conjunto de ataques pode ser realizado com o propósito de revelar informações sobre a identidade de elementos do sistema, comprometendo seu anonimato:

- **ataques ao anonimato do leitor:** um atacante pode desenvolver e publicar no sistema Free Haven uma espécie de vírus que automaticamente contacta um determinado *host* ao ser executado, revelando informações

sobre o nodo que recuperou o objeto. Outro ataque consiste em tornar-se um nodo tanto na *servnet* quanto na *mixnet* e tentar um ataque fim-a-fim, correlacionando, por exemplo, o trânsito das mensagens com a requisição por objetos. Ainda, um nodo comprometido pode divulgar que possui determinado objeto e verificar quem solicita pelo mesmo ou simplesmente monitorar mensagens de requisição por objetos e armazenar a sua origem. A partir daí, seria possível determinar perfis de uso do sistema e enquadrar usuários aos mesmos. Segundo Dingledine em [Dingledine et al., 2001], Free Haven previne esse tipo de ataque por empregar cada bloco de retorno para uma transação apenas.

- **ataques ao anonimato do nodo que armazena objetos:** um atacante pode criar *shares* bastante grandes propositalmente e tentar reduzir o conjunto de nodos conhecidos por sua capacidade em armazenar esses *shares*. Tal ataque compromete parcialmente o anonimato desses nodos. Um atacante pode, ainda, assumir o papel de um dos nodos e, nesse caso, coletar informações sobre o estado e a participação de outros nodos no sistema (ex: listas de nodos). Por fim, um ataque simples porém bastante danoso consiste na disseminação de um verme no sistema, que identifica os objetos armazenados nos nodos e os informa a uma aplicação externa.
- **ataques ao anonimato do nodo que publica:** o atacante pode assumir o papel de um nodo e registrar ações de publicação, procurando associar fonte/origem e tempo. Alternativamente, esse nodo malicioso pode observar nodos que potencialmente publicaram objetos recentemente e procurar determinar quem esteve se comunicando com eles no mesmo período.

Ataques à reputação. Este tipo de ataques visa a comprometer os mecanismos associados à identificação e à contabilização de nodos maliciosos. No caso do sistema Free Haven, os principais ataques à reputação são os seguintes.

- **traição simples (*simple betrayal*):** o atacante pode se tornar parte da rede de *servnets*, se comportar corretamente por tempo suficiente até construir uma boa reputação e, então, passar a agir maliciosamente apagando objetos armazenados localmente antes de sua data de expiração.
- **captação de camarada (*buddy coopting*):** se um nodo malicioso ou conluio obtém controle sobre o *share* e seu respectivo *buddy*, ele pode apagar ambos sem que tal ação repercuta no sistema.
- **recomendação falsa (*false referrals*):** um nodo malicioso pode disseminar “referências” falsas na rede ou enviá-las a um subconjunto de nodos, comprometendo o cálculo de reputação realizado nos mesmos.
- **aprisionamento (*entrapment*):** um nodo malicioso pode violar o protocolo Free Haven de várias maneiras. Quando outro nodo detecta esse mal

comportamento e o acusa, o nodo malicioso pode apresentar recibos que contradizem o delator, além de denunciar o nodo correto de enviar “referências” falsas.

4.4.6. Peer-to-Peer Security Layer

Conforme destacado na Introdução, aplicações P2P começam a ser aceitas em ambientes acadêmicos e corporativos. Por exemplo, aplicações de médio porte, cujos grupos são compostos de dezenas ou centenas de nodos (tal como compartilhamento de recursos e trabalho cooperativo), estão se tornando cada vez mais comuns. Este é o caso na arena corporativa, onde sistemas P2P permitem que instituições troquem serviços [Lawton, 2004].

Embora aplicações P2P possam contribuir para o compartilhamento de recursos e colaboração em larga escala, sua diversificação e disseminação são dificultadas pela atual falta de segurança. Ainda é difícil desenvolver aplicações P2P que atendam a múltiplas combinações de aspectos de segurança. Soluções atuais, como as apresentadas em [Kim et al., 2003, Park and Hwang, 2003, Ptptl, 2006], cobrem apenas aspectos específicos (como por exemplo autenticação e reputação) e não podem ser facilmente integradas. Adicionalmente, essas soluções não conseguem isolar a programação dos aspectos de segurança da aplicação, demandam um comportamento simétrico e uniforme de todos os nodos, e oferecem pouco ou nenhum suporte para implantação gradual. Para suprir essas lacunas é proposta em [Detsch et al., 2006] a *Peer-to-Peer Security Layer*, ou P2PSL, que permite a inclusão de funcionalidades de segurança em aplicações P2P e trata dos problemas de falta de integração, isolamento, assimetria e implantação gradual, recém mencionados.

P2PSL consiste em uma camada de segurança que é implementada e configurada de maneira independente da aplicação P2P e do *middleware* de comunicação subjacente. Os requisitos de segurança são satisfeitos por módulos que implementam diferentes técnicas de segurança. Tais módulos são combinados como peças de um quebra-cabeça para fornecer segurança de maneira flexível e assimétrica. Em consonância com a natureza intrinsecamente descentralizada de aplicações P2P, a definição e a configuração dos módulos a serem empregados são realizadas de maneira autônoma em cada nodo, pelo usuário local. A configuração da camada de segurança é feita com base em perfis, cada um servindo a diferentes necessidades de segurança. Nesse contexto, nodos remotos podem ser associados a um dos perfis definidos; nodos ainda desconhecidos são automaticamente associados a um perfil padrão. Além desta classificação de nodos remotos em perfis, a P2PSL utiliza um protocolo de configuração que guia o usuário e automatiza o processo de ajuste do nodo em relação ao restante da rede. Este protocolo, bastante relevante em função da típica heterogeneidade e dinamicidade existente em redes P2P, é descrito em [Detsch et al., 2006].

A Figura 4.4 ilustra um exemplo de uma rede de nodos usando P2PSL. As configurações para os nodos Peer2 e Peer3 são apresentadas em detalhe.

Observe o Perfil B definido pelo Peer2. O mesmo define que autenticação deve ser utilizada em todas as mensagens enviadas, enquanto autenticação e confidencialidade são exigidas de todas as mensagens recebidas. O Peer2, então, associa o Peer3 a esse perfil.

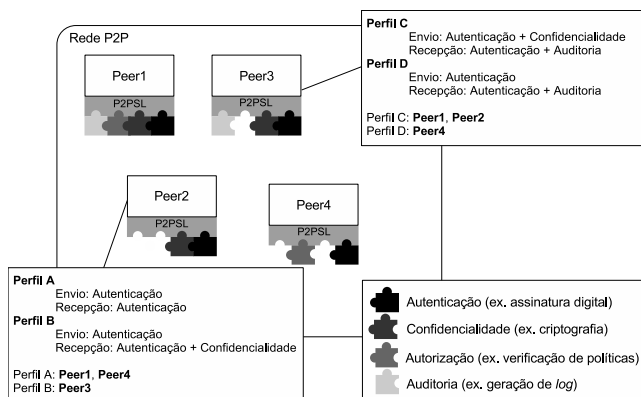


Figura 4.4. Exemplo de rede com P2PSL.

Durante a operação do sistema, a instância da P2PSL associada com o nodo se comporta como um invólucro entre a aplicação P2P e o *middleware* de comunicação subjacente. Sempre que mensagens são recebidas ou enviadas, os módulos envolvidos são acionados para satisfazer os requisitos de segurança estabelecidos. Dependendo do requisito de segurança sendo assegurado, um módulo pode provocar mudanças na mensagem. Este é o caso, por exemplo, de um módulo de assinatura digital, em que a assinatura é adicionada às mensagens em processo de envio.

A P2PSL foi implementada em Java, utilizando JXTA [Gong, 2001] como *middleware* de comunicação subjacente. JXTA é um projeto que busca estabelecer um conjunto de protocolos independentes de implementação que permitam a criação de uma estrutura P2P de propósito geral, empregável por diferentes aplicações.

Quatro módulos foram implementados para prover as seguintes funcionalidades de segurança: autenticação, confidencialidade, integridade, autorização e auditoria. Novos módulos podem ser adicionados sem alterações ou recompilação de código, como no caso da incorporação de um novo mecanismo de segurança ou escolha de uma técnica de segurança alternativa mais adequada para algum determinado cenário. Em linha com a filosofia da P2PSL, os módulos foram implementados de forma que nodos que não possuam a camada de segurança também possam participar do sistema. Isto permite uma adoção gradual da camada de segurança em sistemas P2P operacionais, e possibilita ao usuário de cada nodo a escolha de uso ou não da camada. Entretanto,

nodos que implementam a P2PSL podem bloquear mensagens enviadas por nodos que não a implementam exigindo, por exemplo, autenticação de mensagens desses nodos.

Inicialmente projetada para operar sobre redes P2P envolvendo um número reduzido de nodos (dezenas ou centenas), a P2PSL está sendo reavaliada com o objetivo de determinar a viabilidade técnica de seu uso em overlays de larga escala, estruturados ou não. Além da escalabilidade, aspectos como o comportamento da P2PSL em situações com elevada transiência de nodos e a robustez a ataques vêm sendo, atualmente, investigados pelo grupo (vide [P2P-SeC, 2006]).

4.5. Conclusões

Neste capítulo apresentamos uma síntese dos principais aspectos de segurança a serem considerados em redes P2P, destacando sua importância para o desenvolvimento de aplicações e sistemas P2P na Internet e a implantação de aplicações corporativas com necessidades mais críticas em termos de segurança.

Sistemas P2P não estão mais limitados a usuários domésticos, e começam a ser aceitos em ambientes acadêmicos e corporativos. Por exemplo, sistemas de armazenamento de arquivos em rede, de transmissão de dados, de computação distribuída e de colaboração também têm tirado proveito dessas redes. Este modelo de computação é atrativo por diversas razões. Primeiro, porque redes P2P são escaláveis, pois não possuem um ponto central de falhas ou gargalo, na forma de um servidor central. Segundo, resistem melhor a ataques intencionais como os de negação de serviço. Terceiro, tem o poder de atrair um grande número de usuários em função dos benefícios oferecidos pela coletividade sem no entanto abrir mão da autonomia de seus participantes.

Um dos principais desafios em P2P consiste em oferecer garantias para funcionamento seguro de aplicações P2P em configurações descentralizadas e de larga escala, que atravessam múltiplos domínios institucionais e congregam usuários e corporações com objetivos e demandas tão distintas.

Ao almejar que redes P2P sejam amplamente adotadas, elas precisam estar protegidas contra a ação de nodos maliciosos. Apresentamos diversos tipos de vulnerabilidades, de ataques que as exploram, e propostas de mecanismos de defesa para render tais ataques inócuos. Exemplos de vulnerabilidades discutidas neste capítulo são ataques ao sistema de roteamento (e as possíveis repercussões para o overlay), ataques à anonimidade de comunicações e ataques a sistemas de reputação. Problemas como estes fazem da área de segurança um dos principais campos de estudo em redes P2P.

Agradecimentos

Gostaríamos de agradecer aos alunos Ricardo N. Sanchez, pela prestimosa colaboração na preparação de figuras e na organização da base de dados de artigos (com mais de 300 entradas), e a Juliano Freitas da Silva, pela revisão ortográfica do texto e sugestões de melhorias. Aos editores, nossa gra-

tidão pela cuidadosa revisão e sugestões, que permitiram que melhorássemos o texto final.

Referências

- [Azu, 2006] (2006). Anonymous BitTorrent.
<http://azureus.sourceforge.net/doc/AnonBT/>.
- [Adar and Huberman, 2000] Adar, E. and Huberman, B. (2000). Free riding on gnutella. *First Monday (Peer-Reviewed Journal of the Internet)*, 5(10).
- [Andrade et al., 2004] Andrade, N., Brasileiro, F., Cirne, W., and Mowbray, M. (2004). Discouraging free riding in a peer-to-peer cpu-sharing grid. In *13th IEEE Symposium on High Performance Distributed Computing (HPDC'04)*.
- [Bittorrent, 2006] Bittorrent (2006). BitTorrent website.
<http://www.bittorrent.com/>.
- [Cashmere, 2006] Cashmere (2006). Cashmere: Resilient anonymous routing. <http://p2p.cs.ucsb.edu/cashmere/>.
- [Castro et al., 2002] Castro, M., Drushel, P., Ganesh, A., Rowstron, A., and Wallach, D. S. (2002). Secure routing for structured peer-to-peer overlay networks. In *5th Usenix Symposium on Operating Systems Design and Implementation*, Boston, MA, USA.
- [Chaum, 1981] Chaum, D. L. (1981). Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90.
- [Cheng and Friedman, 2005] Cheng, A. and Friedman, E. (2005). Sybilproof reputation mechanisms. In *P2PECON '05: Proceeding of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems*, pages 128–132, New York, NY, USA. ACM Press.
- [ChimeraTapestry, 2006] ChimeraTapestry (2006). Chimera and Tapestry website. <http://p2p.cs.ucsb.edu/chimera/>.
- [Chord, 2006] Chord (2006). The Chord Project.
<http://pdos.csail.mit.edu/chord/>.
- [Clarke et al., 2001] Clarke, I., Sandberg, O., Wiley, B., and Hong, T. W. (2001). Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*, 2009:46–66.
- [Cohen, 2003] Cohen, B. (2003). Incentives build robustness in bittorrent. In *Proc. 1st Workshop on Economics of Peer-to-Peer Systems*, Berkeley.
- [Comscore, 2006] Comscore (2006). Press release on global Napster usage.
<http://www.comscore.com/press/release.asp?id=249>.
- [Condie et al., 2006] Condie, T., Kacholia, V., Sank, S., Hellerstein, J. M., and Maniatis, P. (2006). Induced churn as shelter from routing-table poisoning. In *Network and Distributed System Security Symposium (NDSS '06)*.

- [Dabek et al., 2001a] Dabek, F., Brunskill, E., Kaashoek, F. M., Karger, D., Morris, R., Stoica, I., and Balakrishnan, H. (2001a). Building peer-to-peer systems with chord, a distributed lookup service. In *8th Workshop on Hot Topics in Operating Systems (HotOS)*, pages 81–86.
- [Dabek et al., 2001b] Dabek, F., Kaashoek, F. M., Karger, D., Morris, R., and Stoica, I. (2001b). Wide-area cooperative storage with cfs. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, volume 35, pages 202–215, New York, NY, USA. ACM Press.
- [Damiani et al., 2002] Damiani, E., di Vimercati, D. C., Paraboschi, S., Samarati, P., and Violante, F. (2002). A reputation-based approach for choosing reliable resources in peer-to-peer networks. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, pages 207–216, New York, NY, USA. ACM Press.
- [Danezis and Anderson, 2005] Danezis, G. and Anderson, R. (2005). The economics of resisting censorship. *IEEE Security and Privacy*, 3(1):45–50.
- [Danezis et al., 2005] Danezis, G., Lesniewski-Laas, C., Kaashoek, F. M., and Anderson, R. (2005). Sybil-resistant DHT routing. In *ESORICS 2005: 10th European Symposium on Research in Computer Security*, LNCS 3679. Springer.
- [Daswani and Molina, 2002] Daswani, N. and Molina, H. G. (2002). Query-flood DoS attacks in Gnutella. In *ACM Conference on Computer and Communications Security*, pages 181–192.
- [Daswani and Molina, 2004] Daswani, N. and Molina, H. G. (2004). Pong-cache poisoning in guess. In *11th ACM Conference on Computer and Communications Security (CCS 2004)*.
- [Detsch et al., 2006] Detsch, A., Gasparly, L. P., Barcellos, M. P., and Sanchez, R. N. (2006). Flexible security configuration & deployment in peer-to-peer applications. In *10th IEEE/IFIP Network Operations and Management Symposium (NOMS 2006)*, pages 1–11.
- [Dingledine et al., 2001] Dingledine, R., Freedman, M. J., and Molnar, D. (2001). The free haven project: distributed anonymous storage service. In *International workshop on Designing privacy enhancing technologies*, pages 67–95, New York, NY, USA. Springer-Verlag New York, Inc.
- [Dingledine et al., 2004] Dingledine, R., Mathewson, N., and Syverson, P. (2004). Tor: The second-generation onion router. In *13th USENIX Security Symposium*.
- [Douceur, 2002] Douceur, J. R. (2002). The sybil attack. In *1st International Workshop on Peer-to-Peer Systems*, pages 251–260.
- [Druschel and Rowstron, 2001] Druschel, P. and Rowstron, A. (2001). Past: A large-scale, persistent peer-to-peer storage utility. In *Eighth Workshop on Hot Topics in Operating Systems (HotOS)*, pages 75–80.

- [Dumitriu et al., 2005] Dumitriu, D., Knightly, E., Kuzmanovic, A., Stoica, I., and Zwaenepoel, W. (2005). Denial-of-service resilience in peer-to-peer file sharing systems. In *ACM SIGMETRICS'05*, pages 38–49, Alberta.
- [eBay, 2006] eBay (2006). eBay website. <http://www.ebay.com/>.
- [eDonkey, 2006] eDonkey (2006). edonkey2000 - overnet website. <http://edonkey2000.com/>.
- [ESM, 2006] ESM (2006). End System Multicast website. <http://esm.cs.cmu.edu/>.
- [Farsite, 2006] Farsite (2006). Farsite: Federated, available, and reliable storage for an incompletely trusted environment. <http://research.microsoft.com/farsite/>.
- [Feldman and Chuang, 2005] Feldman, M. and Chuang, J. (2005). Overcoming free-riding behavior in peer-to-peer systems. *SIGecom Exch.*, 5(4):41–50.
- [Feldman et al., 2004] Feldman, M., Lai, K., Stoica, I., and Chuang, J. (2004). Robust incentive techniques for peer-to-peer networks. In *EC '04: Proceedings of the 5th ACM conference on Electronic commerce*, pages 102–111, New York, NY, USA. ACM Press.
- [Freedman and Morris, 2002] Freedman, M. J. and Morris, R. (2002). Tarzan: a peer-to-peer anonymizing network layer. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, pages 193–206, New York, NY, USA. ACM Press.
- [Genome, 2006] Genome (2006). Genome@home website. <http://genomeathome.stanford.edu/>.
- [Gkantsidis et al., 2004] Gkantsidis, C., Mihail, M., and Saberi, A. (2004). Random walks in peer-to-peer networks. In *INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1.
- [Gnutella, 2006] Gnutella (2006). Gnutella website. <http://www.gnutella.com/>.
- [Gong, 2001] Gong, L. (2001). Jxta: A network programming environment. *IEEE Internet Computing*, 5:88–95.
- [Grandison and Sloman, 2000] Grandison, T. and Sloman, M. (2000). A survey of trust in internet applications. *IEEE Communications Surveys*, 3(4):2–16.
- [Gupta et al., 2003] Gupta, M., Judge, P., and Ammar, M. (2003). A reputation system for peer-to-peer networks. In *NOSSDAV '03: Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video*, pages 144–152, New York, NY, USA. ACM Press.

- [Gupta and Somani, 2004] Gupta, R. and Somani, A. K. (2004). Reputation management framework and its use as currency in large-scale peer-to-peer networks. In *4th International Conference on Peer-to-Peer Computing (P2P'04)*, pages 124–132.
- [Harding, 1968] Harding, G. (1968). The tragedy of the commons. *Science*, 162:1243–1248.
- [ICQ, 2006] ICQ (2006). ICQ.com website. <http://www.icq.com/>.
- [Ivy, 2006] Ivy (2006). Ivy: Read-write peer-to-peer filesystem. <http://pdos.csail.mit.edu/ivy/>.
- [Jabber, 2006] Jabber (2006). Jabber: Open Instant Messaging. <http://www.jabber.org/>.
- [JetFile, 2006] JetFile (2006). Jetfile. <http://www.sics.se/cna/jetfile.html>.
- [Josang, 1998] Josang, A. (1998). A subjective metric of authentication. In *ESORICS '98: Proceedings of the 5th European Symposium on Research in Computer Security*, pages 329–344, London, UK. Springer-Verlag.
- [Jun and Ahamad, 2005] Jun, S. and Ahamad, M. (2005). Incentives in BitTorrent induce free riding. In *ACM SIGCOMM Workshop on Economics of Peer-to-Peer systems (P2P-ECON)*, pages 116–121.
- [Kamvar et al., 2003] Kamvar, S. D., Schlosser, M. T., and Garcia-Molina, H. (2003). The eigenTrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th international conference on World Wide Web (WWW '03)*, pages 640–651, New York, NY, USA. ACM Press.
- [Kim et al., 2003] Kim, Y., Mazzocchi, D., and Tsudik, G. (2003). Admission control in peer groups. In *2nd IEEE International Symposium on Network Computing and Applications*, pages 131–140.
- [Lawton, 2004] Lawton, G. (2004). Is peer-to-peer secure enough for corporate use? *IEEE Computer*, 37(1):22–25.
- [Liang et al., 2004] Liang, J., Kumar, R., and Ross, K. W. (2004). The kazaa overlay: A measurement study. In *19th IEEE Annual Computer Communications Workshop (CCW 2004)*.
- [Lua et al., 2005] Lua, E. K., Crowcroft, J., Pias, M., Sharma, R., and Lim, S. (2005). A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys & Tutorials*, 7(2):72–93.
- [Marti and Garcia-Molina, 2006] Marti, S. and Garcia-Molina, H. (2006). Taxonomy of trust: Categorizing p2p reputation systems. *Computer Networks*, 50(4):472–484.
- [Maymounkov and Mazieres, 2002] Maymounkov, P. and Mazieres, D. (2002). Kademlia: A peer-to-peer information system based on the xor metric. In *International Peer-to-Peer Symposium (IPTPS02)*.

- [Mazieres and Kaashoek, 1998] Mazieres, D. and Kaashoek, F. M. (1998). The design, implementation and operation of an email pseudonym server. In *CCS '98: Proceedings of the 5th ACM conference on Computer and communications security*, pages 27–36, New York, NY, USA. ACM Press.
- [Mojo, 2006] Mojo (2006). Mojo nation project. <http://freshmeat.net/projects/mojonation/>.
- [MSN, 2006] MSN (2006). MSN.com website. <http://www.msn.com/>.
- [Murdoch and Danezis, 2005] Murdoch, S. J. and Danezis, G. (2005). Low-cost traffic analysis of tor. In *2005 IEEE Symposium on Security and Privacy*.
- [OceanStore, 2006] OceanStore (2006). The OceanStore Project website. <http://oceanstore.cs.berkeley.edu/>.
- [OpenNap, 2006] OpenNap (2006). OpenNap: Open Source Napster Server website. <http://opennap.sourceforge.net/>.
- [P2P-SeC, 2006] P2P-SeC (2006). P2P-SeC: Secure and Reliable Computing for Peer-to-Peer Networks. <http://p2p-sec.org>.
- [Park and Hwang, 2003] Park, J. S. and Hwang, J. (2003). Role-based access control for collaborative enterprise in peer-to-peer computing environments. In *8th ACM Symposium on Access Control Models and Technologies*.
- [PAST, 2006] PAST (2006). PAST: A large-scale, peer-to-peer archival storage facility. <http://research.microsoft.com/antr/PAST/default.htm>.
- [Pastry, 2006] Pastry (2006). Pastry: A substrate for peer-to-peer applications. <http://research.microsoft.com/antr/pastry/>.
- [Plaxton et al., 1997] Plaxton, G. C., Rajaraman, R., and Richa, A. W. (1997). Accessing nearby copies of replicated objects in a distributed environment. In *SPAA '97: Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures*, pages 311–320, New York, NY, USA. ACM Press.
- [Ptptl, 2006] Ptptl (2006). The peer-to-peer trusted library website. <http://sourceforge.net/projects/ptptl>.
- [Rabin, 1989] Rabin, M. O. (1989). Efficient dispersal of information for security, load balancing, and fault tolerance. *J. ACM*, 36(2):335–348.
- [Ratnasamy et al., 2001] Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Shenker, S. (2001). A scalable content addressable network. In *ACM SIGCOMM 2001*, pages 161–172.
- [Reed et al., 1998] Reed, M. G., Syverson, P. F., and Goldschlag, D. M. (1998). Anonymous connections and onion routing. *Selected Areas in Communications, IEEE Journal on*, 16(4):482–494.

- [Rocha et al., 2004] Rocha, J., Domingues, M., Callado, A., Souto, E., Silvestre, G., Kamienski, C., and Sadok, D. (2004). Peer-to-peer: Computacao colaborativa na internet. In *Minicursos do XXII Simposio Brasileiro de Redes de Computadores (SBRC 2004)*.
- [Rowstron and Druschel, 2001] Rowstron, A. and Druschel, P. (2001). Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *18th ACM Symposium on Operating System Principles (SOSP'01)*.
- [Saroiu et al., 2002] Saroiu, S., Gummadi, P., and Gribble, S. (2002). A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking*, pages 156–170.
- [Scribe, 2006] Scribe (2006). Scribe: a scalable group communication system. <http://research.microsoft.com/antr/Scribe/default.htm>.
- [SETI, 2006] SETI (2006). SETI@home website. <http://setiathome.ssl.berkeley.edu/>.
- [Shamir, 1979] Shamir, A. (1979). How to share a secret. *Commun. ACM*, 22(11):612–613.
- [Sieka et al., 2004] Sieka, B., Kshemkalyani, A. D., and Singhal, M. (2004). On the security of polling protocols in peer-to-peer systems. In *Peer-to-Peer Computing*, pages 36–44.
- [Singh et al., 2004] Singh, A., Castro, M., Druschel, P., and Rowstron, A. (2004). Defending against eclipse attacks on overlay networks. In *11th ACM SIGOPS European Workshop*.
- [Singh and Liu, 2003] Singh, A. and Liu, L. (2003). Trustme: Anonymous management of trust relationships in decentralized p2p systems. In *3rd International Conference on Peer-to-Peer Computing (P2P'03)*, pages 142–149.
- [Singh et al., 2006] Singh, A., Ngan, T.-W., Druschel, P., and Wallach, D. S. (2006). Eclipse attacks on overlay networks: Threats and defenses. In *25th Conference on Computer Communications (INFOCOM 2006)*. IEEE.
- [Sit and Morris, 2002] Sit, E. and Morris, R. (2002). Security considerations for peer-to-peer distributed hash tables. In *1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*.
- [Skype, 2006] Skype (2006). Skype. <http://www.skype.com/>.
- [Slyck, 2006] Slyck (2006). Slyck's p2p network stats page. <http://www.slyck.com/stats.php>.
- [Song et al., 2005] Song, S., Hwang, K., Zhou, R., and Kwok, Y. K. (2005). Trusted p2p transactions with fuzzy reputation aggregation. *Internet Computing, IEEE*, 9(6):24–34.

- [Srivatsa and Liu, 2004] Srivatsa, M. and Liu, L. (2004). Vulnerabilities and security threats in structured overlay networks: A quantitative analysis. In *20th Annual Computer Security Applications Conference (ACSAC'04)*.
- [Srivatsa et al., 2005] Srivatsa, M., Xiong, L., and Liu, L. (2005). Trustguard: countering vulnerabilities in reputation management for decentralized overlay networks. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 422–431, New York, NY, USA. ACM Press.
- [Stoica et al., 2003] Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, F. M., Dabek, F., and Balakrishnan, H. (2003). Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 11(1):17–32.
- [Theotokis and Spinellis, 2004] Theotokis, S. A. and Spinellis, D. (2004). A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, 36(4):335–371.
- [Tor, 2006] Tor (2006). Tor: An anonymous internet communication system. <http://tor.eff.org/>.
- [Tran et al., 2005] Tran, H., Hitchens, M., Varadharajan, V., and Watters, P. (2005). A trust based access control framework for p2p file-sharing systems. In *HICSS '05: Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05) - Track 9*, Washington, DC, USA. IEEE Computer Society.
- [Tsoumakos and Roussopoulos, 2003] Tsoumakos, D. and Roussopoulos, N. (2003). A comparison of peer-to-peer search methods. In *6th International Workshop on the Web and Databases (WebDB 2003)*.
- [Wallach, 2002] Wallach, D. S. (2002). A survey of peer-to-peer security issues. In *International Symposium on Software Security*, pages 42–57.
- [Xiong and Liu, 2004] Xiong, L. and Liu, L. (2004). Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities. *IEEE Transactions on Knowledge and Data Engineering*, 16(7):843–857.
- [Xuan et al., 2003] Xuan, D., Chellappan, S., and Krishnamoorthy, M. (2003). Rchord: An enhanced chord system resilient to routing attacks. In *International Conference on Computer Networks and Mobile Computing (ICCNMC 2003)*, volume 1.
- [Yahoo, 2006] Yahoo (2006). Yahoo! messenger. <http://messenger.yahoo.com/>.
- [Zhao et al., 2001] Zhao, B. Y., Kubiawicz, J. D., and Joseph, A. D. (2001). Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley.