Estratégia de proteção contra ataques de poluição em Sistemas P2P de Compartilhamento de Arquivos

Lucas A. Seewald, Rodolfo S. Antunes, Marinho P. Barcellos, Gabriel Pedebos, Marlom A. Konrath
PIPCA – UNISINOS
São Leopoldo, Brasil

Luciano P. Gaspary
PGCC - UFRGS
Porto Alegre, Brasil

Juliano F. Silva
IBM Brasil
Porto Alegre, Brasil

Resumo

Apesar de ser uma das principais aplicações distribuídas na atualidade, o compartilhamento de arquivos P2P tem sido fortemente prejudicado por ataques de poluição de conteúdo. Em um trabalho anterior, apresentamos uma estratégia de contenção de poluição que consiste em controlar a taxa na qual um conteúdo é disseminado. A taxa de disseminação de uma versão é determinada de acordo com a sua reputação e assegurada através de um mecanismo de autorização. Neste artigo, aperfeiçoamos o esquema anterior e avaliamos por simulação quatro mecanismos distribuídos que implementam a estratégia de contenção. Os resultados mostram que a estratégia é eficaz para ambos os casos de versões poluídas e não-poluídas, causando uma pequena penalidade, mesmo com ataques em conluio de uma parcela significativa de pares maliciosos. Os resultados das simulações obtidos usando configurações realísticas indicam que o mecanismo de contenção de poluição pode efetivamente reduzir a poluição a um baixo custo.

1. Introdução

Compartilhamento de arquivos P2P é uma das aplicações distribuídas mais importantes hoje em dia. Existem estimativas de que mais de 8 milhões de usuários estejam conectados simultaneamente às redes *FastTrack/Kazaa*, *Gnutella*, *eDonkey* e *eMule* [7]. Em uma rede P2P, conteúdo é disponibilizado na forma de *títulos*, que podem ser músicas, vídeos, fotos, documentos ou programas. Podem existir diferentes *versões* de um mesmo título, sendo elas diferenciadas por um identificador único, normalmente gerado com base no *hash* do seu conteúdo.

Poluição de conteúdo é atualmente uma das maiores ameaças a redes de compartilhamento de arquivos P2P. Este ataque tem o objetivo de impedir a disseminação de um determinado conteúdo. Seu *modus operandi* típico é disponibilizar quantidades massivas de versões incorretas de um dado conteúdo na rede P2P. Usuários incapazes de distinguir versões incorretas (poluídas) de versões corretas (não-poluídas) podem realizar o download de múltiplas versões antes de obter uma correta. O ataque é muito comum atualmente: em 2004 mais de 50% dos arquivos de música populares na rede FastTrack/Kazaa estavam poluídos e chegando à 80% de poluição a determinados títulos [9]. Assumindo que usuários tentam obter o mesmo título através de downloads repetidos até que encontrem a versão apropriada, a poluição afeta usuários e provoca uma sobrecarga de comunicação.

Em um trabalho anterior [12], propusemos um método de contenção de poluição no qual a estratégia é limitar o número de downloads simultâneos de uma versão, isto é, limitar a taxa de disseminação. Sua filosofia é estabelecer o limite de acordo com a *reputação* da versão de um determinado título, que é adquirida ou perdida pelo download e voto sobre a integridade do conteúdo. Este mecanismo está presente desde o momento em que o conteúdo é publicado, e aplicado para todas as novas versões de um título.

No presente artigo, são explorados quatro mecanismos distribuídos para contenção de poluição. Resultados de simulação são discutidos e permitem comparar a eficiência e eficácia de cada mecanismo. Foi medida a penalidade imposta na disseminação de uma versão não-poluída, assim como sua eficiência na redução da disseminação de uma versão poluída.

O restante do artigo está organizado como segue. A Seção 2 discute trabalhos relacionados à poluição de conteúdo em redes P2P. A Seção 3 resume a estratégia de controle de poluição usando um modelo simplificado. A Seção 4 trata do problema em um contexto distribuído, apresentando quatro mecanismos de contenção de poluição distribuída. Estes mecanismos são avaliados e comparados na Seção 5. A Seção 6 finaliza o artigo apresentando as conclusões e perspectivas para trabalhos futuros.

2. Trabalhos Relacionados

Poluição de conteúdo é um assunto que recentemente tem despertado grande interesse da comunidade científica. Diversos trabalhos têm se concentrado na detecção de poluição em sistemas P2P e na modelagem da disseminação de poluição e avaliando seu impacto nocivo [3, 8, 7, 14].

Em [10, 11] é proposta a criação de uma "lista-negra" (*blacklist*) baseada na avaliação de metadados e um sistema global de reputação de sub-redes. A lista é composta de faixas de IP rotuladas como poluidoras, sendo construída com base na densidade de arquivos corrompidos disponibilizados. Nessa abordagem há limitações ao caracterizar determinada rede como poluidora com base na alta densidade de versões devido à popularidade de NAT (*Network Address Translation*).

Credence [16] é um sistema de reputação de objetos que possibilita ao usuário obter informações sobre a integridade de uma versão. Uma limitação do Credence reside no fato de que uma versão precisa de diversos downloads para que sua reputação seja estabelecida. Em um ataque de poluição no qual *muitas* versões diferentes e poluídas do mesmo conteúdo são geradas, e a reputação *inicial* é desconhecida, um número substancial de usuários terá que fazer o download de uma versão antes que a mesma seja não recomendada. Ou seja, o sistema não consegue combater eficientemente o problema se houver um grande número de versões.

Um aspecto chave nesta discussão é que um conteúdo poluído pode ser redistribuído por usuários corretos por um tempo considerável, até que eles verifiquem a integridade do conteúdo [7, 4]. Isso é particularmente verdade em sistemas que permitem a disseminação parcial do conteúdo durante o download, disseminando conteúdo quando os usuários ainda não verificaram e votaram no mesmo. Em contraste, em nosso mecanismo de contenção, o controle é aplicado desde o início para todos os novos títulos de uma versão e a taxa de disseminação ajustada de acordo com a *confiança* na integridade do conteúdo.

O presente artigo representa uma extensão de um trabalho anterior. Em [12], introduzimos a estratégia de contenção através do limite do número de downloads simultâneos. Naquele trabalho, descrevemos de forma geral quatro mecanismos distribuídos, sem avaliar os mesmos. O presente artigo traz diversas melhorias. Primeiro, ajusta a estratégia de contenção, cuja fórmula para obtenção do número de downloads autorizados foi modificada para um comportamento linear, conforme descrito na próxima seção. Segundo, apresenta uma avaliação de desempenho baseada em resultados de simulação obtidos com a implementação dos quatro mecanismos distribuídos que haviam sido propostos. Por último, mostra que o mecanismo sendo proposto é eficiente e eficaz em cenários realísticos.

3. Estratégia de Contenção de Poluição

Esta seção revisa a estratégia de contenção de poluição proposta em [12]. A estratégia é descrita com base em um modelo simplificado, que assume memória compartilhada, infinitos processadores e pares que respeitam o protocolo estipulado. Sem perda de generalidade, a modelagem é feita com base em uma única versão de um título. Pares disponibilizam cópias de uma versão, que exceto no momento inicial, são obtidas por download de outros pares. Assume-se que os *sugadores* (pares que ainda não possuem uma cópia do arquivo/versão) chegam concentrados logo depois do conteúdo ser publicado, e seus tempos entre as chegadas seguem uma distribuição de Poisson [5]. Um *semeador* (par que possui o arquivo completo) pode realizar upload concorrentemente a um dado número de sugadores, denotado por δ ($\delta \geq 1$). Entretanto, o número total de uploads que um par pode realizar é dado por ρ ($\rho \in N \wedge \rho \geq \delta$).

Conseqüentemente, sem nenhuma contenção de poluição, uma versão será disseminada de acordo com os fatores acima (padrão de chegada do sugador, δ e ρ). Em contraste, quando o mecanismo está ativo, ele deve evitar que alguns sugadores realizem o download, reduzindo a disseminação de conteúdo poluído.

Pares precisam receber uma autorização antes de realizar o download de uma versão. Ao término do download o par avalia a integridade da versão e emite um voto, positivo ou negativo, sobre a mesma. Um *voto positivo* atesta a integridade da versão, enquanto o *voto negativo* atesta o contrário. O total de votos positivos e negativos são mantidos em duas variáveis denotadas por r and s, respectivamente. Estas são utilizadas para derivar um escore de reputação da versão $E[\omega]$, um valor no intervalo $[0, 1]$ calculado segundo a Lógica Subjetiva [6].

O mecanismo de contenção de poluição acompanha o número de downloads corrente e usa $E[\omega]$ para determinar se um novo download deve ser autorizado. Neste contexto, o presente mecanismo emprega duas variáveis chave: A e D, definidas a seguir. A representa o número máximo de downloads correntes a ser permitidos (*allowed*), enquanto D representa o número corrente de downloads sendo realizados. O valor de D é incrementado quando um download começa e decrementado quando ele termina (completado ou abortado). Finalmente, os valores A e D são empregados pelo mecanismo de contenção de poluição para garantir que $D \leq A$. Resumindo, o valor de A irá depender da contagem atual de votos, expressado por r e s, e a emissão de novos votos dependerá de A, da quantidade de semeadores, e de sugadores interessados.

O valor de A reflete a reputação de uma versão $E[\omega]$, conforme Eq. 1. Quando $r \gg s$, A converge para um limiar superior, denominado A_{free}. Quando este limiar é alcançado, assume-se que a versão é de fato correta e libera-se totalmente os downloads da mesma. Em contraste,

quando $r \ll s$, A converge para um limiar inferior, denotado como A_{min} (com $A_{min} \geq 1$). Os casos intermediários em que há incerteza sobre a corretude de uma versão terão valores de A situados proporcionalmente entre os limiares.

$$A = E[\omega](A_{free} - A_{min}) + A_{min} \quad (1)$$

Mesmo que o mecanismo de poluição permita que downloads aconteçam, podem não existir pares interessados. Portanto, é possível que $D \ll A$ em determinados momentos. Ressalta-se que as análises focam nas primeiras horas de uma sessão (12h), onde o conteúdo é considerado "quente" e centenas de pares estão esperando para requisitar seu download.

4. Mecanismo de Contenção de Poluição

Esta seção aplica a estratégia introduzida na seção anterior e descreve quatro mecanismos de contenção distribuídos. Dada a impossibilidade de trabalhar com valores precisos de A e D em um ambiente distribuído, o mecanismo é restrito a *estimativas* de A e D, representadas por $E[A]$ e $E[D]$, respectivamente. Existem diferentes formas de se manter $E[A]$ e $E[D]$ de maneira distribuída. As principais questões que afetam o mecanismos são: (a) grau de *descentralização*: esquemas descentralizados exigem alguma forma de coordenação a fim de manter A e assegurar que o valor D não exceda A; (b) grau de *segmentação*: a rede P2P pode ser vista como uma simples entidade ou organizada como uma federação de entidades menores independentes; (c) se o mecanismo de controle de poluição requer a utilização do conceito de Tabelas Hash distribuídas (Distributed Hash Table - DHT).

Apresenta-se a seguir os quatro mecanismos para controle de poluição distribuído identificados em [12], a saber: *Centralizado*, *Superpar*, *Baseado em Chord* e *Descentralizado*. A presente descrição é mais detalhada do que a anterior, haja visto que os mesmos foram desde então implementados. Nas próximas subseções é descrito como o seguinte conjunto de operações básicas são executadas: (a) obter uma estimativa de A, $E[A]$; (b) obter uma estimativa de D, $E[D]$; (c) decidir sobre a autorização do download (se $E[D] < E[A]$). Além destas operações, mecanismos irão ajustar o valor de $E[D]$ para refletir as variações em D.

Na discussão, T_a representa o tempo levado para tomar a decisão sobre a autorização do download; T_d, o tempo de execução do download, T_v, tempo até que o usuário verifique sua cópia e envie seu voto e RTT, o tempo de ida e volta de uma mensagem. Tipicamente em sistemas P2P, $T_d \gg RTT$, assim T_d é dominante (minutos ou horas) sobre os outros tempos (geralmente milisegundos).

4.1 Centralizado

Este mecanismo é o mais simples, estando baseado em um *gerente de download*, denotado por GD. Um GD controla todas as operações para uma versão de um dado título; o número de versões (ou títulos) gerenciados por um GD deveria ser restringido através de conceitos de escalabilidade. Assumindo que pares conhecem a identidade do GD correspondente, eles requisitam autorizações de download da versão enviando uma mensagem de REQUEST ao GD e esperando por uma resposta. O GD verifica o valor atual de $E[A]$, de $E[D]$ e imediatamente decide sobre a autorização, baseado nos valores locais disponíveis. Se $E[D] \geq E[A]$, então a requisição é negada: o GD envia DENY ao par, que deve tentar novamente mais tarde (note que o valor de $E[A]$ oscila dinamicamente segundo a reputação atual). Há um tempo mínimo de retentativa que é estabelecido seguindo uma política similar ao procedimento de *backoff* exponencial utilizado em algumas redes de meio compartilhado. Caso $E[D] < E[A]$, a autorização é concedida, o GD executa $D \leftarrow D + 1$ e responde ao par com uma mensagem GRANT, possivelmente contendo informações sobre a localização do recurso. Então, o par realiza o download da versão (representada pelo fato da mensagem RETR ser enviada ao par que possui o recurso). Quando ele termina o download, o par verifica a integridade do arquivo e envia uma mensagem de voto VOTE positiva ou negativa. O GD recebe o voto, atualiza r ou s corretamente e conseqüentemente seu $E[A]$, e então faz $D \leftarrow D-1$. A Figura 1 ilustra o mecanismo.

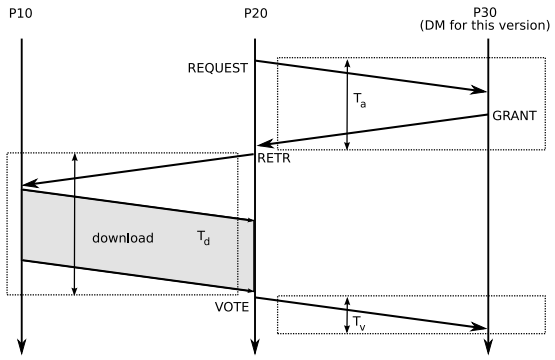

Figura 1. Exemplo ilustrando o mecanismo Centralizado

Há consenso na literatura sobre o impacto negativo de se adotar um elemento central. Na teoria, o mecanismo possui baixa escalabilidade, pois a presença de um GD representa um gargalo e ponto único de falha. Entretanto, na prática um GD pode manter o consumo de recursos em níveis aceitáveis através da limitação do número de versões e títulos sobre seu controle (tal como a arquitetura BitTorrent vale-se de um *tracker*). Além disso, existem outras

formas de se aumentar a escalabilidade de abordagens centralizadas, como por exemplo a adotada pelo Napster para comportar milhões de usuários simultâneos.

4.2 Superpar

Uma das alternativas naturais para o mecanismo centralizado é o conceito de dividir-para-conquistar, no qual a rede é organizada em *segmentos*. Cada um é controlado por um único gerente, que coordena e assegura as regras de downloads no segmento. O desafio entretanto, é como dividir a rede – quantos segmentos e qual o tamanho de cada um. Isso não é uma tarefa trivial por causa da escala da rede e a oscilação na população (*churning*). Nosso mecanismo beneficia-se da estrutura de superpares existente em redes P2P como FastTrack. Em tais redes, existem dois tipos de pares: pares normais e superpares. Este último passa a acumular o papel de GD, além de indexar arquivos de pares normais e ajudar a resolver buscas.

Um segmento abrange o superpar e um conjunto de pares que ele gerencia. A rede P2P, neste caso, é dividida em S segmentos, onde S é o número de superpares. Cada segmento é associado com um pedaço proporcional de A, $A_i = \frac{A}{S}$ global, assim proporcionalmente menos downloads devem ser autorizados. Do mesmo modo, $D_i = \frac{D}{S}$ e representa o número de downloads sendo realizados em um segmento (S_i). Assume-se que a alocação de pares normais a superpares seja relativamente balanceada.

A Figura 2 ilustra o funcionamento do mecanismo quando o par $P13$ deseja realizar o download da versão. O par $P13$ envia uma mensagem REQUEST ao seu superpar $P11$, que mantém-se a par das estimativas locais de $E[A_i]$ e $E[D_i]$. Se $E[D_i] < E[A_i]$, ele autoriza o download, enviando mensagem GRANT (ou DENY, caso contrário). Assim como no mecanismo centralizado, a mensagem GRANT pode conter informações da localização do recurso. Se a requisição é concedida, $P11$ incrementa $E[D_i]$. Finalmente, $P13$ envia um VOTE a $P11$, que irá atualizar $E[A_i]$. Este mecanismo precisa no mínimo alguma cooperação entre os superpares, para determinar o número de segmentos S, e quando possível, também o número total de pares. Note que superpares são interconectados através de uma rede não-estruturada. Além disso, valores considerando cada segmento (número de votos positivos e negativos e downloads correntes) devem ser trocados, permitindo estimativas globais serem computadas. Baseado em estimativas globais para o número de votos r e s, um superpar poderia manter um $E[A]$ global. Um dos inconvenientes deste modelo é a organização da rede. Em uma rede baseada em superpares, o papel do superpar é muito dinâmico; um par é escolhido de acordo com sua capacidade (atual). Além disso, contrariando a literatura, um par normal não tem um único superpar, ele tem muitos. Conseqüentemente, seria necessário

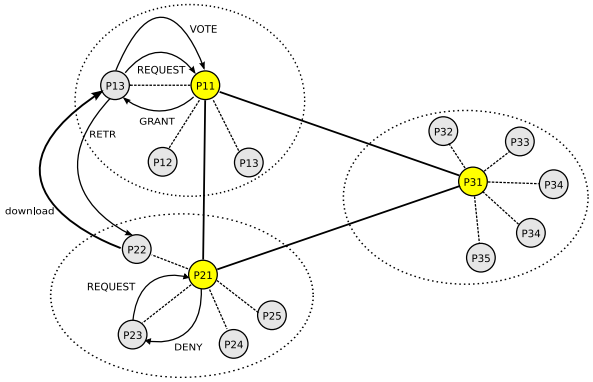

Figura 2. Exemplo ilustrando o mecanismo Superpar

aplicar uma regra (determinística) de modo que pares normais pudessem ser diretamente atribuídos a um único superpar.

4.3 Baseado em Chord

Esta aproximação resolve as dificuldades organizacionais anteriores, segmentando a rede com base em uma DHT ou, mais precisamente, o identificador do nodo no espaço de identificadores. Assim como no mecanismo anterior, ele é escalável e não tem um ponto central de falhas. Em contraste, existe uma regra determinística para formar segmentos e escolher cada GD.

Embora Chord [13] tenha sido escolhido como DHT para o mecanismo proposto, diferentes esquemas poderiam ser utilizados. Como no mecanismo de Superpares, o sistema P2P é dividido em segmentos, porém regularmente distribuídos em função de seus identificadores. O sistema, com 2^τ identificadores de pares, é particionado em 2^ϕ segmentos, cada um com $2^{\tau-\phi}$ IDs. O valor de A é adaptado de acordo com o tamanho de cada segmento, para representar uma fração do global: $A_i = \lceil A/2^\phi \rceil$. Refletindo isso, D ($E[D_i]$) é determinado de acordo com a soma de downloads ocorrendo no segmento. Entretanto, $A = \sum_{i=1}^{2^\phi} A_i$ e $D = \sum_{i=1}^{2^\phi} D_i$, onde 2^ϕ denota o número de segmentos.

Em cada segmento S_i, existe um gerente de download GD_i, que é autônomo para controlar o número de downloads permitidos no segmento (A_i) e o número corrente de downloads no segmento (D_i). O gerente do segmento é escolhido como o primeiro par do *próximo* segmento, de modo que um par encontre seu gerente resolvendo a primeira chave sob resposabilidade do próximo segmento. Além disso, GD_i é responsável por um segmento S_{i-1}. Baseado em valores locais de A_i e D_i, o gerente pode conceder ou rejeitar um download.

Para ilustração, assume-se que $\tau = 128$ e $\phi = 3$. A

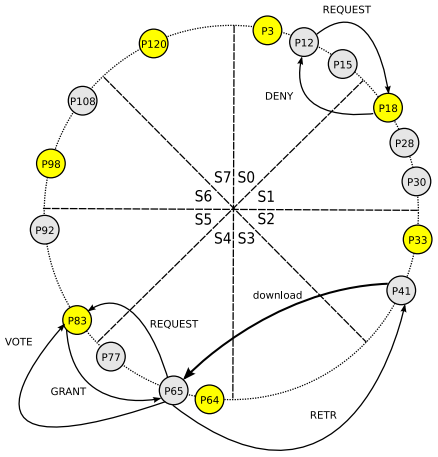

Figura 3. Exemplo ilustrando o mecanismo Baseado em Chord

Figura 3 representa como o mecanismo funciona: o par $P12$ resolve a chave 16, a primeira sob responsabilidade do próximo segmento, para encontrar seu gerente. O par que for responsável por tal chave será o gerente do segmento em que $P12$ se situa (e possivelmente outros). No exemplo, o sucessor e portanto GD é $P18$. $P12$ então envia uma mensagem REQUEST a $P18$, que observa, por exemplo $D_0 \geq A_0$, e assim rejeita o download. Em outro exemplo, $P65$ solicita autorização ao seu gerente, $P83$, que verifica que $D_4 < A_4$ e autoriza a mesma (e pode informar a localização onde o recurso pode ser encontrado). $P65$ então contata $P41$, que tem a cópia de uma versão e está desejando realizar upload. Como mostrado por este exemplo, a localização do seu recurso é ortogonal ao mecanismo de controle. Uma vez que o download é completado, $P65$ envia uma mensagem VOTE a $P83$ de modo que posteriormente pode atualizar $E[A]$.

Os valores de τ e ϕ são os principais parâmetros deste mecanismo. Como ϕ aproxima-se de τ, o número de segmentos aumenta e seu tamanho diminui. Em um extremo, se $\phi = \tau$, deveriam existir 2^τ segmentos com no mínimo um par (mas nunca nenhum) em cada. Neste caso, $A_i = 1$, que é, $A = 2^\tau$; isto é equivalente à inexistência de controle, pois todos os pares terão autorização garantida. No outro extremo, se $\phi = 0$, existe um simples segmento controlando todos os pares, caso específico que é idêntico ao centralizado.

4.4 Descentralizado

Diferente dos mecanismos anteriores, esta aproximação é completamente descentralizada e desestruturada: todos os pares são iguais e desempenham o mesmo papel. Cada par é autônomo para decidir se pode realizar o download ou não. Sua decisão é baseada na execução correta do protocolo proposto, que inclui a determinação das estimativas $E[A]$ e $E[D]$. Protocolos de acordo distribuído [15] não são apropriados para este tipo de problema, dada a sua escala.

O mecanismo segue os mesmos princípios da rede Gnutella: inundação de consultas com escopo limitado. Neste protocolo, quando um par deseja encontrar algum conteúdo, ele envia uma mensagem QUERY aos seus vizinhos, que verificam por uma cópia local e podem repassar a consulta aos vizinhos deles, e assim por diante, implementando a inundação através do *overlay*. Pares nos quais existe uma cópia enviam mensagem REPLY de volta através do mesmo roteador. É tipicamente impraticável consultar a rede inteira, então a pesquisa tem escopo limitado (*horizonte*) e pode não encontrar uma cópia existente de um conteúdo. Já no mecanismo proposto, uma mensagem GET_INFO é enviada à rede para coletar informações considerando o número de downloads sendo realizados (para determinar $E[D]$) assim como o número de votos positivos e negativos (para computar $E[A]$). O horizonte, definido como ψ, limita o escopo de GET_INFO. O mecanismo portanto compartilha vantagens e desvantagens de Gnutella, já extensivamente discutidas na literatura [2].

Assim, antes de requisitar um download, o par obtém estimativas de $E[A]$ e $E[D]$ através da consulta a outros pares: (a) se o par está atualmente realizando o download de uma versão; (b) se ele já tiver realizado-o, qual é o voto. Similarmente ao processo de consulta usual em redes não-estruturadas, quanto mais pares são consultados, mais precisa será a consulta.

A consulta aos valores em cada par é realizada através de um *algoritmo sistólico* através de overlay, como descrito na seqüência. Cada par P_i armazena uma tripla de valores binários (D_i, r_i, s_i) para representar, respectivamente, o número de downloads correntes desta versão neste par (ou 0 ou 1), se um voto positivo foi emitido, e se um voto negativo foi emitido. Além da tripla, um par mantém uma variável que indica se ele está *marcado* por uma dada operação de inundação ou não (usando o id do recurso). Inicialmente, todos os pares estão desmarcados. A rede é inundada com mensagens de requisição de informação, que retornam recursivamente à origem.

A Figura 4 ilustra um exemplo desta aproximação para uma rede com 7 pares e uma única consulta. O processo inicia com a transmissão da mensagem GET_INFO pela origem, $P10$, aos seus vizinhos, $P20$ and $P30$. Cada vizinho que permanece desmarcado (assumindo que todos são inicialmente) é então marcado e repassa a mensagem aos seus vizinhos, diferente do par do qual ele recebeu a mensagem.

Inundando a rede, mesmo que parcialmente, e esperando que respostas sejam retornadas ao par que iniciou a inundação, pode levar centenas de milisegundos. Embora não negligível, este tempo deve ser considerado na perspectiva no tempo de download, tipicamente na ordem de minutos ou horas. Entretanto, para obter informações de outros pares possivelmente não acrescentaria atrasos sig-

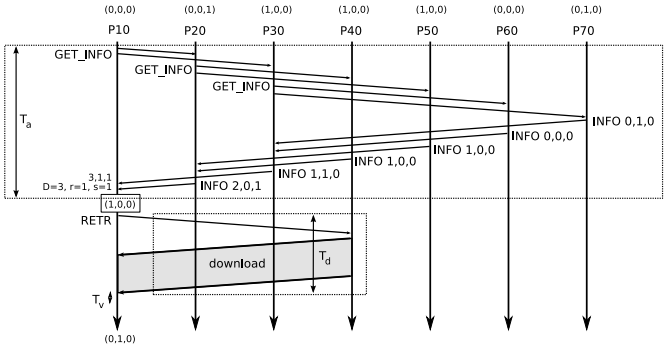

Figura 4. Exemplo ilustrando o mecanismo Descentralizado

nificativos. O mecanismo pode ser melhorado se um par não precisar consultar a rede toda vez para estimar $E[A]$ ou $E[D]$. Pares poderiam *manter* informações sobre estimativas de A e D e, periodicamente e sob demanda, iniciar uma nova inundação para realizar um download. Contudo, este esquema sofre com o alto custo de comunicação: a quantidade de mensagens para atualização é muito alta. Cada atualização irá solicitar a inundação da rede e coletar respostas; em um grande sistema, muitos pares irão consultar e tentar realizar download concorrentemente, não somente inundando a rede mas também obtendo informações imprecisas do estado atual. Por fim, uma alternativa é aproveitar as inundações que já são parte do protocolo de busca por um recurso, e anexar às mesmas uma consulta por valores (D, r, s), evitando-se nova inundação.

5. Avaliação do Mecanismo de Contenção de Poluição

Esta seção apresenta os resultados da avaliação que permitem comparar a eficiência dos quatro mecanismos apresentados na seção anterior. O funcionamento dos mecanismos foi modelado em detalhe via simulação discreta, e cada mecanismo foi implementado com o auxílio do arcabouço de simulação Simmcast [1]. Cada um dos modelos implementados reflete a descrição da seção anterior, incluindo troca de pacotes, autorização, download, etc.

Para efeito de avaliação, a eficiência em mecanismos de controle de poluição significa duas coisas: primeiro, ser capaz de detectar precisamente quando um conteúdo é poluído e reduzir substancialmente sua disseminação quando possível; segundo, *não* impedir a disseminação de um conteúdo não-poluído. Detectar se um conteúdo é poluído ou não depende do sistema reputação empregado, enquanto efetivar a decisão depende da escolha de mecanismo distribuído.

A estratégia de contenção de poluição proposta possui dois objetivos conflitantes: minimizar a disseminação de versões poluídas e ao mesmo tempo minimizar a penalidade imposta a versões não-poluídas. Assim, neste trabalho é avaliada a adequação da estratégia de acordo com estes objetivos. Conseqüentemente, são comparados os casos com e sem ataque de conluio, para versões poluídas e não-poluídas. Os seguintes cenários são apresentados e comparados:

1. versão não-poluída que é publicada por um par correto e disseminada sem ataque de conluio: a versão deveria ser disseminada tão rápida quanto os recursos permitirem;

2. versão poluída que é publicada por pares maliciosos e disseminada sem ataque de conluio: conteúdo poluído é obtido por pares corretos que votam negativamente;

3. versão não-poluída que é publicada por pares corretos mas sua disseminação é dificultada por um ataque em conluio: depois de algum tempo da versão ter sido publicada, pares maliciosos chegam em conjunto e iniciam o ataque. Os pares maliciosos competem por recursos, realizam o download da versão, e então votam negativo para indicar aos pares corretos que a versão é poluída, reduzindo potencialmente a disseminação de uma versão correta;

4. versão poluída que é publicada por pares maliciosos e disseminada com a ajuda de pares em conluio: conteúdo poluído é obtido por pares maliciosos que votam positivamente, enquanto pares corretos realizam download e votam negativamente, ajudando potencialmente a disseminar uma versão poluída.

São empregados os seguintes valores como parâmetros de entrada: (a) o tempo de simulação é limitado a 12h; (b) o número de pares corretos é definido em $C = 500$, de pares maliciosos em $M = 100$, e o número inicial de semeadores em $I = 1$; (c) T_d é definido como uma distribuição randômica Normal, no qual a média é T_d e o desvio padrão é um décimo da média; (d) quando uma requisição é negada, o tempo para a próxima tentativa é escolhido uniformemente entre 0 e $\frac{T_d}{2}$; (e) quando a segmentação é empregada (Superpar e Baseado em Chord), o número de segmentos é definido como $S = 16$; (f) a chegada de pares corretos é distribuída exponencialmente, enquanto pares maliciosos iniciam o ataque uma hora após o início sessão; (g) no esquema Baseado em Chord, os parâmetros são definidos como $\tau = 12$ e $\phi = 4$; (h) quando a rede é inundada, o mecanismo Descentralizado emprega um horizonte de $\psi = 6$.

Os valores apresentados são estatiscamente corretos. Resultam da execução de 60 repetições de cada experimento individual, empregando-se sementes aleatórias diferentes. A título de curiosidade, empregou-se um conjunto de 7

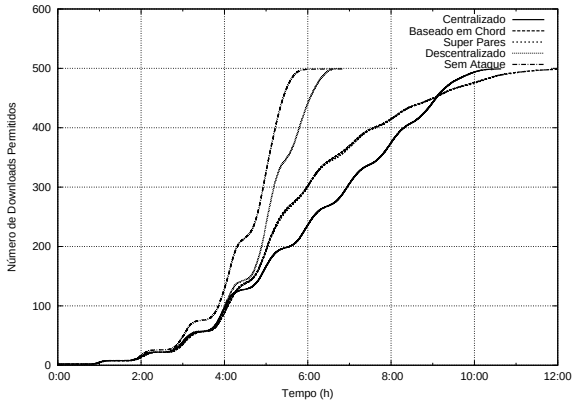

Figura 5. Downloads cumulativos de versão não poluída em cenário com 500 pares corretos e 100 maliciosos

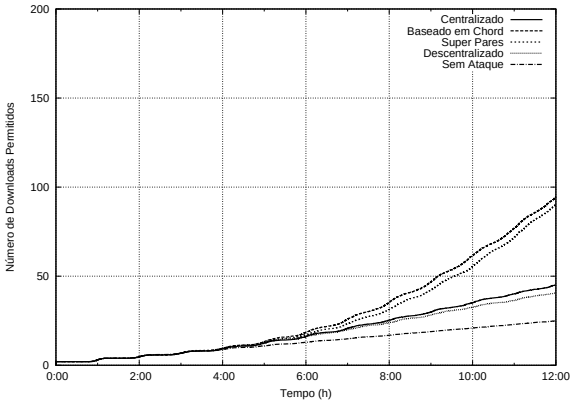

Figura 6. Downloads cumulativos de versão poluída em cenário com 500 pares corretos e 100 maliciosos

máquinas duais, gerenciadas por uma instância isolada do OurGrid, que processou durante aproximadamente 30 horas.

Primeiramente, os mecanismos distribuídos foram avaliados em cenários sem ataque de conluio. No caso de versão não poluída, todos os quatro mecanismos foram efetivos e causaram impacto negligível no espalhamento da versão. Mais especificamente, em um espaço pouco inferior a 6h foram autorizados todos os 500 downloads existentes. A variação entre os mecanismos foi consistentemente inferior a 20 minutos. No caso da versão poluída, os quatro mecanismos se mostraram eficientes na contenção do espalhamento da versão: após 12h de sessão, foram autorizados 24 downloads apenas. Note-se que estes downloads de versão poluída são um mal necessário, conforme explicado na Seção 3 pois é precisso manter o progresso do sistema, com novos downloads e consequentemente novos votos sendo emitidos. Os resultados correspondentes aos casos sem ataque, não poluído e poluído, são apresentados através de uma única curva "Sem Ataque" nas Figuras 5 e 6, respectivamente.

Nas Figuras 5 e 6, os eixo *x* e *y* representam, respectivamente, o tempo em horas e o número cumulativo de downloads permitidos. Note-se que as escalas no eixo *y* diferem. A análise é separada em dois casos: quando a versão é íntegra (sem poluição) e quando é não íntegra (poluída). Cada gráfico conta com cinco curvas, representando o número cumulativo de downloads autorizados para cada um dos quatro mecanismos distribuídos: Centralizado, Superpar, Baseado em Chord e Descentralizado, além de uma curva que representa os resultados obtidos quando não há ataque em conluio.

A Figura 5 ilustra o cenário em que uma versão não-poluída é disponibilizada, porém sua disseminação é dificultada devido à ação de pares maliciosos agindo em conluio e votando negativamente em relação à integridade da versão em questão. Inicialmente, pode-se notar que no caso sem ataque todos os pares terão realizado o download em um tempo aproximado de 6 horas, conforme já comentado. A presença de pares maliciosos implica um atraso na obtenção da autorização por parte de vários pares.

No cenário em que há controle centralizado, o mecanismo de contenção exerce um controle mais "estrito" sobre o sistema como um todo, pois um GD autoriza cada download ($E[D] = D$) e a estimativa $E[A]$ pode ser computada fielmente com base nos votos de todos os pares, na escala considerada (500 pares honestos e 100 maliciosos). Paradoxalmente, este caso reflete, ou sofre, mais fielmente o impacto dos votos falsos no ataque de conluio. Note-se que, após 6 horas, menos da metade dos pares terão recebido autorização para realizar o download. Outra constatação interessante é que os mecanismos Baseado em Chord e Super Pares apresentaram comportamento muito semelhante. Ambos os casos correspondem também ao cenário em que o ataque mostrou-se mais efetivo. Por último, o caso descentralizado foi o que sofreu o menor impacto: o tempo final para que todos os pares obtivessem autorização é bastante similar ao caso sem ataque, demonstrando a efetividade do mecanismo.

Já a Figura 6 apresenta os resultados do cenário em que a versão é poluída. Quando não há pares emitindo votos falsos (ataque de conluio), o mecanismo mostra-se extremamente eficaz em limitar o número de downloads autorizados. Conforme comentado anteriormente, mesmo após 12 horas, qualquer um dos mecanismos distribuídos concederá apenas 24 autorizações, fato que é representado com uma única curva. Tal curva cresce à proporção de A_{min}, ou seja, o menor limite possível a fim de não impedir o progresso do sistema. Novamente, os esquemas Baseado em Chord e Super Pares apresentaram comportamento similar e foram

os mais afetados pelo ataque. Ao término da simulação, haviam sido concedidas 94 e 90 autorizações, respectivamente. Esses valores são quase 4 vezes maiores do que o caso sem ataque, refletindo o impacto de votos de pares maliciosos. Os mecanismos Centralizado e Descentralizado apresentaram maior resiliência, tendo sido concedidas 45 e 41 autorizações, respectivamente.

6. Conclusões e trabalhos futuros

Sistemas de compartilhamento P2P são uma das aplicações distribuídas mais populares hoje em dia. Elas estão sujeitas, entretanto, ao problema da poluição de conteúdo, na qual uma versão falsa ou corrompida de um título é disseminada por pares maliciosos e em certos casos inadvertidamente por pares corretos. Para reduzir o impacto desses ataques, propomos anteriormente um mecanismo de controle de poluição que limita a taxa de espalhamento de uma versão de acordo com sua reputação. A reputação é mantida através de um sistema de gerência de reputação, em que pares fazem o download de uma versão de um conteúdo e então emitem um voto sobre a integridade da mesma.

Neste trabalho, o esquema de contenção foi aperfeiçoado, os mecanismos distribuídos implementados via simulação em nível de mensagem, e uma série de experimentos foram conduzidos de maneira a estudar o comportamento do esquema proposto. Foi mostrada a efetividade global dos mecanismos distribuídos propostos, tanto ao conter o espalhamento de versões poluídas, como ao causar sobrecarga mínima no caso de versões não poluídas.

Este trabalho pode ser estendido de diversas formas. Inicialmente, estamos expandindo os experimentos de simulação de maneira a considerar um leque mais amplo de cenários. Segundo, estudando formas de garantir com que os pares corretos emitam seu voto após realizar o download. Terceiro, modelando analiticamente o funcionamento do esquema com equações diferenciais, prevendo o comportamento global do sistema de contenção em cenários com milhões de pares.

Referências

[1] M. P. Barcellos, G. Facchini, H. H. Muhammad, G. B. Bedin, and P. Luft. Bridging the gap between simulation and experimental evaluation in computer networks. In *Simulation Symposium, 2006. 39th Annual*, pages 8 pp.+, 2006.

[2] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like p2p systems scalable. In *Proceedings of ACM SIGCOMM 2003*, Karlsruhe, Germany, August 2003.

[3] N. Christin, A. S. Weigend, and J. Chuang. Content availability, pollution and poisoning in file sharing peer-to-peer networks. In *6th ACM conference on Electronic commerce (EC '05)*, pages 68–77, New York, NY, USA, 2005. ACM Press.

[4] C. Costa, V. Soares, F. Benevenuto, M. Vasconcelos, J. Almeida, V. Almeida, and M. Mowbray. Disseminação de conteúdo poluído em redes p2p. In *XXIV Simpósio Brasileiro de Redes de Computadores*, Curitiba, PR, Brasil, May 2006.

[5] K. Eger and U. Killat. Bandwidth trading in unstructured p2p content distribution networks. In *6th IEEE International Conference on Peer-to-Peer Computing, 2006 (P2P 2006)*, pages 39–48, Washington, DC, USA, September 2006. IEEE Computer Society.

[6] A. Josang, R. Hayward, and S. Pope. Trust network analysis with subjective logic. In *ACSC '06: Proceedings of the 29th Australasian Computer Science Conference*, pages 85–94, Darlinghurst, Australia, 2006. Australian Computer Society, Inc.

[7] R. Kumar, D. Yao, A. Bagchi, K. W. Ross, and D. Rubenstein. Fluid modeling of pollution proliferation in p2p networks. In *ACM/IFIP SIGMETRICS/Performance 2006*, volume 34, pages 335–346, St. Malo, France, June 2006.

[8] U. Lee, M. Choiz, J. Choy, M. Y. Sanadidiy, and M. Gerla. Understanding pollution dynamics in P2P file sharing. In *5th International Workshop on Peer-to-Peer Systems (IPTPS'06)*, Santa Babara, CA, USA, February 2006.

[9] J. Liang, R. Kumar, Y. Xi, and K. W. Ross. Pollution in P2P file sharing systems. In *The 24th Conference on Computer Communications (INFOCOM 2005)*, volume 2, pages 1174–1185, Miami, FL, USA, March 2005.

[10] J. Liang, N. Naoumov, and K. W. Ross. Efficient blacklisting and pollution-level estimation in P2P file-sharing systems. In *ASIAN INTERNET ENGINEERING CONFERENCE (AINTEC)*, pages 1–21, Bangkok, Thailand, December 2005.

[11] J. Liang, N. Naoumov, and K. W. Ross. The index poisoning attack in p2p file-sharing systems. In *The 25th Conference on Computer Communications (INFOCOM 2006)*, Barcelona, Spain, April 2006.

[12] J. F. Silva, M. P. Barcellos, M. A. Konrath, L. P. Gaspary, and R. S. Antunes. Métodos para contenção de poluição de conteúdo em redes p2p. In *XXV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2007)*, volume 2, pages 855–868. SBC, May 2007.

[13] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, F. M. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking (TON)*, 11(1):17–32, February 2003.

[14] R. Thommes and M. Coates. Epidemiological modelling of peer-to-peer viruses and pollution. In *The 25th Conference on Computer Communications (INFOCOM 2006)*, pages 981–993, Barcelona, Spain, April 2006. IEEE.

[15] P. Veríssimo and L. Rodrigues. *Distributed Systems for System Architects*. Springer, Boston, USA, 1 edition, January 2001.

[16] K. Walsh and E. G. Sirer. Fighting peer-to-peer spam and decoys with object reputation. In *P2PECON '05: Proceeding of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems*, pages 138–143, New York, NY, USA, 2005. ACM Press.