Uma Nova Estratégia Completamente Distribuída para Combate à Poluição de Conteúdo em Transmissões ao Vivo

Roverli P. Ziwich[1], **Glaucio P. Silveira**[2], **Elias P. Duarte Jr.**[1]

Universidade Federal do Paraná (UFPR), Curitiba, PR, Brasil
[1]Dept. de Informática, P.O.Box 19018, CEP 81531-980
[2]Centro de Computação Eletrônica, P.O.Box 19037, CEP 81531-980

roverli@inf.ufpr.br, glaucio@ufpr.br, elias@inf.ufpr.br

Resumo. *É notável o crescente uso da Internet para transmissões ao vivo. Por outro lado a poluição de conteúdo nas transmissões ao vivo em redes P2P continua sendo um desafio, já que soluções incorrem necessariamente em sobrecarga de processamento, de uso de rede, ou até atrasos na transmissão. Este trabalho apresenta uma nova estratégia distribuída e descentralizada com o objetivo de combater a propagação de conteúdo poluído em transmissões ao vivo. Para impedir a propagação da poluição, cada peer do sistema executa comparações periódicas sobre determinados chunks de seus vizinhos. Com base nos resultados das comparações, cada peer, de forma independente dos demais, deixa de solicitar chunks aos seus vizinhos considerados poluidores. A solução proposta foi implementada no Fireflies, um protocolo escalável para redes overlay. Resultados experimentais mostram que esta estratégia adiciona baixa sobrecarga no tráfego da rede, e é uma solução viável para tratar a poluição de conteúdo em transmissões ao vivo: em vários casos a solução foi capaz de eliminar a poluição no decorrer das transmissões.*

Abstract. *Live streaming transmissions are becoming increasingly frequent in the Internet. One of the main challenges of those transmissions is to detect and prevent content pollution. This work presents a novel strategy to combat content pollution in P2P-based live streaming that is fully distributed. In order to prevent the propagation of polluted content, each peer independently compares and classifies its neighbors, and stops requesting chunks from those neighbors identified as polluters. The proposed solution was implemented using Fireflies, a scalable overlay network protocol. Experimental results evaluate the impact of the proposed strategy in terms of the network bandwidth overhead, and show that the strategy is an effective solution to combat content pollution in realistic scenarios of live streaming transmissions.*

1. Introdução

Transmissões ao vivo, notadamente de vídeos, estão se tornando cada vez mais populares na Internet [Lin et al. 2010] e diversos sistemas para transmissões ao vivo que utilizam redes P2P surgiram nos últimos anos – como por exemplo o PPLive[1], o SopCast[2], e o

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

PPStream[3]. As redes P2P possuem algumas vantagens sobre o formato tradicional cliente-servidor para transmissões ao vivo pois os próprios *peers* podem compartilhar o conteúdo que é transmitido. Desta forma a quantidade, capacidade de processamento e de largura de banda dos servidores que distribuem o conteúdo transmitido pode ser significativamente menor do que a dos mesmos servidores nas redes que utilizam o formato tradicional.

Por outro lado, como os próprios *peers* são responsáveis pelo conteúdo transmitido, a poluição de conteúdo nas transmissões ao vivo em redes P2P é um dos desafios que continuam relevantes. Um ataque de poluição de conteúdo consiste na modificação não autorizada do conteúdo transmitido – que é dividido em pedaços, chamados *chunks*. A modificação dos *chunks* pode ser de diferentes tipos [Gheorghe et al. 2010, Dhungel et al. 2009, Lin et al. 2010], que incluem: a troca de conteúdo; a criação de novos dados; e até a destruição ou omissão dos *chunks* transmitidos.

Outras características que agravam o problema da poluição de conteúdo nas transmissões ao vivo incluem o limite de tempo no qual o conteúdo transmitido tem para alcançar os *peers* da rede, e o *churn*, isto é, o fato de *peers* entrarem e saírem da rede continuamente durante a transmissão. Estas características são relevantes pois a detecção de conteúdo poluído e a consequente criação de novas solicitações pode causar atrasos e até saltos na transmissão assistida pelos usuário [Yang et al. 2008, Zhang and Helvik 2011].

Algumas soluções que tratam o problema da poluição de conteúdo em transmissões ao vivo assumem que todos os *peers* sabem previamente, ou recebem durante a própria transmissão o valor *hash* dos respectivos *chunks* [Wong and Lam 1999]. Esta estratégia é bastante usada para tratar falhas físicas nos canais de comunicação, mas ainda permite a um *peer* malicioso modificar indevidamente um *chunk* e retransmiti-lo juntamente com um novo valor *hash* correspondente. Outras soluções ainda propõem o uso de assinaturas digitais, ou seja, criptografia de chave pública, para todos os *chunks* que são transmitidos [Haridasan and van Renesse 2006]. A assinatura digital é gerada pelo servidor fonte e transmitida juntamente com os *chunks* pela rede. Nesta estratégia, cada *peer* que recebe um *chunk* deve conferir se a assinatura digital é válida. Por outro lado este é um procedimento que pode ser considerado computacionalmente custoso, dependendo dos dispositivos usados pelos usuários da transmissão.

Recentemente, em [Schimidt et al. 2011], uma solução alternativa é apresentada, que utiliza o diagnóstico baseado em comparações para detectar poluição de conteúdo em transmissões ao vivo em redes P2P. Esta solução não utiliza criptografia de chave pública e não utiliza o envio de valores *hash* junto à transmissão. Por outro lado, a solução apresentada assume a existência de uma unidade central – um *tracker* – que é a unidade que realiza o diagnóstico da poluição na rede. Este *tracker* central, por sua vez, pode representar um problema de escalabilidade, caso a solução seja utilizada em redes extremamente grandes. Além disso a solução atua apenas na identificação da existência de poluição na rede, sem combatê-la.

Este trabalho apresenta uma nova estratégia que também utiliza o diagnóstico baseado em comparações e que também não utiliza criptografia de chave pública e não pressupõe o envio de valores *hash* juntamente com a transmissão dos *chunks*. Por outro lado, a nova estratégia proposta neste trabalho, tem por objetivo o combate à propagação

[3]PPStream - *http://www.ppstream.com*

de conteúdo poluído nas transmissões ao vivo em redes *overlay*. Outra contribuição deste trabalho é que a estratégia proposta, diferente da anterior [Schimidt et al. 2011], é completamente distribuída e descentralizada, ou seja, esta solução não utiliza um *tracker* central. Para combater a propagação da poluição, cada *peer* do sistema executa comparações periódicas sobre determinados *chunks* de seus vizinhos. Com base nos resultados das comparações, cada *peer*, de forma independente dos demais, deixa de solicitar *chunks* aos seus vizinhos considerados poluidores.

A solução proposta foi implementada no Fireflies, um protocolo escalável para redes *overlay* [Johansen et al. 2006]. O Fireflies usa a estratégia *pull-based* para a transmissão dos dados e emprega a topologia *mesh*. A implementação foi realizada usando o mesmo simulador Fireflies descrito em [Haridasan and van Renesse 2006]. Resultados experimentais mostram que a estratégia proposta é uma solução viável para identificar e combater a poluição de conteúdo em transmissões ao vivo. Em diversas configurações a solução foi capaz de reduzir consideravelmente a poluição de conteúdo, em vários casos chegando a eliminá-la no decorrer das transmissões. Além disso, os resultados ainda mostram que a solução adiciona baixa sobrecarga no tráfego da rede.

O restante deste trabalho está organizado da seguinte forma. A Seção 2 realiza uma descrição sobre o protocolo Fireflies e sobre o diagnóstico baseado em comparações. A Seção 3 apresenta a solução proposta para o combate à propagação de poluição de conteúdo em transmissões ao vivo. Já na Seção 4 resultados experimentais são apresentados. A Seção 5 descreve trabalhos relacionados e, por fim, a Seção 6 apresenta as conclusões e trabalhos futuros.

2. Definições Preliminares

Esta seção apresenta inicialmente uma descrição do protocolo Fireflies – um protocolo escalável para redes *overlay* no qual a solução proposta neste trabalho foi implementada. Na sequência é apresentada uma visão geral sobre o modelo de diagnóstico baseado em comparações que é utilizado pela solução proposta.

2.1. O Fireflies

O protocolo Fireflies é um protocolo escalável que cria uma rede *overlay* tolerante a intrusões [Johansen et al. 2006]. Todos os *peers* da rede executam o protocolo Fireflies usando a estratégia *pull-based* para a transmissão dos dados [Loocher et al. 2007], e os *peers* são organizados em uma topologia *mesh* [Pai et al. 2005]. Além dos *peers*, também existe um *servidor fonte* que gera os *chunks* que são transmitidos na rede. O servidor fonte é considerado uma unidade confiável e que nunca falha.

No Fireflies os *chunks* são enviados pelo servidor fonte para um número configurável de *peers* da rede. Os *peers* então compartilham os *chunks* entre si, com o objetivo de que todos os *peers* da rede recebam todos os *chunks* transmitidos pelo servidor fonte. No protocolo Fireflies todos os *peers* possuem um identificador sequencial e são organizados em múltiplos anéis [Johansen et al. 2006]. O número de anéis é configurável e cada anel contem todos os *peers* do sistema. Estes anéis têm o simples propósito de determinar o conjunto de vizinhos de cada *peer*. Além disso, é possível que um determinado *peer* possua vizinhos em comum em mais de um anel. Portanto cada *peer* da rede sempre possui pelo menos 2 vizinhos e no máximo $(2 * \lambda)$, onde λ representa o número de anéis configurados.

Como exemplo, a Figura 1 ilustra um sistema com 12 *peers* organizados em 4 anéis. Note que os vizinhos do *peer* 1 são os *peers* 3, 4, 5, 6, 7 e 9. A figura também ilustra a situação onde um *peer* possui vizinhos em comum em mais de uma anel: o *peer* 1 possui os *peers* 3 e 9 como vizinhos em dois diferentes anéis. No Fireflies o servidor fonte recebe o identificador 0 e não participa da configuração dos anéis.

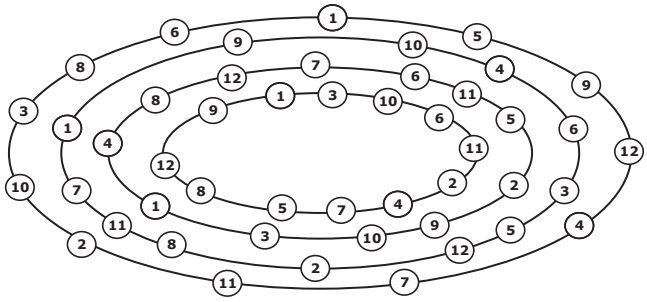


Figura 1. Um exemplo de rede Fireflies com 12 *peers* organizados em 4 anéis.

O protocolo Fireflies ainda configura em cada *peer* uma *janela de disponibilidade* e uma *janela de interesse*. A janela de disponibilidade é uma lista que indica quais *chunks* cada *peer* possui disponíveis para envio a seus vizinhos. Já a janela de interesse indica quais *chunks* cada *peer* ainda precisa receber. Quando um *peer* recebe um *chunk*, ele notifica a todos os seus vizinhos sobre a disponibilidade daquele chunk. Com base nestas notificações cada *peer* é capaz de manter uma lista de quais *chunks* estão disponíveis em cada um dos seus vizinhos. Desta forma, se um *peer* i for notificado por um de seus vizinhos v sobre a disponibilidade de *chunk* c, e se este *chunk* c estiver na janela de interesse do *peer* i, este *peer* requisita o *chunk* c ao vizinho v. Quando o *peer* v receber a requisição, se o *chunk* c ainda estiver na sua janela de disponibilidade, o *peer* v envia o *chunk* c ao *peer* i; caso contrário o *peer* v simplesmente ignora aquela requisição. Este é exatamente o procedimento que também ocorre com todos os *chunks* gerados pelo servidor fonte: quando o fonte gera um novo *chunk* ele notifica seus vizinhos sobre a disponibilidade daquele *chunk*, e então a sua difusão se inicia pelos *peers* da rede.

2.2. Diagnóstico Baseado em Comparações

O diagnóstico baseado em comparações [Duarte Jr. et al. 2011] utiliza a comparação do resultado de tarefas para determinar o estado de cada unidade do sistema, que por sua vez pode ser *falho* ou *sem-falha*. O modelo MM de diagnóstico baseado em comparações foi proposto por Maeng e Malek em [Maeng and Malek 1981] para detectar falhas em sistemas multiprocessados compostos por unidades homogêneas. Neste modelo o sistema é representado por um grafo $G = (V, E)$, onde V é o conjunto de unidades e E é o conjunto de *links* de comunicação. Uma unidade i é uma unidade *comparadora* – ou *testadora* – de outras duas unidades j e k se e somente se $(i, j) \in E$ e $(i, k) \in E$, e também se $i \neq j$ e $i \neq k$. Um *teste*, denotado por $(j, k)_i$, é definido como o envio de uma tarefa a partir de uma unidade comparadora i para outras duas unidades do sistema, unidades j e k. As unidades j e k executam a tarefa recebida e devolvem o resultado da tarefa à unidade testadora que então realiza a comparação do resultado das tarefas.

Se a unidade testadora for uma unidade sem-falha e o resultado da comparação das tarefas indicar *igualdade*, ambas as unidades comparadas são consideradas sem-falha. Por

outro lado, se a comparação indicar *diferença*, ao menos uma das três unidades i, j e k é falha. Este modelo ainda assume que a saída de tarefas produzidas por duas unidades falhas é sempre diferente. O conjunto com todos os resultados das comparações – ou seja, o conjunto de todos os resultados dos testes – é chamado de *síndrome* do sistema.

Em [Ziwich et al. 2005] um modelo distribuído de diagnóstico baseado em comparações é apresentado no qual as próprias unidades são capazes de realizar o diagnóstico. Além disso, outra característica importante é que este modelo assume que a comparação do resultado de tarefas produzidas por duas unidades falhas pode resultar em igualdade. A estratégia proposta neste trabalho emprega este modelo distribuído de diagnóstico [Ziwich et al. 2005], ou seja, dois diferentes *peers* do sistema podem eventualmente possuir uma mesma cópia poluída de um determinado *chunk*. Além disso, na solução proposta, um determinado *peer* testador realiza testes através da solicitação de um determinado *chunk* a todos os seus vizinhos. O resultado da tarefa, que é o conteúdo do próprio *chunk* recebido, é então comparado, em pares. Com base no resultado das comparações os *peers* são agrupados (ou classificados) em conjuntos de acordo com o conteúdo dos *chunks* recebidos.

3. A Estratégia Proposta para Combate à Poluição de Conteúdo

Esta seção apresenta a estratégia proposta para combater a propagação de poluição de conteúdo em transmissões ao vivo em redes P2P. A estratégia proposta utiliza o diagnóstico baseado em comparações para detectar *peers* poluidores e é baseada no protocolo Fireflies. Além do servidor fonte e dos *peers* – que já são componentes da arquitetura do Fireflies – a solução proposta implementa um novo componente, chamado de *módulo comparador*.

O módulo comparador é um componente que é executado por cada *peer*, e é integrado ao próprio código do protocolo Fireflies. Como o módulo comparador é integrado ao próprio protocolo Fireflies, ele possui acesso aos *chunks* recebidos e às janelas de disponibilidade daquele *peer*. Além disso o módulo comparador é o componente responsável por realizar as comparações de determinados *chunks*. Para isso cada *peer*, de forma independente dos demais, escolhe aleatoriamente um *chunk* para ser comparado, dentro dos chamados intervalos de monitoramento. Por sua vez, o intervalo de monitoramento é uma configuração do módulo comparador que indica o tempo máximo no qual o módulo comparador de cada *peer* deve escolher aleatoriamente um *chunk* para ser comparado.

De forma resumida, o procedimento executado por cada *peer* da rede é o descrito a seguir. Assim que um *peer* i recebe um *chunk* de identificador cid de um *peer* vizinho v, este *peer* i verifica se o identificador cid é o identificador de um dos *chunks* que foi escolhido aleatoriamente para ser comparado. Caso este seja um dos *chunks* que devem ser comparados, o módulo comparador do *peer* i o requisita a cada um dos vizinhos que informou sua disponibilidade. É importante ressaltar que estas requisições adicionais realizadas pelo modulo comparador, são requisições regulares do protocolo Fireflies, e são realizadas pelo próprio *peer* e através de conexões do próprio sistema. Em outras palavras, um *peer* que receber esta requisição vinda do módulo comparador do *peer* i não consegue diferenciá-la de qualquer outra requisição recebida de qualquer outro *peer*, e portanto não é possível tratá-la de forma diferenciada.

Assim que o *peer* i concluir a requisição e receber as respostas com o *chunk* re-

quisitado de identificador cid de cada um de seus vizinhos, este *peer* i compara os *chunks* recebidos em pares, e de acordo com o resultado das comparações classifica cada um dos *peers* em um conjunto $U_{i,cid}$. Cada conjunto $U_{i,cid}$ é um conjunto criado no *peer* i e contém cada um um dos *peers* vizinhos do *peer* i, classificados de acordo com o conteúdo do *chunk* cid, e tem o seguinte formato:

$$U_{i,cid} = \{(chunk_a, \{peer_j, peer_k, ...\}), (chunk_b, \{peer_m, peer_n, ...\}), ...\}.$$

Como cada *peer* tem um tempo limite para responder a cada requisição de *chunks*, se por qualquer motivo um *peer* não responder a uma requisição sobre um determinado *chunk*, ele é inserido em um conjunto especifico dos *peers* que não responderam às requisições. Destaca-se que este tempo máximo para cada *peer* esperar as respostas de seus vizinhos é o mesmo tempo configurado como a janela de interesse, ou seja, é o mesmo tempo em que cada *peer* normalmente já espera um determinado *chunk* da transmissão.

Assim que cada *peer* i finalizar um conjunto $U_{i,cid}$, ou seja, o conjunto $U_{i,cid}$ está completo e contém todos os vizinhos do *peer* i, o seguinte procedimento é executado. Se neste conjunto $U_{i,cid}$ existir um subconjunto de *peers* que responderam, e cuja quantidade de $peers$ neste subconjunto seja maior que a metade do número de vizinhos do *peer* i, então: aquele *peer* i (e apenas quele *peer* i) passa a partir daquele momento a ignorar todos os *chunks* e notificações de disponibilidade de *chunks* de qualquer um dos *peers* que não estiverem neste maior conjunto. Esta lista de *peers* bloqueados é constantemente atualizada por cada um dos *peers* do sistema, sempre que cada módulo comparador finalizar um novo conjunto U.

Como exemplo, a Figura 2 ilustra as requisições realizadas pela estratégia proposta. Este exemplo considera que o *peer* 20 escolheu o *chunk* com identificador 325 como um dos *chunks* para serem comparados. Como os vizinhos do *peer* 20 na rede são os *peers* 5, 12, 32, 43 e 57, assim que estes *peers* vizinhos notificarem ao *peer* 20 que possuem o *chunk* 325 disponível, o *peer* 20 irá solicitar a cada um de seus vizinhos este *chunk* 325. Nesta figura as setas direcionadas representam o envio do *chunk* 325 para o *peer* 20, a partir de cada um dos seus vizinhos; as demais arestas representam *links* de comunicação entre os próprios *peers* ou entre os *peers* e o servidor fonte. Ainda neste exemplo, o conteúdo original do *chunk* 325 é ilustrado por "OrigData" e o conteúdo modificado (poluído) indevidamente deste *chunk* é ilustrador por "PollData". O exemplo considera que apenas o *peer* 43 é um *peer* poluidor e que neste momento os *peers* 5, 12, 32 e 57 possuem uma cópia correta do *chunk* 325.

Ainda com base na Figura 2, assim que o *peer* 20 possuir informações de todos os seus vizinhos a respeito do *chunk* 325, o *peer* 20 irá realizar as comparações dos *chunks* recebidos, em pares, e classificará cada um dos seus *peers* vizinhos no conjunto $U_{20,325}$. Especificamente neste exemplo o conjunto gerado, após finalizado, será $U_{20,325} = \{(OrigData, \{5, 12, 32, 57\}), (PollData, \{43\})\}$. Assim que este conjunto $U_{20,325}$ for finalizado pelo *peer* 20, será possível identificar que existe um conjunto com mais da metade do número de vizinhos do *peer* 20. Em outras palavras, o conjunto $(OrigData, \{5, 12, 32, 57\})$ possui 4 *peers*, e $4 \geq (5/2)$, onde $5/2$ é a metade do número de vizinhos do *peer* 20. A partir deste momento o *peer* 20 irá parar de solicitar *chunks* ao *peer* 43.

De forma detalhada e complementando o resumo apresentado acima, a Figura 3 mostra o algoritmo em pseudocódigo que implementa o módulo comparador. A Fi-

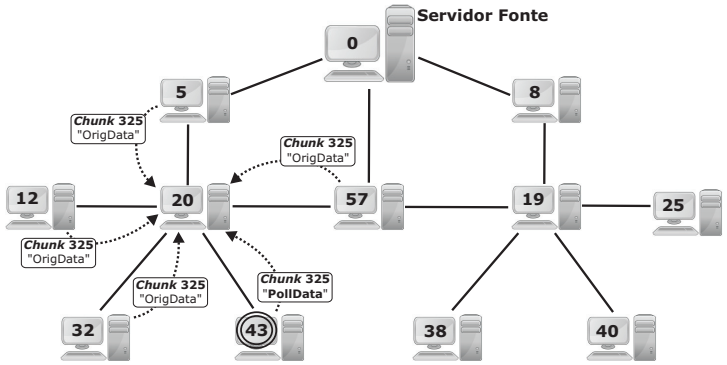


Figura 2. Transmissão do *chunk* 325 para o *peer* 20; o *peer* 43 é poluidor.

gura 3 mostra o algoritmo em pseudocódigo executado pelo módulo comparador, ou seja, é o código responsável pelas solicitações adicionais de *chunks*, pela realização das comparações e pela geração da lista de *peers* vizinhos que terão notificações de *chunks* ignoradas – a lista dos *peers* bloqueados.

Inicialmente (na linha 2) o módulo comparador que executa no *peer* i determina aleatoriamente os identificadores dos primeiros *chunks* para serem comparados por aquele *peer* i. Os identificadores de *chunks* para comparação são escolhidos aleatoriamente, da seguinte forma: $proximo\_cid\_para\_comparar \leftarrow ultimo\_cid\_gerado + mod(random, (monitoring\_interval * mcast\_rate))$, onde $random$ representa um número aleatório, $monitoring\_interval$ é o tempo máximo (em segundos) configurado como o intervalo de monitoramento da solução, e $mcast\_rate$ é o número de *chunks* gerados por segundo pelo fonte. Em outras palavras, o próximo *chunk* a ser monitorado (ou comparado) será qualquer um dos *chunks* gerados pelo servidor fonte nos próximos $monitoring\_interval$ segundos após o momento de geração do último *chunk*. Como exemplo, caso $monitoring\_interval = 15$, $mcast\_rate = 30$ e $ultimo\_cid\_gerado = 5674$, o valor do $proximo\_cid\_para\_comparar$ será um número aleatório entre 5674 e $(5674 + (15 * 30))$. Assim que os próximos identificadores para comparação forem sendo escolhidos eles são inseridos na lista $lista\_de\_cids$.

```
Algoritmo: ModuloComparador /* executando no peer i */
 1: inicio
 2:    lista_de_cids ← gera e atualiza a lista de chunks aleatórios para serem comparados
 3:                    (considerando o intervalo de monitoramento configurado);
 4:    sempre que um vizinho v disponibilizar um novo chunk cid faça
 5:       se cid ∈ lista_de_cids então
 6:          requisitar o chunk cid ao peer v;
 7:          atualizar o U_{i,cid} correspondente com a resposta do peer v;
 8:       fim se
 9:    fim sempre que
10:
11:    sempre que ((U_{i,cid} possuir informação sobre todos os peers vizinhos do i) ou
12:                (acabou o tempo limite para obtenção de informações sobre aquele chunk cid)) faça
13:       se (acabou o tempo limite para obtenção de informações sobre aquele chunk cid) então
14:          incluir vizinhos que não responderam em um subconjunto específico do U_{i,cid};
15:       fim se
16:       se U_{i,cid} possui um subconjunto com mais de N(i)/2 de peers então
17:          lista_de_peers_bloqueados ← ∅; /* limpa a lista de peers bloqueados */
18:          lista_de_peers_bloqueados ← peers que não estão no maior conjunto;
19:       fim se
20:       list_de_cids ← atualiza a lista com novos chunks aleatórios para serem comparados;
21:    fim sempre que
22: fim
```

Figura 3. Algoritmo em pseudocódigo implementado pelo módulo comparador.

O módulo comparador então espera por notificações dos seus vizinhos sobre a disponibilidade de novos *chunks*. Sempre que um vizinho v notificar a disponibilidade de requisição de um novo *chunk* cid (linha 4), o *peer* i executa o código das linhas 5–8: se o *chunk* de identificador cid é um *chunk* para ser monitorado, uma requisição do *chunk* cid é realizada pelo *peer* i ao *peer* v (linha 6). Assim que receber a resposta da requisição, ou seja, uma cópia do próprio *chunk* cid enviada pelo *peer* v, o conjunto $U_{i,cid}$ é atualizado (linha 7) e o *peer* v é classificado de acordo com o resultado da comparação do *chunk* v recebido.

Já as linhas 11–21 são executadas sempre que o conjunto $U_{i,cid}$ estiver completo, ou seja, já possuir informações sobre todos os vizinhos do *peer* i, ou ainda se o tempo limite para obtenção de informações sobre o *chunk* cid estiver esgotado. Caso tenha ocorrido este último caso (o tempo limite foi esgotado), os *peers* que não enviaram nenhuma informação a respeito do *chunk* cid são classificados em um subconjunto específico (linha 14). Por sua vez, o teste da linha 16 verifica se o conjunto $U_{i,cid}$ em questão possui algum subconjunto com mais de $N(i)/2$ *peers* que responderam, onde $N(i)$ é o número de vizinhos do *peer* i. Caso ocorra esta condição, o módulo comparador atualiza a lista $lista\_de\_peers\_bloqueados$ (linhas 17–18) que é usada pelo *peer* i como a lista dos *peers* dos quais o *peer* i irá a partir daquele momento ignorar a disponibilização de novos *chunks*. A lista $lista\_de\_peers\_bloqueados$ é sempre atualizada com todos os *peers* que não estão no maior subconjunto de *peers*.

Finalmente, cada *peer* do sistema Fireflies deve realizar um simples teste adicional para verificar se as notificações de disponibilidade de *chunks* devem ou não ser descartadas: sempre que um vizinho v notificar a disponibilidade de um novo *chunk*, o *peer* i verifica se este vizinho v está ou não na lista $lista\_de\_peers\_bloqueados$. Caso ocorra este caso, o *peer* i simplesmente ignora aquela notificação. Caso contrário, o *peer* i executa os procedimentos previstos pelo protocolo Fireflies para aquele evento da rede *overlay*.

4. Resultados Experimentais

A estratégia proposta foi implementada usando o simulador Fireflies descrito em [Haridasan and van Renesse 2006]. Um grande número de simulações foi executado para sistemas com 200 *peers*. Cada um dos experimentos simulou uma transmissão ao vivo por um período de 180 segundos. O servidor fonte gerou 30 *chunks*/segundo e o Fireflies foi configurado para organizar os *peers* em 15 anéis. O tamanho do *chunk* foi de 10KB. Ambas as janelas de disponibilidade e de interesse de todos os *peers* foram configuradas com 3000 *chunks*. Além disso, em todos os experimentos o intervalo de monitoramento configurado para o módulo comparador foi de 15 segundos, ou seja, no máximo a cada 15 segundos o módulo comparador de cada *peer* escolhia aleatoriamente um *chunk* para ser comparado entre todos os seus vizinhos. Os experimentos foram executados em um computador com processador AMD Phenom 9500 quad-core x64 e 4GB de memória RAM, executando o sistema operacional Linux 64-bits com kernel versão 2.6.18-238.el5.

Os principais propósitos dos experimentos foram (a) verificar qual foi o impacto da poluição nas transmissões, para diferentes quantidades de *peers* poluidores, e (b) computar o *overhead* adicionado pela solução proposta, em termos do número de *chunks* adicionais requisitados pelo módulo comparador. Os principais parâmetros variados nas simulações foram:

(1) a quantidade de *peers* poluidores, variando entre 0%, 5%, 10%, 15%, 20% e 25% do total de *peers* na rede;
(2) foram experimentadas simulações com e sem *churn*: para os experimentos com *churn*, 100 *peers* entraram na rede e outros 100 *peers* saíram da rede. O momento de entrada dos *peers* seguiu uma distribuição normal com média 100 e desvio padrão 20. Para modelar a saída dos *peers* da rede foi utilizada uma distribuição de Poisson com média 100;
(3) também foram realizadas simulações com o módulo comparador ativo e inativo, com o objetivo de comparar o efeito da poluição em redes com a solução proposta e em transmissões sem nenhuma solução para tratar a poluição.

Foram executadas um total de 1.000 simulações. Os resultados foram sumarizados e são apresentados nos gráficos das Figuras 4 a 7. As linhas dos gráficos representam os valores médios, enquanto que as linhas verticais de intervalo mostram o intervalo de confiança de 95% para a amostra de dados correspondente.

A Figura 4(a) mostra o número médio de *chunks* enviados normalmente pelo Fireflies durante as transmissões, sem a solução proposta, para as simulações com *churn* e sem *churn*. Pode-se notar que o número médio de *chunks* enviados pelo Fireflies esteve sempre entre 1 e 1.15 milhões de *chunks*. Já a Figura 4(b) mostra o número médio de *chunks* adicionais requisitados pela solução proposta, isto é, por todos os módulos comparadores de todos os *peers*. Pode-se notar que o número de *chunks* adicionais requisitados esteve sempre em torno de 70.000 a 100.000 *chunks*.

A proporção entre as informações mostradas em ambos os gráficos da Figura 4 indica a percentagem do *overhead* gerado pela solução proposta, em termos da quantidade de *chunks* adicionais requisitados pelo módulos comparadores. Pode-se notar que o módulo comparador adicionou uma sobrecarga de 7% a 8% ao uso do tráfego de rede. É importante lembrar que o intervalo de monitoramento configurado foi de até 15 segundos, e dependendo da largura de banda de rede disponível esta frequência pode ser aumentada ou diminuída.

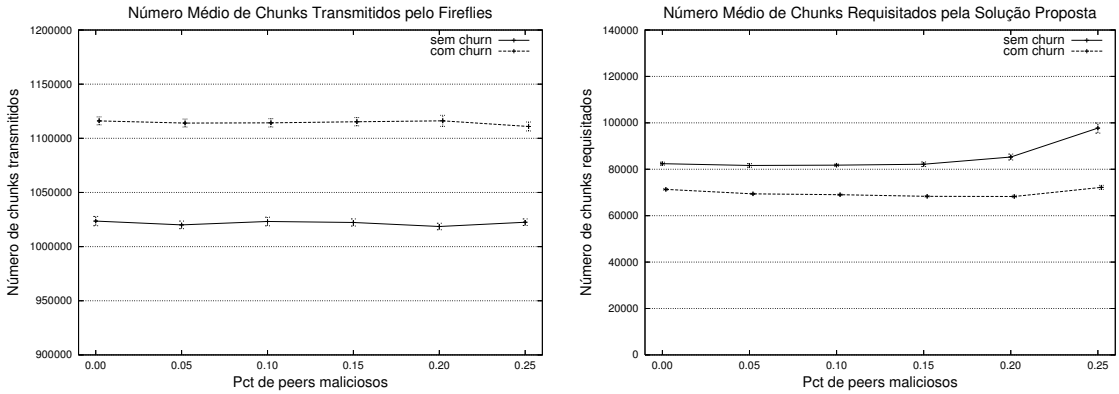
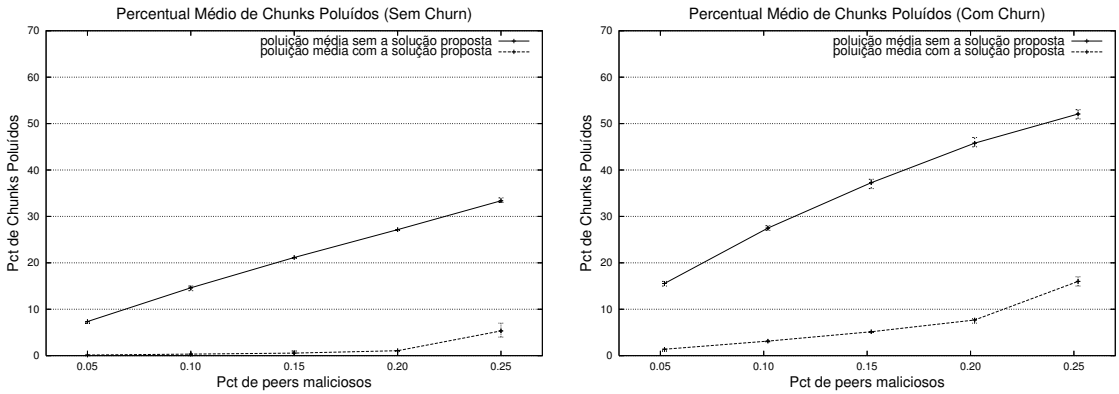


Figura 4. *Chunks* transmitidos normalmente pelo Fireflies e *chunks* adicionais requisitados pelo módulo comparador.

A Figura 5 mostra a quantidade média de *chunks* poluídos durante toda a transmissão para redes sem a solução proposta e para transmissões com a solução proposta

implementada. Pode-se notar que para experimentos sem *churn* com a solução proposta implementada, o percentual médio de *chunks* poluídos durante toda a transmissão caiu de 27.1% para 1% em experimentos onde 20% dos *peers* da rede foram configurados como poluidores, e caiu de 33.3% para 5.3% nos experimentos onde $1/4$ dos *peers* da rede eram *peers* poluidores. Já para experimentos com *churn* o percentual de poluição caiu de 45.7% para 7% em experimentos onde 20% dos *peers* eram poluidores, e caiu de 52% para 16% nos experimentos onde 25% dos *peers* eram poluidores.



(a) (b)

Figura 5. Percentual médio de *chunks* poluídos durante as transmissões em redes sem a solução proposta e com a solução proposta.

As próximas duas figuras mostram o percentual da quantidade de *chunks* poluídos, também para redes com e sem a solução proposta, mas agora durante cada segundo do tempo das transmissões: a Figura 6 mostra a poluição em redes sem a solução proposta, e a Figura 7 mostra a poluição durante as transmissões com a solução proposta ativa.

Em transmissões sem *churn* e sem a solução proposta – Figura 6(a) – pode-se notar que o percentual de *chunks* poluídos tem valores diferentes para cada quantidade de *peers* poluidores na rede, mas de forma geral possui dispersão pequena durante toda a transmissão. Já para as simulações com *churn* – Figura 6(b) – pode-se notar um aumento do percentual de *chunks* poluídos, conforme o tempo de simulação passa da metade do tempo de transmissão – chegando a percentuais próximos de 70% da quantidade de *chunks* poluídos, em transmissões com 25% de *peers* poluidores.

Com base na Figura 7(a) (experimentos sem *churn*) nota-se que, nas transmissões com a solução proposta ativa a poluição praticamente acabou após cerca de 40 segundos de transmissão para as simulações com até 20% de *peers* configurados como poluidores, e nas simulações com 25% de *peers* poluidores, o percentual de *peers* poluídos caiu para cerca de 2% após 80 segundos de transmissão. Casos como este das simulações com 25% de *peers* poluidores onde a poluição ainda continuou com valores acima de 0% ocorrem pois o aumento da quantidade de *peers* poluidores na rede aumenta também a probabilidade da maior parte dos vizinhos de um determinado *peer* serem *peers* poluidores.

A Figura 7(a) ainda mostra que a poluição no sistema foi mais alta apenas nos segundos iniciais das transmissões, ou seja, após as primeiras comparações serem realizadas pelos módulos comparadores, cada *peer* foi capaz de identificar e não solicitar mais dados aos *peers* identificados como poluidores. Além disso, também com base nestes

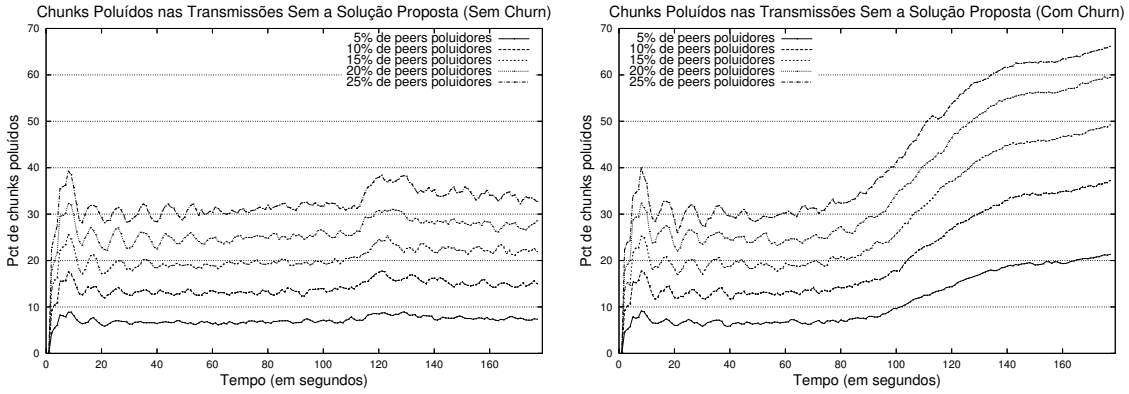


Figura 6. *Chunks* poluídos transmitidos em redes sem a solução proposta, em cada segunda da transmissão.

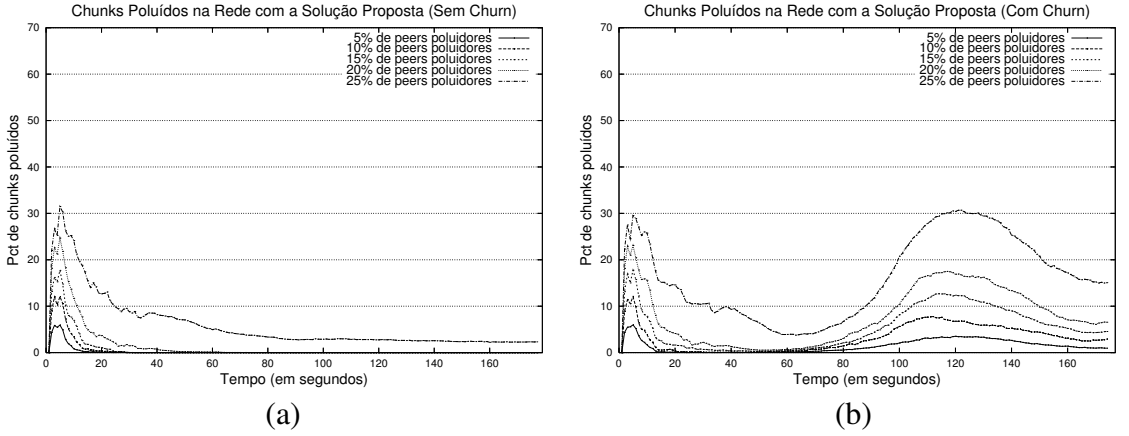


Figura 7. *Chunks* poluídos transmitidos em redes com a solução proposta, em cada segunda da transmissão.

dados, se os 30 ou 40 segundos iniciais de cada transmissão fossem retirados do cálculo da média, o percentual de *chunks* poluídos de toda a transmissão – informação que foi mostrada na Figura 5(b) – reduziria ainda mais significativamente.

Finalmente com base na Figura 7(b) – que mostra o percentual de poluição nas transmissões com a solução proposta nos experimentos com *churn* – nota-se que nos momentos onde ocorrem maiores taxas de entrada de novos *peers* na rede (entre os tempos 100 e 120 segundos) ocorre um aumento no percentual de *chunks* poluídos. Este comportamento ocorre pois os novos *peers* que entram na rede não possuem conhecimento inicial sobre quais dos seus vizinhos são *peers* poluidores. Por outro lado, conforme o tempo de simulação avança e estes novos *peers* iniciam as suas comparações, o percentual de poluição volta a cair novamente. Neste sentido, o experimento sem *churn* mostrado na Figura 7(a) reflete melhor o comportamento geral de cada *peer* quando ele inicia sua participação na transmissão, ou seja, nos momentos iniciais, o *peer* acaba recebendo uma maior quantidade de *chunks* poluídos por desconhecer quais dos seus vizinhos são poluidores, mas conforme o tempo avança – e o módulo comparador realiza e finaliza as comparações – este *peer* identifica e para de solicitar *chunks* a vizinhos poluidores.

5. Trabalhos Relacionados

Diversas estratégias que tratam a poluição de conteúdo – ou *poisoning* [Christin et al. 2005] – em redes P2P foram publicadas na literatura. Algumas das mais relevantes são brevemente descritas a seguir. Uma estratégia básica para combate à poluição de conteúdo, empregada pelo BitTorrent [Dhungel et al. 2007] é permitir que os *peers* obtenham previamente os valores *hash* de todos os *chunks* [Wong and Lam 1999]. Desta forma quando um *chunk* é recebido cada *peer* pode verificar a integridade daqueles dados. Em transmissões ao vivo um dos problemas dessa técnica está em receber previamente o valor *hash* de conteúdos que são gerados durante a própria transmissão.

Outra estratégia consiste na aplicação da criptografia de chave pública, ou seja, disseminar todo *chunk* juntamente com uma assinatura digital correspondente gerada pelo servidor fonte [Haridasan and van Renesse 2006]. A desvantagem dessa estratégia é que principalmente para transmissões ao vivo, dependendo dos dispositivos dos usuários envolvidos nas transmissões, a criptografia de chave publica pode ser proibitiva. Uma variante desta estratégia, chamada de *Linear Digests* também é apresentada em [Haridasan and van Renesse 2006] e agrupa os valores *hash* de um conjunto de *chunks* em uma mesma assinatura digital, que também é gerada pelo servidor fonte.

Algumas outras ferramentas [Liang et al. 2006, Li et al. 2012] ainda aplicam criptografia simétrica a todos os *chunks* usando uma chave secreta compartilhada pré-definida. Em [Li et al. 2012] também é proposto um mecanismo seguro de gerenciamento de chaves no qual o servidor fonte periodicamente recria e retransmite a nova chave compartilhada para um número limitado de *peers* da rede. Já nas técnicas de listas negras [Liang et al. 2005] utilizam faixas de IPs para englobar o maior número possível de *peers* que disseminaram conteúdo poluído. O desafio desta técnica é englobar o menor número de *peers* não poluidores nestas faixas de IPs. Além disso [Gheorghe et al. 2010] aponta que esta técnica se mostra custosa quando aplicadas para transmissões ao vivo.

Em [Chen et al. 2008] é apresentada uma outra solução que emprega o conceito de grupos de *peers* que são mantenedores da integridade do conteúdo transmitido. Nesta solução o servidor fonte publica o conteúdo a este grupo de peers. Todos os demais *peers* podem realizar requisições a este grupo de *peers* para verificar a integridade de um *chunk* específico. Já em [Walsh and Sirer 2006] os autores apresentam um sistema P2P para compartilhamento de arquivos, baseado em reputação. Neste sistema os próprios *peers* da rede classificam outros *peers* como honestos, que por sua vez adquirem acesso ao conteúdo compartilhado. Em [Borges et al. 2008] uma outra solução também baseada no mesmo princípio de reputação é apresentada, mas agora aplicada a transmissões ao vivo.

Em [Dhungel et al. 2009] os autores realizam a avaliação de quatro técnicas mencionadas acima: criptografia simétrica, verificação de *hashes*, assinaturas digitais e lista negra. Os autores concluem que o uso de árvores de Merkle é um dos mecanismos mais eficientes em termos da sobrecarga computacional adicionada. Mais recentemente em [Lin et al. 2010] uma avaliação do impacto de ataques de poluição é realizada. O trabalho mostra que o impacto de um ataque de poluição não está diretamente relacionado ao tamanho da rede em si, mas que depende fortemente dos níveis de *churn* e da banda de rede disponível nos *peers* maliciosos e no servidor fonte.

Uma estratégia baseada em *network coding* [Feng and Li 2008] é apresentada em

[Wang et al. 2010] para identificar e limitar a poluição de conteúdo em transmissões ao vivo em redes P2P. Cada *chunk* transmitido pelo servidor fonte é dividido em blocos. Por sua vez cada bloco é subdividido em palavras (ou *codewords*), que acabam convertendo cada um dos *chunks* em uma matriz de elementos de um campo de Galois (*Galois Field*, ou GF). Por fim blocos codificados – que são criados baseados nas matrizes GF e combinam um vetor de coeficientes aos blocos originais – são as informações transmitidas pelo servidor fonte para os peers. Cada *peer* que recebe estes blocos codificados, os decodificam para reconstruir os *chunks* originais.

Já em [Borges et al. 2012, Oliveira et al. 2009] uma caracterização do tráfego do SopCast é apresentada. Uma das conclusões do trabalho é que um *peer* malicioso foi capaz de comprometer 50% dos *peers* da rede e 30% da banda de rede. Por fim, recentemente em [Coelho et al. 2011] os autores apresentam uma avaliação dos mecanismos de autenticação de conteúdo em redes P2P para transmissão ao vivo. Os autores comparam a sobrecarga gerada e a segurança adicionada por diversas técnicas e mostram que, para transmissões ao vivo de alta resolução, os mecanismos com sobrecarga aceitáveis avaliados não foram resilientes aos ataques de poluição.

6. Conclusão

Este trabalho apresentou uma nova estratégia que utiliza o diagnóstico baseado em comparações para combater a poluição de conteúdo em transmissões ao vivo em redes *overlay*. Cada *peer* do sistema executa comparações periódicas sobre determinados *chunks* de seus vizinhos. Com base nos resultados das comparações, cada *peer*, de forma independente dos demais, deixa de solicitar *chunks* aos seus vizinhos poluidores. A estratégia proposta foi implementada usando o protocolo Fireflies e um grande número de experimentos através de simulações foram conduzidos. Os resultados dos experimentos mostraram que a solução adiciona cerca de 7% a 8% de sobrecarga ao tráfego de rede, e que é uma solução viável para a detecção e contenção da poluição em transmissões ao vivo em redes P2P. Trabalhos futuros incluem implementar a estratégia em um serviço real de transmissões ao vivo na Internet, avaliar a estratégia proposta para ataques do tipo omissão, com uma maior quantidade de peers e com outros intervalos de monitoramento, além de avaliar a estratégia em outros tipos de redes *overlay*.

Referências

Borges, A., Almeida, J. M., and Campos, S. V. A. (2008). Fighting Pollution in P2P Live Streaming Systems. *Proc. of the IEEE Intl. Conf. on Multimedia and Expo*, pages 481–484.

Borges, A., Gomes, P., Nacif, J., Mantini, R., Almeida, J. M., and Campos, S. V. A. (2012). Characterizing SopCast Client Behavior. *Computer Communications*, 35(8):1004–1016.

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

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

Coelho, R. V., Pastro, J. T., Antunes, R. S., Barcellos, M. P., Jansch-Pôrto, I., and Gaspary, L. P. (2011). Challenging the Feasibility of Authentication Mechanisms for P2P Live Streaming. *Proc. of the 6th Latin America Networking Conference*, pages 55–63.

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

Dhungel, P., Hei, X., Ross, K. W., and Saxena, N. (2009). Pollution in P2P Live Video Streaming. *Intl. Journal of Computer Networks and Communications*, 1(2):99–110.

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

Feng, C. and Li, B. (2008). On Large-Scale Peer-to-Peer Streaming Systems with Network Coding. *Proc. of the 16th ACM Intl. Conf. on Multimedia*, pages 269–278.

Gheorghe, G., Cigno, R. L., and Montresor, A. (2010). Security and Privacy Issues in P2P Streaming Systems: A Survey. *Peer-to-Peer Networking and Applications*, 4(2):75–91.

Haridasan, M. and van Renesse, R. (2006). Defense Against Intrusion in a Live Streaming Multicast System. *Proc. of the 6th IEEE Intl. Conf. on Peer-to-Peer Computing*, pages 185–192.

Johansen, H., Allavena, A., and van Renesse, R. (2006). Fireflies: Scalable Support for Intrusion-Tolerant Network Overlays. *Proc. of the 1st ACM SIGOPS/EuroSys European Conf. on Computer Systems*, 40(4):3–13.

Li, J.-S., Hsieh, C.-J., and Wang, Y.-K. (2012). Distributed Key Management Scheme for Peer-to-Peer Live Streaming Services. *Intl. Journal of Communication Systems*.

Liang, J., Kumar, R., and Ross, K. W. (2006). The FastTrack Overlay: A Measurement Study. *Computer Networks*, 50(6):842–858.

Liang, J., Naoumov, N., and Ross, K. W. (2005). Efficient Blacklisting and Pollution-Level Estimation in P2P File-Sharing Systems. *Proc. of the Asian Internet Engineering Conf.*, pages 173–175.

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

Loocher, T., Meier, R., Schmid, S., and Wattenhofer, R. (2007). Push-to-Pull Peer-to-Peer Live Streaming. *Proc. of the 21st Intl. Symp. on Distributed Computing*, pages 388–402.

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

Oliveira, J., Borges, A., and Campos, S. V. A. (2009). Content Pollution on P2P Live Streaming Systems. *Proc. of the 15th Brazilian Symp. on Multimedia and the Web*, (50).

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

Schimidt, E. A., Ziwich, R. P., Duarte Jr., E. P., and Jansch-Pôrto, I. (2011). Diagnóstico de Poluição de Conteúdo em Redes P2P para Transmissões de Mídia Contínua ao Vivo. *Proc. of the 17th Brazilian Symp. on Multimedia and the Web*, pages 221–228.

Walsh, K. and Sirer, E. G. (2006). Experience with an Object Reputation System for Peer-to-Peer Filesharing. *Proc. of the 3rd USENIX Symp. on Networked Systems Design and Implementation*, 3:1–14.

Wang, Q., Vu, L., Nahrstedt, K., and Khurana, H. (2010). MIS: Malicious Nodes Identification Scheme in Network-Coding-Based Peer-to-Peer Streaming. *Proc. of the 29th IEEE Intl. Conf. on Computer Communications*, pages 1–5.

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

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

Zhang, P. and Helvik, B. E. (2011). Modeling and Analysis of P2P Content Distribution under Coordinated Attack Strategies. *IEEE Consumer Communications and Networking Conf.*, pages 131–135.

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