Diagnóstico de Poluição de Conteúdo em Redes P2P para Transmissões de Mídia Contínua ao Vivo

Emanuel Amaral Schimidt
Universidade Federal do Paraná (UFPR)
emanuel@inf.ufpr.br

Roverli Pereira Ziwich
Universidade Federal do Paraná (UFPR)
roverli@inf.ufpr.br

Elias Procópio Duarte Jr.
Universidade Federal do Paraná (UFPR)
elias@inf.ufpr.br

Ingrid Jansch-Pôrto
Universidade Federal do Rio Grande do Sul (UFRGS)
ingrid@inf.ufrgs.br

RESUMO

A poluição de conteúdo é um dos desafios do uso de redes P2P para a transmissão de mídia contínua ao vivo. Como os próprios *peers* são responsáveis pela retransmissão dos dados, este não é um problema de solução trivial. Este trabalho apresenta uma nova solução para a detecção de poluição que utiliza o diagnóstico baseado em comparações para identificar alterações no conteúdo dos dados transmitidos. Cada *peer* do sistema executa comparações sobre determinados *chunks* de seus vizinhos. Com base no resultado das comparações, é possível detectar se há poluição de conteúdo e quem são os *peers* poluídos. A solução proposta foi implementada no Fireflies, um protocolo escalável para redes *overlay* tolerante a intrusões. Resultados experimentais mostram que esta estratégia é uma solução viável para a detecção de alterações de conteúdo e que a solução apresenta baixa sobrecarga no tráfego da rede.

ABSTRACT

Content pollution is one of the challenges for deploying live streaming with P2P networks in the Internet. As the peers themselves are responsible to retransmit data, there is no trivial solution to this problem. This work presents a new strategy to detect content pollution that employs comparison-based diagnosis to identify modifications on the data stream. A peer compares selected chunks with those of its neighbors. Based on the comparison results, peers that transmitted polluted content are identified. The proposed solution was implemented using Fireflies, a scalable and intrusion-tolerant overlay network. Experimental results show that the strategy represents a feasible solution to detect content pollution and adds a low overhead in terms of network bandwidth.

WebMedia'11: Proceedings of the 17th Brazilian Symposium on Multimedia and the Web. Full Papers.
October 3 -6, 2011, Florianópolis, SC, Brazil.
ISSN 2175-9642.
SBC - Brazilian Computer Society

Categories and Subject Descriptors

C.2 [**Computer-Communication Networks**]: General—*Security and protection*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*

General Terms

Algorithms, Security

Keywords

P2P, Comparison-Based Diagnosis, Live Streaming, Content Pollution

1. INTRODUÇÃO

Fortemente influenciados pelo Youtube, os vídeos na Internet se popularizaram nos últimos anos. Além de vídeos previamente disponíveis, a demanda por transmissões em larga escala de mídias contínuas ao vivo tem aumentado, mas ainda sem soluções definitivas. Dentre os impeditivos está o alto custo para manter ambientes computacionais e largura de banda grande o suficiente para atender uma enorme demanda. Em 2001, [4] já anunciava que no futuro, as transmissões de mídias ao vivo seriam responsáveis por grande parte do tráfego da Internet, e sugere que, para diminuir custos, fossem utilizadas redes P2P (*peer-to-peer*) para a difusão dos dados.

Nos últimos anos surgiram diversos sistemas que implementam a transmissão ao vivo de vídeos em redes P2P, como PPLive[1], SopCast[2], e PPStream[3]. Nestes sistemas a difusão de conteúdo é realizada por um servidor *fonte*. O conteúdo transmitido é dividido em pequenos pedaços, chamados *chunks*. O servidor fonte é responsável pela difusão inicial dos *chunks* na rede que, por sua vez, são compartilhados entre os usuários – os *peers*. Mas, os desafios desses sistemas são vários, dentre eles: o *churn*, ou seja, a frequência com que os *peers* entram e saem durante uma transmissão – o que pode prejudicar a qualidade da transmissão; e a existência de *peers* maliciosos, que podem causar problemas de diversas naturezas, entre eles a poluição de conteúdo. Os ataques de poluição são aqueles em que os participantes maliciosos

[1]PPLive - *http://www.pplive.com/en*
[2]SopCast - *http://www.sopcast.com*
[3]PPStream - *http://www.ppstream.com*

enviam dados adulterados, sejam alterações nos dados originais, criação de novos dados ou mesmo a inutilização e omissão dos mesmos [17]. Um agravante no problema da poluição dos dados em transmissões de mídia contínua ao vivo nas redes P2P é a restrição de tempo, já que detectar e solicitar novamente os dados poluídos ocasiona atrasos [20].

Algumas das soluções para combater a poluição de conteúdo em transmissão ao vivo nas redes P2P pressupõem que todos os participantes da rede ou sabem de antemão ou recebem durante a própria transmissão resumos digitais (*hash*) [19] dos *chunks*. Desta forma, todos os *peers* da rede P2P podem identificar qualquer modificação em um *chunk* através da geração do *hash* do *chunk* recebido e da comparação do valor gerado com o *hash* previamente recebido. Por outro lado, um *peer* malicioso poderia transmitir para seus vizinhos, *chunks* diferentes do original junto com os valores *hash* correspondentes, enganando os demais *peers* da rede, que eventualmente receberão estes *chunks* e *hashes* modificados.

Outras soluções ainda utilizam a geração de assinaturas digitais – ou seja, resumos digitais assinados utilizando criptografia de chave pública – dos *chunks* transmitidos [9]. Nesta estratégia as assinaturas digitais são geradas pelo servidor fonte e são transmitidas junto com os *chunks*. Desta forma, mesmo que um *peer* malicioso faça a transmissão de um *chunk* diferente do original para seus vizinhos, este *peer* malicioso não conseguirá forjar uma assinatura digital correspondente. Por outro lado, nesta estratégia todos os *peers* precisam realizar a verificação das assinaturas dos *chunks*, o que é um procedimento custoso.

Este trabalho apresenta uma alternativa para o diagnóstico de poluição de conteúdo em redes P2P para transmissões de mídia contínua ao vivo, que não utiliza criptografia de chave pública e que não pressupõe o envio prévio ou durante a transmissão dos *hashes* dos *chunks*. A solução proposta utiliza o diagnóstico baseado em comparações [7], para detectar alterações no conteúdo dos dados transmitidos. Cada *peer* do sistema executa comparações sobre determinados *chunks* de todos os seus vizinhos. Com base no resultado das comparações executadas por todos os *peers*, uma classificação de todos os *peers* em conjuntos é realizada com o objetivo de determinar se há poluição de dados.

A solução proposta foi implementada no Fireflies – um protocolo escalável para redes *overlay* tolerante a intrusões [10, 9]. O Fireflies utiliza a estratégia *pull-based* para a transmissão de dados e a topologia da rede é baseada em *mesh*. A implementação utilizou o mesmo simulador Fireflies construído e utilizado em [9]. Milhares de experimentos foram realizados com o objetivo de avaliar a sobrecarga que as comparações dos *chunks* acarretam sobre a rede. Os resultados mostram que a aplicação do diagnóstico baseado em comparação em redes P2P para o diagnóstico de poluição de conteúdo é uma solução viável e que impõe pequena sobrecarga ao tráfego da rede.

O restante deste trabalho está organizado da seguinte forma. Na seção 2 são abordados os conceitos de transmissões de mídia contínua ao vivo em redes P2P. A seção 3 apresenta a solução proposta, incluindo uma breve descrição sobre diagnóstico baseado em comparações e o protocolo Fireflies. Na seção 4 são apresentados os resultados experimentais realizados através de simulação. Na seção 5 uma descrição de trabalhos relacionados é apresentada. Por fim, seguem as conclusões.

2. TRANSMISSÕES DE MÍDIA CONTÍNUA AO VIVO EM REDES P2P

A transmissão de mídia contínua ao vivo em redes P2P ocorre através da difusão de conteúdo gerado por um servidor *fonte*. O conteúdo transmitido é dividido em pequenos pedaços, chamados *chunks*. Este servidor fonte é responsável pela inserção dos *chunks* na rede que, por sua vez, são compartilhados entre os usuários – os *peers* – que estão acompanhando a transmissão. Existem duas principais topologias utilizadas para realizar a transmissão de fluxo contínuo ao vivo em redes P2P [14]: topologia em *árvore* e em *mesh*. As principais diferenças das duas estratégias são apresentadas a seguir.

A principal vantagem do uso de árvores é que, após a árvore estar construída, as decisões de envio dos dados são simples: um *peer* recebe dados do seu pai e repassa para seus filhos [4]. Esta estratégia diminui o tempo de atraso (*delay*) entre o envio dos dados pelo servidor fonte e a chegada dos dados em todos os usuários do sistema. Entretanto, existem três principais desvantagens nesta topologia. (1) Qualquer perda de dados causada por um *peer* localizado perto da raiz da árvore afeta todos os filhos daquele *peer*. (2) Outro grande problema é a baixa resiliência ao *churn*: se um *peer* que não seja um nó folha deixa o sistema, todos os *peers* abaixo dele deixam de receber os dados até que a estrutura seja reconstruída [14]. E (3), a taxa média de *upload* é menor do que em outras topologias, uma vez que todos os *peers* que são nós folha apenas recebem dados e não participam da retransmissão.

A principal vantagem da topologia em *mesh* é que ela não é estruturada, ou seja, não se restringe a uma rígida estrutura de rede [16]. Nesta topologia, quando um *peer* deseja assistir à transmissão, ele simplesmente se conecta a uma lista de *peers*, e inicia uma troca de informações. O principal problema das redes *mesh* está relacionado com a forma em que as trocas dos dados são realizadas: para receber os dados, um *peer* deve requisitá-los; por sua vez, para requisitá-los, o *peer* deve possuir a lista de dados que cada um dos seus vizinhos possui. Portanto, nesta topologia há um maior consumo de banda, o que pode ocasionar atraso na propagação dos dados pelo sistema.

Sobre a trocas dos dados em si, existem três grandes estratégias [14]: *push-based*, *pull-based* e *push-pull-based*. A estratégia *push-based* é usada principalmente pelas árvores. Nesta estratégia os dados são enviados pelos *peers* emissores sem uma requisição explícita dos *peers* destinatários. Por outro lado, em um sistema estritamente *push-based* não existe como recuperar ou solicitar a retransmissão de um dado perdido ou cuja transmissão falhou. Ainda, se existirem múltiplos emissores de dados, um mesmo *peer* pode receber dados duplicados, o que consequentemente resulta em desperdício no uso de banda do sistema.

Na estratégia *pull-based*, os dados são enviados pelos *peers* apenas em resposta a uma solicitação. Assim, é possível que um *peer* faça um novo pedido de envio dos dados que não chegaram. Por outro lado cada *peer* deve possuir a lista dos dados que cada um dos seus vizinhos possui, o que também acarreta em um maior consumo de banda. Por este motivo podem ocorrer atrasos na propagação dos dados pelo sistema. Já na estratégia *push-pull-based*, existe a possibilidade da transmissão dos dados para um conjunto de vizinhos escolhidos previamente, além de permitir a requisição de dados

que não foram recebidos [14]. Uma forma de implementar esta estratégia utiliza o agendamento por um determinado tempo da transmissão de dados em modo *push-based*, além de possibilitar a requisição em modo *pull-based* [8].

3. DIAGNÓSTICO DE POLUIÇÃO DE CONTEÚDO PARA TRANSMISSÕES P2P AO VIVO

Esta seção apresenta a solução proposta para o diagnóstico de poluição de conteúdo para transmissões de mídia contínua ao vivo em redes P2P. Primeiramente esta seção descreve o protocolo Fireflies. Na sequência é apresentada uma breve descrição do diagnóstico baseado em comparações. Por fim, seguem detalhes da estratégia proposta para o diagnóstico de poluição.

3.1 O Protocolo Fireflies

O protocolo Fireflies é um protocolo que cria uma rede *overlay* tolerante a intrusões [10]. Todos os *peers* da rede P2P executam o protocolo Fireflies utilizando a estratégia *pull-based* para a transmissão de dados e a topologia da rede é baseada em *mesh*. O sistema é composto por um servidor *fonte* e os *peers* que são os próprios usuários do sistema. O servidor fonte é responsável pela geração e difusão dos *chunks*. Considera-se que o servidor fonte é confiável e nunca falha.

Os *chunks*, por sua vez, são enviados a partir do fonte para uma quantidade variável – e configurável – de *peers*. Os *peers* realizam o compartilhamento dos *chunks* entre si, de forma com que todos os *peers* possam obter cada um dos *chunks* gerados e disseminados pelo fonte. Os *peers* são organizados através de múltiplos anéis [10] – de número também configurável – onde cada anel contém todos os *peers*. Os *peers* do sistema recebem identificadores sequenciais. O protocolo também determina com quem cada um dos *peers* do sistema estará conectado, ou seja, quem são os vizinhos de cada *peer*. Como exemplo, a Figura 1 mostra um sistema de 9 *peers* configurados através de 3 anéis. Pode-se notar que, os vizinhos do *peer* 1 são os *peers* 2, 3, 4, 7 e 9. Considerando os diferentes anéis, um mesmo *peer* pode possuir vizinhos em comum, ou seja, cada *peer* sempre possui no mínimo dois vizinhos e no máximo $(2 * \lambda)$ vizinhos, onde λ é o número de anéis configurados no Fireflies. Este caso ocorre, no exemplo apresentado, com o *peer* 1, que possui o mesmo *peer* 3 como vizinho em dois diferentes anéis. No Fireflies o fonte recebe o identificador 0 mas não participa da configuração dos anéis. Por outro lado, existe uma configuração que define a quantidade de *peers* ao qual o fonte estará conectado.

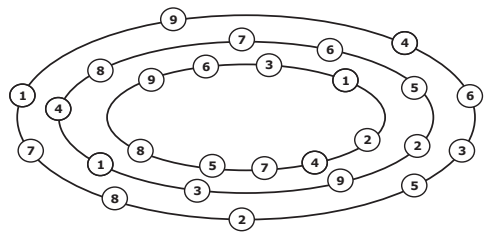

Figura 1: O Fireflies configurando um sistema de 9 unidades através de 3 anéis.

O protocolo Fireflies configura em cada um dos *peers* uma janela de disponibilidade que indica quais são os *chunks* que cada *peer* tem disponível para envio aos seus vizinhos. Além disso uma janela de interesse também é configurada e indica quais *chunks* cada *peer* precisa receber. Quando um *peer* recebe um determinado *chunk*, ele avisa todos os seus vizinhos que possui tal *chunk* disponível para envio. Desta forma cada *peer* mantém uma tabela com a relação de quais *chunks* cada um dos seus vizinhos informou como disponíveis. Em outras palavras, se um *peer* p souber que um dos seus vizinhos v possui o *chunk* c disponível para envio, se este *chunk* estiver na janela de interesse do *peer* p, este *peer* requisita o *chunk* c. Quando o *peer* v receber a requisição vinda do *peer* p, o *peer* v verifica se o *chunk* c ainda se encontra em sua janela de disponibilidade. Caso afirmativo, o *chunk* c é enviado ao *peer* p; caso contrário, a requisição é simplesmente ignorada. Este mesmo procedimento ocorre com os *chunks* gerados pelo fonte: sempre que o fonte gera e disponibiliza para envio um novo *chunk*, ele notifica seus vizinhos sobre a disponibilidade daquele *chunk* e assim começa a difusão pela rede.

3.2 Diagnóstico Baseado em Comparações

O diagnóstico baseado em comparações [7] determina o estado das unidades do sistema a partir da comparação dos resultados de tarefas produzidos por pares de unidades. Qualquer diferença na comparação indica que uma ou ambas as unidades estão falhas. O diagnóstico completo do sistema é baseado no resultado de todas as comparações. Ao conjunto de resultados de todas as comparações dá-se o nome de *síndrome* do sistema. O modelo MM, proposto por Maeng e Malek [15] para o diagnóstico de sistemas compostos de multiprocessadores homogêneos, é representado por um grafo $G = (V, E)$, onde V é o conjunto de unidades e E o conjunto de links de comunicação. Neste modelo são comparadas as saídas de uma mesma tarefa executada por pares de unidades. Assim, uma unidade k é comparadora de outras duas unidades i e j somente se $(k, i) \in E$ e $(k, j) \in E$, além do que $k \neq i$ e $k \neq j$.

Caso a comparação indique igualdade e, ainda, caso a unidade comparadora k não estiver falha, então as unidades i e j também não estão falhas. Mas se a comparação resultar em diferença, ao menos uma das unidades i, j ou k está falha. No modelo MM, o resultado gerado por duas unidades falhas para uma mesma tarefa é sempre diferente. Por outro lado, se a unidade comparadora k estiver falha, os resultados das comparações não são confiáveis, não permitindo que haja qualquer conclusão sobre o estado das unidades i e j. Após a comparação do resultado das tarefas, a unidade comparadora envia o resultado das comparações a um observador central, que por sua vez realiza o diagnóstico completo do sistema. Maeng e Malek também apresentam um caso especial do modelo MM, chamado MM*, na qual todas as unidades comparam todas as unidades vizinhas a que estão conectadas. Esta mesma característica de comparar todos os vizinhos foi empregada na solução proposta neste trabalho.

Em [22] um modelo generalizado de diagnóstico baseado em comparações é apresentado no qual as próprias unidades do sistema realizam o diagnóstico completo do sistema, ao invés de um observador central. Além disso, uma das principais diferenças deste modelo para outros é que a comparação do resultado de tarefas executadas por duas unidades falhas

pode resultar em igualdade.

3.3 A Estratégia Proposta para o Diagnóstico de Poluição

A estratégia proposta para o diagnóstico de poluição de conteúdo em redes P2P foi construída sobre o Fireflies. O sistema é composto por um servidor *fonte* e os *peers*. O servidor fonte é responsável pela geração e difusão dos *chunks*. Além do fonte e dos *peers*, a estratégia implementa dois novos componentes: o *módulo comparador* e o *tracker*, cujos papeis são descritos a seguir.

O *módulo comparador* é um componente que executa integrado aos próprios *peers* do sistema Fireflies. Este módulo é responsável por executar a comparação do conteúdo de determinados *chunks*, classificar todos os *peers* vizinhos em conjuntos de acordo com os resultados das comparações, e enviar esta classificação ao *tracker*. O *tracker* por sua vez, é uma entidade central confiável e que nunca falha acessível por todos os *peers*. Ele é responsável por receber as classificações enviadas pelos *peers* (através do módulo comparador), consolidá-las em uma única e nova classificação, e mais importante, realizar o diagnóstico do sistema, ou seja, determinar se há ou não poluição de dados, e quais são as unidades que estão com dados poluídos.

O módulo comparador é executado em cada *peer* i e faz a requisição de determinados *chunks* com identificador *cid* (*chunk identifier*) a todos os vizinhos do *peer* i. É importante destacar que toda requisição realizada pelo módulo comparador é direcionada ao próprio sistema Fireflies dos *peers* vizinhos, e o formato destas requisições é idêntico ao de qualquer requisição do sistema Fireflies. Em outras palavras, um *peer* que recebe uma requisição do módulo comparador – mesmo que seja um *peer* malicioso – não consegue distingui-las a ponto de tratá-las de forma diferenciada.

Os identificadores dos *chunks* (*cid*) que serão comparados são determinados pelo *tracker* e repassados ao módulo comparador. Assim que o *peer* i receber os *chunks* de identificador *cid* dos seus vizinhos, o módulo comparador classifica os *peers* em conjuntos $U_{i,cid}$. Um conjunto $U_{i,cid}$ contém o conteúdo de cada um dos diferentes *chunks* recebidos e o identificador dos *peers* que retornaram o *chunk* com aquele exato conteúdo, ou seja, $U_{i,cid} = \{(chunk_a, \{peer_i, peer_j, ...\}), (chunk_b, \{peer_k, peer_l, ...\}), ...\}$. Além disso, um subconjunto específico é criado para relacionar os *peers* vizinhos do *peer* i que não enviarem nenhuma informação sobre o *chunk cid*. Logo que o conjunto $U_{i,cid}$ estiver completo – ou seja, com informações de todos os *peers* vizinhos do i – este conjunto U é enviado ao *tracker*.

Uma otimização foi realizada apenas com o propósito de reduzir o tamanho da mensagem enviada pela rede: o conjunto $U_{i,cid}$ contém o valor *hash* ao invés do próprio conteúdo de cada um dos diferentes *chunks* recebidos. Desta forma, $U_{i,cid} = \{(hash\_chunk_a, \{peer_i, peer_j, ...\}), (hash\_chunk_b, \{peer_k, peer_l, ...\}), ...\}$. Vale lembrar que mesmo sem esta otimização, ou seja, mesmo sem o uso da função *hash*, o funcionamento do algoritmo continua o mesmo. Uma asserção é feita sobre o módulo comparador, na qual ele sempre classifica e troca mensagens de forma correta com o *tracker*. Para implementar esta asserção, pode-se utilizar uma abordagem similar à usada pelo HTTPS, na qual criptografia assimétrica é usada no início da sessão e em seguida uma chave secreta é estabelecida para a comunicação entre o *tracker* e o módulo comparador.

A Figura 2 mostra um exemplo do funcionamento do sistema. Neste exemplo considera-se que o *chunk* 13 é um dos *chunks* determinados pelo *tracker* para serem comparados. Não é mostrada na figura, mas a requisição do *chunk* 13 a partir dos *peers* 4 e 6 para todos os seus vizinhos já foi realizada. A figura mostra o envio do *chunk* 13 por todos os vizinhos dos *peers* 4 e 6. As arestas direcionadas representam o envio dos *chunks* que foram requisitados. As demais arestas (não direcionadas) representam os links de comunicação entre os *peers* ou o servidor fonte. Neste exemplo, são mostrados apenas os envios do *chunk* 13 pelos vizinhos dos *peers* 4 e 6, com o propósito de simplificar a figura, mas este mesmo procedimento ocorre com todos os *peers* do sistema.

Ainda com base no exemplo apresentado na Figura 2, considere que o *chunk* original de identificador 13 possui valor *hash* igual a AA, e uma versão poluída do mesmo *chunk* modificada indevidamente pelo *peer* 5 possui o *hash* AB. Considerando este caso, os conjuntos $U_{i,cid}$ após a classificação realizada pelos *peers* 4 e 6 são: $U_{4,13} = \{(AA, \{1, 3, 4, 5, 8\})\}$ e $U_{6,13} = \{(AA, \{2, 6, 7, 9\}), (AB, \{5\})\}$. Os próprios *peers* 4 e 6 se incluem nos conjuntos $U_{i,cid}$, sendo inseridos no grupo correspondente ao *chunk* que possuem.

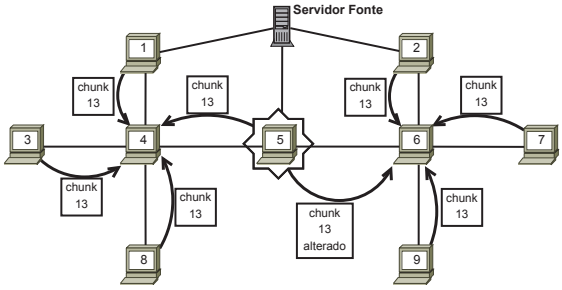

Figura 2: Envio do *chunk* 13 para os *peers* 4 e 6 por cada um dos seus vizinhos. O *peer* 5 é malicioso.

Como o *tracker* recebe de cada *peer* i o conjunto $U_{i,cid}$, o *tracker* terá condições de realizar o diagnóstico completo do sistema e identificar quais unidades estavam com conteúdo poluído. A partir de todos estes conjuntos $U_{i,cid}$, o *tracker* realiza uma nova e única classificação de todos os *peers*, agora em um novo conjunto T_{cid}, que por sua vez tem o mesmo formato do conjunto $U_{i,cid}$.

Neste conjunto T_{cid}, diferentemente dos conjuntos $U_{i,cid}$, um determinado *peer* poderá estar associado a mais de um subconjunto. Um exemplo desta situação também pode ser notada na Figura 2, na qual o *peer* 5 enviou *chunks* de valores *hash* diferentes para os seus *peers* vizinhos 4 e 6. Neste caso o *tracker* irá incluir o *peer* 5 em dois subconjuntos diferentes: no conjunto indicado por AA e no conjunto indicado por AB.

Como o fonte é confiável e nunca envia diferentes versões de um mesmo *chunk*, neste conjunto T_{cid} o fonte estará sempre vinculado a um único subconjunto. Para realizar o diagnóstico considera-se como falhos, ou seja, com conteúdo diferente do considerado correto, todos os *peers* que estiverem em mais de um subconjunto e também os *peers* que não estiverem no mesmo subconjunto ao qual o fonte pertence. A Figura 3 ilustra o envio dos conjuntos $U_{i,13}$ pelos *peers* 4 e 6, e o Conjunto T com a classificação final realizada pelo *tracker* para o *chunk* 13 é $T_{13} = \{(AA, \{fonte, 1, 2, 3, 4, 5, 6, 7, 8, 9\}), (AB, \{5\})\}$,

obtido através da junção dos conjuntos $U_{4,13}$ e $U_{6,13}$. Este conjunto T_{13} ainda é parcial, pois o *tracker* continua aguardando os conjuntos $U_{i,13}$ dos demais *peers* do sistema.

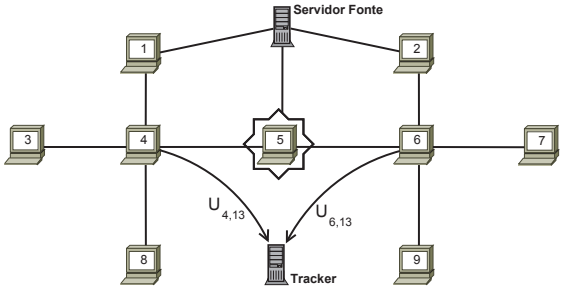

Figura 3: Envio para o *tracker* dos conjuntos $U_{i,cid}$ gerados pelos *peers* 4 e 6 para o *chunk* 13.

Além disso, como o módulo comparador executa continuamente, é possível que o *tracker* ainda esteja recebendo por parte de alguns *peers* informações de um determinado *chunk* cid_a enquanto que outros *peers* já estejam enviando informações de outro *chunk* cid_b. Por este motivo, o *tracker* mantém separada e concorrentemente a classificação dos conjuntos T_{cid_a} e T_{cid_b}.

A Figura 4 apresenta em pseudo-código o algoritmo executado pelo módulo comparador. A primeira tarefa executada por este algoritmo, na linha 2, é obter a lista dos *chunks* que serão comparados. Esta informação é obtida através de uma requisição ao *tracker*, que, a cada intervalo de tempo, escolhe aleatoriamente uma lista de *chunks* para ser utilizada como base para as comparações. A partir da lista dos *chunks* que devem ser comparados, o módulo comparador permanece, a todo instante, esperando a informação de que algum de seus vizinhos possui um novo *chunk* disponível. O bloco iniciado na linha 4 é executado sempre que algum vizinho v possui um novo *chunk* disponível. Caso o *cid* do novo *chunk* disponível esteja na lista dos *chunks* que devem ser comparados, é verificado se o *timer* daquele *cid* foi inicializado (linha 6). Este *timer* será usado como tempo limite para que o *peer i* realize todas as comparações e a classificação referente ao *chunk cid*. Se este for o primeiro dos vizinhos do *peer i* que esteja disponibilizando um *chunk cid*, o *timer* correspondente ao *chunk cid* é iniciado (linha 7). Na linha 9, ocorre a requisição do *chunk cid* para o *peer v*, e na sequência ocorre a atualização do conjunto $U_{i,cid}$ classificando o *peer v* de acordo com o conteúdo do *chunk cid* recebido.

O bloco iniciado na linha 14 verifica se o conjunto $U_{i,cid}$ já possui todos os vizinhos do *peer i* ou se o tempo limite para que os vizinhos enviassem informações sobre aquele *chunk* terminou. Em ambos os casos, o conjunto $U_{i,cid}$ é enviado ao *tracker* (linha 18). No entanto, caso ocorra a existência de vizinhos do *peer i* que ainda não enviaram informações sobre o *chunk cid*, estes vizinhos são classificados em um subconjunto do conjunto $U_{i,cid}$ específico para este propósito (linha 16). O tempo limite de resposta ($limite\_resposta$) é um valor relacionado ao tamanho da janela de disponibilidade dos *peers* que também considera a frequência de geração de novos *chunks* do fonte.

Por sua vez o *tracker* fica continuamente recebendo os conjuntos $U_{i,cid}$ referentes à classificação realizada pelo *peer i* para o *chunk cid*. Esta ação é mostrada na linha 2 do algoritmo *Diagnostico* mostrado na Figura 5. Toda vez que recebe um conjunto $U_{i,cid}$, o *tracker* classifica os *peers* contidos nos subconjuntos daquele conjunto $U_{i,cid}$ em um novo conjunto T_{cid} (linhas 4–8). Após terminar a classificação para todos os *peers* do sistema ou caso o tempo limite para que os *peers* enviassem os seus respectivos conjuntos $U_{i,cid}$ tenha terminado (linha 12), o *tracker* imprime o diagnóstico do sistema (linha 16). Considerando o conjunto T_{cid}, o *tracker* irá considerar como *peers* que possuem conteúdo diferente do considerado correto, todos os *peers* que estiverem em mais de um subconjunto e também os *peers* que não estiverem no mesmo subconjunto ao qual o fonte pertence. Neste algoritmo, se ocorrer o mesmo caso onde um determinado *peer i* não envie o conjunto $U_{i,cid}$ dentro do tempo limite de resposta, este *peer i* é classificado em um subconjunto específico do conjunto T_{cid} (linhas 13 e 14).

4. RESULTADOS EXPERIMENTAIS

A solução apresentada para o diagnóstico de poluição de conteúdo foi implementada no Fireflies utilizando o mesmo simulador dirigido a eventos construído em [9]. Todas as simulações foram executadas construindo uma rede de 200 *peers*. Cada uma das simulações de transmissão de mídia foi executada durante 200 segundos e o servidor fonte sempre gerou os *chunks* em uma frequência de 30 *chunks*/s. Foram utilizados *chunks* de tamanho de 10 KB. A janela de disponibilidade e a janela de interesse de todos os *peers* foram configuradas com o valor de 3000 *chunks*. Além disso, o Fireflies foi configurado para organizar os *peers* em três anéis, portanto cada *peer* possui no mínimo dois e no máximo seis vizinhos.

Foram executados um total de 2.400 experimentos, com dois principais objetivos: (a) verificar se o diagnóstico identifica corretamente os *peers* que possuem conteúdo poluído; e (b) verificar a sobrecarga adicionada à rede pela solução de comparações em termos da quantidade adicional de *chunks* enviados. Em todos os experimentos, o sistema foi configurado de forma a monitorar *chunks* a cada 15 segundos, isto é, as simulações imprimem um novo diagnóstico sobre a poluição no sistema a cada 15 segundos.

Todos os experimentos foram executados através de 24 diferentes configurações variando três parâmetros: a existência ou não de *churn*; a quantidade de *peers* maliciosos, variando entre 0%, 5%, 10%, 15%, 20% e 25%; e o tipo dos *peers* maliciosos – foram experimentados *peers* maliciosos que sempre alteram o conteúdo e *peers* maliciosos que alteram o conteúdo de forma aleatória (50% de chance de alteração). Para os experimentos onde foi utilizado *churn*, 100 *peers* entraram e 100 *peers* foram removidos da rede seguindo distribuições probabilísticas [21]. Uma distribuição normal é empregada – com média 100 e desvio padrão 20 – para determinar as entradas de *peers* na rede, e uma distribuição de Poisson – com média 100 – é utilizada para determinar a frequência de saída de *peers* da rede.

Para cada uma das 24 diferentes configurações foram executadas 100 simulações, e os resultados destes grupos de 100 simulações estão sumarizados e são apresentados nos gráficos das figuras a seguir. As linhas dos gráficos, representam os valores médios dos resultados destes grupos de 100 simulações, enquanto as linhas pontilhadas de intervalo indicam os respectivos valores mínimo e máximo encontrados.

A Figura 6 mostra o número de *chunks* enviados pela rede pelo protocolo Fireflies. Nesta figura, duas linhas represen-

```
Algoritmo: ModuloComparador
1: início
2:     lista_de_cids ← obter do tracker lista de chunks a serem comparados
3:
4:     sempre que um vizinho v disponibiliza um novo chunk cid
5:       se cid ∈ lista_de_cids então
6:         se timer_cid não foi inicializado então
7:           inicializar timer_cid
8:         fim se
9:         obter o chunk cid de v
10:        atualizar U_{i,cid}
11:      fim se
12:    fim sempre que
13:
14:    sempre que (U_{i,cid} possui dados de todos os vizinhos) ou (timer_cid > limite_resposta)
15:      se timer_cid > limite_resposta então
16:        incluir os peers vizinhos que não responderam em conjunto específico de U_{i,cid}
17:      fim se
18:      enviar U_{i,cid} ao tracker
19:      lista_de_cids ← obter do tracker e atualizar lista de chunks a serem comparados
20:    fim sempre que
21: fim
```

Figura 4: Algoritmo em pseudo-código do módulo comparador executado em todos os *peers* do sistema.

```
Algoritmo: Diagnostico
1: início
2:     para todo conjunto U_{i,cid} recebido
3:       para todo subconjunto u de U_{i,cid}
4:         se ∃ hash_chunk_u ∈ T_{cid} então
5:           inserir os peers associados ao hash_chunk_u no grupo correspondente em T_{cid}
6:         senão
7:           criar novo subgrupo em T_{cid} com o subconjunto u = (hash_chunk_u, {lista de peers})
8:         fim se
9:       fim para
10:    fim para
11:
12:    sempre que (T_{cid} possui dados de todos os peers) ou (timer_cid > limite_resposta)
13:      se timer_cid > limite_resposta então
14:        incluir os peers que não responderam em conjunto específico de T_{cid}
15:      fim se
16:      imprime diagnóstico referente às comparações do chunk cid com base no conjunto T_{cid}
17:    fim sempre que
18: fim
```

Figura 5: Algoritmo de diagnóstico executado pelo *tracker*.

tam os experimentos executados com *churn* e as outras duas representam os experimentos sem *churn*. O número médio de *chunks* enviados pelo Fireflies para todas as sessões foi sempre por volta de 1,2 milhões de *chunks*. Além disso nota-se também que os experimentos sem *churn* possuem uma variação muito pequena no número de *chunks* enviados. Em todas as figuras, as linhas identificadas "sem churn" e "com churn" (*labels* sem asterisco) referem-se aos experimentos nos quais os *peers* maliciosos sempre alteram o conteúdo transmitido. Já as duas linhas identificadas com "sem churn*" e "com churn*" (*labels* com asterisco) referem-se aos experimentos onde os *peers* maliciosos alteram o conteúdo transmitido de forma aleatória.

Na Figura 7 é mostrada a quantidade média de *chunks* adicionais requisitados pelos módulos comparadores em cada experimento. Nota-se que esta quantidade, para todos os experimentos – incluindo os experimentos com *churn* e com *peers* maliciosos – esteve entre 18.000 e 22.000 *chunks*. Portanto, a solução proposta adiciona cerca de 1,8% de sobrecarga ao tráfego da rede. Vale ressaltar que, dependendo da largura de banda disponível na rede, a frequência com que os *chunks* são monitorados pode ser aumentada, já que nos experimentos a frequência utilizada foi de 15 segundos. O mesmo vale para o caso contrário, onde esta frequência pode ser diminuída caso exista grande restrição em relação à largura de banda disponível.

A Figura 8 mostra a média do número de *peers* que receberam dados poluídos, considerando os *chunks* que foram monitorados. Nesta figura nota-se que, mesmo com apenas 5% de *peers* maliciosos, o número médio de *peers* que possuíram *chunks* poluídos em um grupo de experimentos com *churn* chegou a 45, o que equivale à 22% dos *peers*. Já com 25% de *peers* maliciosos, o número médio de *peers* que receberam dados poluídos chegou a 142, o que equivale à 71% dos *peers* da rede. Nota-se ainda que em alguns casos específicos, também na presença de 25% de *peers* maliciosos, o número máximo de *peers* que receberam dados poluídos chegou a 166 – 83% dos *peers*. A porcentagem dos *peers* que receberam *chunks* poluídos e que foram diagnosticados corretamente pela solução proposta foi de 100% em todos os experimentos.

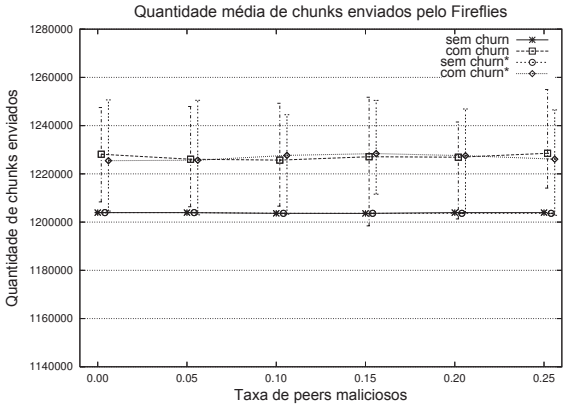

Figura 6: Número de *chunks* enviados pelo Fireflies.

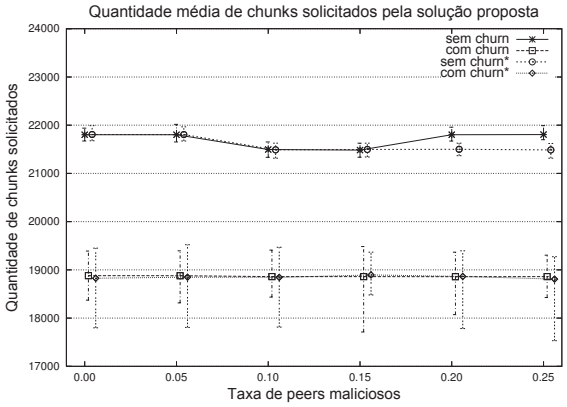

Figura 7: Número de *chunks* requisitados especificamente pelo módulo comparador.

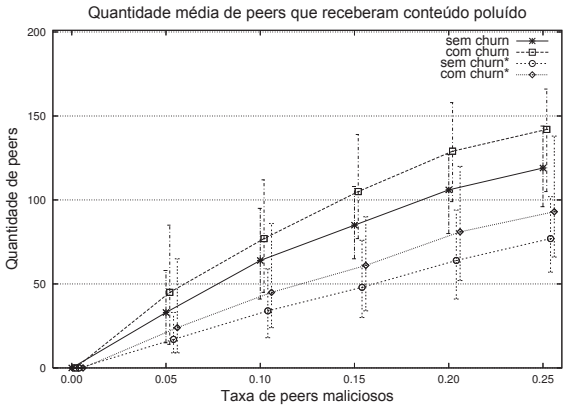

Figura 8: *Peers* que receberam *chunks* poluídos.

5. TRABALHOS RELACIONADOS

Para combater os ataques de poluição de conteúdo – ou *poisoning* [3] – em redes de compartilhamento P2P, foram criadas diversas técnicas. No entanto, quando aplicadas para transmissões de mídia contínua ao vivo aumentam a sobrecarga de dados na rede, o que pode ocasionar atraso na transmissão e desperdício de banda [17].

Na técnica de lista negra [12] os *peers* poluidores são incluídos em uma lista, com base nos seus respectivos endereços IP na rede. A lista é criada para conter as faixas de endereços que englobam os poluidores, mas de forma a incluir o menor número possível de *peers* não poluidores. Nesta estratégia, os demais *peers* do sistema não enviam nem recebem dados de *peers* que estão na lista negra. Por outro lado, quando aplicada para transmissões de mídia contínua ao vivo, a técnica se mostra custosa [17]. Além disso, esta técnica pode ser comprometida se um *peer* malicioso falsificar seu endereço na rede. Na técnica de assinaturas baseadas em *hash* [19] cada *chunk* tem um valor *hash* pré-calculado e esse valor *hash* é difundido por toda a rede. Esta técnica é eficaz para erros gerados durante a transmissão dos *chunks*, sendo empregada pelo BitTorrent [5].

Outras soluções utilizam a assinatura digital dos *chunks* realizada pelo servidor fonte e a transmissão dessa assinatura juntamente com o *chunk* [9]. Destaca-se que a verificação das assinaturas digitais para cada um dos *chunks* é um processo considerado custoso e pode ser um impeditivo em casos de transmissões ao vivo, por exemplo, por dispositivos móveis com recursos limitados. Uma melhoria nesta técnica foi proposta também em [9] – chamada de *Linear Digests* – onde o *hash* de vários *chunks* são agrupados e uma única assinatura digital é gerada pelo servidor fonte para todo este grupo de *chunks*. Desta forma o impacto da verificação da assinatura digital nos *peers* é reduzido.

Algumas outras ferramentas, como por exemplo em [11], utilizam ainda a criptografia completa de todos os dados transmitidos na rede através do estabelecimento de uma chave privada compartilhada. Já em [2] é apresentada uma solução que utiliza um grupo responsável por manter a integridade do conteúdo transmitido pelo fonte. Nesta solução cada *peer* que requisita e recebe um dado pela rede, verifica a integridade do dado através deste grupo.

Os autores de [18] apresentam *Credence*, um sistema P2P descentralizado aplicado para compartilhamento de arquivos baseado em reputação e *ranking*. Neste sistema, os próprios *peers* endossam como honestos outros *peers* do sistema, que consequentemente podem acessar o conteúdo compartilhado. Em [1] os autores apresentam outras duas soluções baseadas em reputação, mas diferentemente de [18], estas soluções são aplicadas para a transmissão de mídia contínua ao vivo.

Outras técnicas são encontradas como formas alternativas para diminuir o custo de autenticação em transmissões ao vivo. Uma delas é a *Merkle-Tree* [19] onde o servidor fonte calcula os valores *hash* de um número limitado de *n chunks* consecutivos. Estes valores *hash* são usados como as folhas de uma árvore *Merkle* e os nós intermediários são identificados pelos *hashes* dos seus nós filhos. Os *hashes* de todos os nós nesta estrutura de árvore são combinados para realizar a autenticação de cada *chunk*.

Em [6] os autores analisam quatro estratégias já listadas: lista negra, criptografia, verificação de *hash* e assinaturas digitais. O trabalho conclui que o uso de árvores *Merkle* é um dos mais eficientes em termos de *overhead* computacional. Mais recentemente, os autores de [13] conduzem um estudo sobre o impacto de ataques de poluição de conteúdo, e mostram que o impacto e a eficiência dos ataques não são diretamente relacionados ao tamanho da rede em si, mas dependem fortemente da estabilidade da rede e da largura de banda disponível pelos *peers* maliciosos e pelo fonte.

6. CONCLUSÃO

Este trabalho apresentou uma nova estratégia para o diagnóstico de poluição de conteúdo em redes P2P para transmissões de mídia contínua ao vivo. A solução proposta utilizou o diagnóstico baseado em comparações para detectar alterações no conteúdo dos dados disseminados pela rede. Cada *peer* do sistema executa comparações sobre determinados *chunks* de todos seus vizinhos. O diagnóstico é realizado com base no resultado de todas as comparações, ou seja, através de todas as comparações é possível identificar os *peers* que possuem dados poluídos. A solução proposta foi implementada no Fireflies, um protocolo escalável para redes *overlay*. Um grande número de experimentos foram realizados através de simulação e mostraram que a estratégia proposta é uma solução viável para a detecção de poluição de conteúdo em transmissões de mídia contínua ao vivo. Além disso, nos experimentos realizados, a solução adicionou uma sobrecarga de apenas 1,8% ao tráfego da rede.

Em trabalhos futuros pretende-se incluir estratégias para detecção de *peers* que possuam falhas de outras naturezas, além da implementação de uma solução prática para transmissões de mídia ao vivo na Internet.

7. REFERÊNCIAS

[1] A. Borges, J. Almeida, and S. Campos. Fighting Pollution in P2P Live Streaming Systems. *IEEE Intl. Conf. on Multimedia and Expo (ICME'08)*, pages 481–484, 2008.

[2] R. Chen, E. K. Lua, J. Crowcroft, W. Guo, L. Tang, and Z. Chen. Securing Peer-to-Peer Content Sharing Service from Poisoning Attacks. *Proc. of the 8th IEEE Intl. Conf. on Peer-to-Peer Computing (P2P'08)*, pages 22–29, 2008.

[3] N. Christin, A. S. Weigend, and J. Chuang. Content Availability, Pollution and Poisoning in File Sharing Peer-to-Peer Networks. *Proc. of the 6th ACM Conf. on Electronic Commerce (EC'05)*, pages 68–77, 2005.

[4] H. Deshpande, M. Bawa, and H. Garcia-Molina. Streaming Live Media over a Peer-to-Peer Network. *Technical Report, Stanford InfoLab*, (2001-30), 2001.

[5] P. Dhungel, X. Hei, K. W. Ross, and N. Saxena. The Pollution Attack in P2P Live Video Streaming: Measurement Results and Defenses. *Proc. of the Workshop on Peer-to-peer Streaming and IP-TV (P2P-TV'07)*, pages 323–328, 2007.

[6] P. Dhungel, X. Hei, K. W. Ross, and N. Saxena. Pollution in P2P Live Video Streaming. *Intl. Journal of Computer Networks and Communications (IJCNC'09)*, 1(2), 2009.

[7] E. P. Duarte Jr., R. P. Ziwich, and L. C. P. Albini. A Survey of Comparison-Based System-Level Diagnosis. *ACM Computing Surveys (CSUR)*, 43(3):22:1–22:56, 2011.

[8] V. Fodor and G. Dan. Resilience in Live Peer-to-peer Streaming. *IEEE Communications Magazine*, 45(6), 2007.

[9] M. Haridasan and R. van Renesse. Defense against Intrusion in a Live Streaming Multicast System. *6th IEEE Intl. Conf. on Peer-to-Peer Computing (P2P'2006)*, pages 185–192, 2006.

[10] H. Johansen, A. Allavena, and R. van Renesse. Fireflies: Scalable Support for Intrusion-Tolerant Network Overlays. *Proc. of the 1st ACM EuroSys.*, C-25, 2006.

[11] J. Liang, R. Kumar, and K. W. Ross. The FastTrack Overlay: A Measurement Study. *Computer Networks*, 2006.

[12] J. Liang, N. Naoumov, and K. W. Ross. Efficient Blacklisting and Pollution-Level Eestimation in P2P File-Sharing Systems. *Asian Internet Engineering Conference*, pages 173–175, 2005.

[13] E. Lin, D. M. N. de Castro, M. Wang, and J. Aycock. SPoIM: A close Look at Pollution Attacks in P2P Live Streaming. *Proc. of the 18th Intl. Workshop on Quality of Service (IWQoS'10)*, pages 1–9, 2010.

[14] T. Loocher, R. Meier, S. Schmid, and R. Wattenhofer. Push-to-Pull Peer-to-Peer Live Streaming. *21st Intl. Symp. on Distributed Computing (DISC'07)*, pages 388–402, 2007.

[15] J. Maeng and M. Malek. A Comparison Connection Assignment for Self-Diagnosis of Multiprocessor Systems. *Proc. of the 11th IEEE Fault-Tolerant Computing Symp.*, pages 173–175, 1981.

[16] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, A. E. Mohr, and E. E. Mohr. Chainsaw: Eliminating Trees from Overlay Multicast. *Proc. of the 4th Intl. Workshop on Peer-To-Peer Systems (IPTPS'05)*, pages 127–140, 2005.

[17] A. B. Vieira. Transmissão de Mídia Contínua ao Vivo em P2P: Modelagem, Caracterização e Implementação de Mecanismos de Resiliência a Ataques. *Tese de Doutorado, Universidade Federal de Minas Gerais (UFMG)*, 2010.

[18] K. Walsh and E. G. Sirer. Experience with an Object Reputation System for Peer-to-Peer Filesharing. *Proc. of the 3rd USENIX Symp. on Networked Systems Design and Implementation (NSDI'06)*, 3, 2006.

[19] C. K. Wong and S. S. Lam. Digital Signatures for Flows and Multicasts. *IEEE/ACM Trans. on Networking*, 7(4):502–513, 1999.

[20] S. Yang, H. Jin, B. Li, X. Liao, H. Yao, and X. Tu. The Content Pollution in Peer-to-Peer Live Streaming Systems: Analysis and Implications. *Proc. of the 37th Intl. Conf. on Parallel Processing (ICPP'08)*, pages 652–659, 2008.

[21] Z. Yao, D. Leonard, X. Wang, and D. Loguinov. Modeling Heterogeneous User Churn and Local Resilience of Unstructured P2P Networks. *Proc. of the 14th IEEE Intl. Conf. on Network Protocols (ICNP'06)*, pages 32–41, 2006.

[22] R. P. Ziwich, E. P. Duarte Jr., and L. C. P. Albini. Distributed Integrity Checking for System with Replicated Data. *Proc. of the 11th IEEE Intl. Conf. on Parallel and Distributed Systems*, pages 363–369, 2005.