Protegendo BitTorrent: projeto e avaliação de contra-medidas eficazes para ataques DoS*

Daniel Bauermann[1], Matheus Lehmann[1], Rodrigo Mansilha[1], Marinho P. Barcellos[2,3]

[1] UNISINOS – Universidade do Vale do Rio dos Sinos
Av. Unisinos, 950 – São Leopoldo, RS

[2]PPGC – Programa de Pós-Graduação em Computação
UFRGS – Universidade Federal do Rio Grande do Sul
Av. Bento Gonçalves, 9500 – Campus do Vale – Bloco IV – Porto Alegre, RS

[3]PUCRS – Pontifícia Universidade Católica do Rio Grande do Sul
Av. Ipiranga, 6681, Prédio 32 – Porto Alegre, RS

{dbauermann,matios,rmansilha}@gmail.com, marinho@acm.org

Abstract. *BitTorrent is a P2P file-sharing protocol that can be used to efficiently distribute files such as software updates and digital content to very large numbers of users. In a previous paper, we have shown that vulnerabilities can be exploited to launch Denial-of-Service (Dos) attacks against BitTorrent swarms. In this paper, we review the three most damaging attacks, and propose two algorithms as countermeasures to effectively tackle them. We implemented the attacks and countermeasures in TorrentSim, a packet-level BitTorrent simulator. The results indicate that our proposed approach is effective when there is an ongoing attack while at the same time efficient when the countermeasure is active but there is no attack.*

Resumo. *BitTorrent é um protocolo de compartilhamento de arquivos P2P que pode ser usado para distribuir de forma eficiente arquivos de grande volume para um número elevado de usuários. Em um trabalho anterior, foi demonstrado que vulnerabilidades podem ser exploradas para efetivar ataques de negação de serviço (DoS) em enxames BitTorrent. Neste trabalho, são revistos os três ataques mais prejudiciais e proposto dois algoritmos como contra-medidas de defesa destes ataques. Os ataques e as contra-medidas foram implementados no TorrentSim, um simulador de BitTorrent. Os resultados indicam que a proposta deste trabalho é eficaz e eficiente em situações de ataque e sem ataque, respectivamente.*

1. Introdução

O BitTorrent é um sistema de compartilhamento de arquivos baseado no paradigma P2P. Através dele são utilizados recursos dos pares na rede para distribuir eficientemente um grande volume de dados. Em função de suas qualidades, aplicações estão sendo desenvolvidas com base no BitTorrent para distribuição de conteúdo digital gerado por companhias de mídia, como BBC e Twentieth Century Fox.

Por essas razões, ataques de negação de serviço (DoS) a aplicações baseadas em BitTorrent podem ter grande impacto em seus usuários. Existem estudos na literatura científica sobre o comportamento nocivo dos pares em redes BitTorrent. Por exemplo, [Locher et al. 2006] apresenta o BitThief, um agente de usuário que executa downloads

*Auxílio financeiro do CNPq: Projeto de Pesquisa P2P-SeC; Bolsa Mestrado D. Bauermann.

mas nunca contribui para a rede, e [Sirivianos et al. 2007] identifica um ataque (*Large View Exploit*) no qual um *free-rider* beneficia-se de uma visão mais ampla (em número de pares) do enxame. Em [Piatek et al. 2007] é apresentado o BitTyrant, uma versão modificada do Azureus [Azu 2007] aderente ao protocolo mas que usa diferentes políticas para incrementar o desempenho de download enquanto reduz as contribuições de upload. Em [Shneidman et al. 2004] são apresentas duas estratégias, denominadas múltiplas identidades e conteúdo arbitrário, que combinadas dão desempenho ao par semelhante ao comportamento de um *free-riding*. Por fim, em [Liogkas et al. 2006] são apresentadas três técnicas nas quais um par não respeita o protocolo, mente sobre a disponibilidade de peças e recusa-se a cooperar.

O aspecto comum entre os trabalhos anteriores é o foco em como o par pode *maximizar o benefício* recebido de um enxame BitTorrent enquanto minimiza os recursos por ele contribuídos. Em contraste, no presente trabalho um par malicioso *não* está interessado em executar downloads; em vez disto, sua única intenção é fazer uma negação de serviço: *impedir* o progresso de um ou mais enxames usando o mínimo de recursos necessários. Em [Konrath et al. 2007] é demonstrado que enxames BitTorrent são vulneráveis a um conjunto de ataques DoS. No presente trabalho, são introduzidos dois algoritmos de contra-medida, denominados "Rotação de Pares" e "Anti-corrupção", que eficientemente reduzem o impacto dos ataques previamente identificados e não geram sobrecarga nos casos sem ataques. Até onde sabe-se, não existem contra-medidas de segurança para BitTorrent na literatura científica. Assim, a contribuição deste trabalho é projetar e avaliar estes dois algoritmos.

As contra-medidas propostas foram implementadas e tiveram sua eficácia avaliada através de simulações. Para isto, cenários representativos foram montados, comparando situações em que a presença de ataque, assim como o mecanismo de contra-medida, são variados. Os resultados sugerem que os algoritmos propostos possam ser incorporados ao protocolo como contra-medidas, tornando implementações de agentes de usuário e aplicações derivadas mais seguras.

O restante deste trabalho está organizado como segue. A Seção 2 formaliza a dinâmica de um enxame em função de conjuntos de pares, revisa brevemente os três ataques combatidos neste artigo e termina com uma análise das contra-medidas existentes em alguns agentes. A seguir, as Seções 3 e 4 apresentam os algoritmos propostos, Rotação de Pares e Anti-corrupção, respectivamente, e discutem seu funcionamento. A implementação dos algoritmos em um simulador de BitTorrent em nível de mensagem viabilizou a avaliação da eficácia e eficiência dos algoritmos propostos, conforme abordado na Seção 5. A Seção 6 encerra com conclusões e perspectivas de trabalhos futuros.

2. Visão Geral

Existem diversos trabalhos na literatura e fontes menos confiáveis sobre o funcionamento do protocolo BitTorrent. Pode-se assumir seguramente que trata-se de um protocolo razoavelmente conhecido[1]. Apresenta-se a seguir uma revisão sobre BitTorrent, os ataques já identificados, bem como as contra-medidas existentes.

[1]O leitor interessado pode recorrer a uma lista de artigos que mantemos em `http://www.citeulike.org/user/p2p-sec/tag/bittorrent`.

2.1. Visão orientada a conjuntos do BitTorrent

O conteúdo compartilhado é dividido em "peças", cujo tamanho é fixo (exceto a peça final). Peças se encontram sub-divididas em "blocos" de 16KB. Considerando a relação de composição entre peças e blocos, denota-se uma peça x como b_x ou $b_{x,*}$ e um de seus blocos, y, como $b_{x,y}$. Um arquivo .torrent possui informações sobre o conjunto de peças que compõe o conteúdo, incluindo o *hash* SHA1 de cada peça b_x. Denota-se abstratamente que existe uma função *hash* h_x, para uma peça b_x, que retorna verdadeiro caso seja íntegra e falso caso contrário.

Um rastreador serve de ponto de encontro entre pares. Ao obter um arquivo .torrent e se conectar com o rastreador, um par tipicamente solicita uma lista de outros pares participantes do enxame (denotado como S_t) e informa seu endereço IP e a porta TCP em que estará esperando por conexões. Quando um par conecta, ele ao mesmo tempo se registra e obtém uma lista aleatória de pares (PEERLIST), definida como L.

O "par local" é denotado como p_i, enquanto $p_j, p_k, \ldots$ representam "pares remotos". Cada p_i mantém uma lista de "pares conhecidos" frequentemente referenciada como *Peer Set*, a qual define-se como o conjunto P (denotado também como P_i, no caso do P em p_i). A lista de pares correntemente conectados a p_i é representada pelo conjunto ACTIVEPEERSET, ou A_i. Quando uma conexão é aberta entre p_i e p_j, $A_i \leftarrow A_i \cup \{p_j\}$. Quando a conexão é fechada, $A_i \leftarrow A_i \setminus \{p_j\}$.

Um par que possui todas as peças é dito "semeador", ao passo que é "sugador" caso contrário. Pares podem ser *honestos* ou *maliciosos*. Assume-se que sugadores podem ser maliciosos ou honestos, mas nunca trocam de comportamento. Um semeador pode ser honesto ou malicioso – caso em que estará distribuindo conteúdo poluído, que é um tipo diferente de ataque [Christin et al. 2005]. No contexto deste trabalho, assume-se que semeadores são sempre honestos.

2.2. Ataques

Este trabalho propõe contra-medidas para três ataques: Eclipse, Mentira de Peças e Corrupção de Peças. Os dois primeiros ataques somente são efetivos com o uso de um grande número de pares; como o BitTorrent não requer autenticação, é permitido que usuários maliciosos simulem dezenas ou centenas de identidades (denominadas "*sybils*") que são percebidos na rede como inúmeros pares [Douceur 2002]. Segue uma descrição sucinta de cada ataque.

Eclipse. No contexto deste trabalho, o ataque consiste em um atacante criar um número proporcionalmente grande de *sybils*. Nesta proporção, os pares conhecidos pelo honesto tendem a ser dominados por maliciosos. **Mentira de Peças**. Visa desequilibrar a homogeneidade na quantidade de cópias de cada peça no enxame [Konrath et al. 2007]. Para tal, um par malicioso irá falsamente anunciar inúmeras vezes, através de seus *sybils*, a posse de uma peça que não possui. **Corrupção de Peças**. Consiste em enviar blocos corrompidos, tentando comprometer o maior número possível de peças. A peça com os blocos incorretos não será aceita pelo par realizando o download, sendo a mesma descartada e o seu download realizado novamente. A combinação dos ataques Eclipse e Mentira de Peças resulta em um ataque a ser referenciado como **Mentira em Massa**.

2.3. Contra-medidas existentes

O BitTorrent originalmente inclui um mecanismo, denominado "anti-snubbing", que objetiva lidar com pares cuja comunicação se encontra estagnada. Não existe, no entanto,

consenso quanto à forma de funcionamento desse mecanismo. Segundo [Cohen 2003], o *anti-snubbing* aumenta o número de pares que são concorrentemente beneficiados com *optimistic unchoke*. Em princípio, o esquema de *optimistic unchoke* seleciona um (único) par de forma aleatória para o qual enviará blocos quando solicitado. Em contraste, com *anti-snubbing* o par p_i monitora a quantidade de dados contribuida pelos pares beneficiados por upload e, caso não seja recebido nenhum bloco completo de p_j em 1 minuto, então o par é sufocado e utiliza-se a vaga dele para um *optimistic unchoke* adicional. Portanto, é possível que um um dado momento múltiplos *optimistic unchoke* sejam acionados porque os pares dos quais se pode esperar contribuição ficaram inativos.

Em relação a peças corrompidas, os agentes BitTorrent mais populares possuem mecanismos de contra-medida que punem pares que enviam peças corrompidas, denominados usualmente de "IP filters". A alternativa mais agressiva é banir todos os pares que contribuem para uma peça. Um problema com esse esquema é que um par não malicioso p_j que contribui para o par local com grande quantidade de dados tem maior chance de ser punido. Isso prejudicaria o próprio par local p_i. Um mecanismo mais sofisticado, adotado no Azureus, consiste em monitorar quantas vezes cada par contribui para peças corrompidas, e comparar com a quantidade de peças corretas que este par tem enviado.

3. Algoritmo de Rotação de Pares

Esta seção descreve o algoritmo de contra-medida, denominado "Rotação de Pares", que busca mitigar o impacto de um ataque de "mentira em massa". Em termos gerais, o algoritmo executando em p_i busca identificar os pares que constantemente permanecem "inativos", sem enviar blocos nem solicitar seu envio para p_i. Tais pares são considerados "suspeitos" e temporariamente desconectados por p_i. As conexões disponibilizadas podem ser usadas para estabelecer conexões com outros pares ou aceitar novas conexões requisitadas, conforme o valor corrente de $|A|$.

O algoritmo em p_i apenas entra em ação em situações que poderia trazer algum benefício para o p_i: pode valer a pena rotacionar pares inativos somente se todas as conexões liberadas forem preechidas. Quando o par não conhece outros pares que seriam potenciais candidatos a substituir os atuais, de nada adiantaria desconectar os pares suspeitos. Adicionalmente, se há diversas vagas livres de conexão em A, é menor a motivação para desconectar pares suspeitos e liberar vagas adicionais. Refletindo isso, como regra geral define-se que o número de pares em A deve ser pelo menos $\frac{3}{4}A_{min}$ para que rotações possam ocorrer.

Pares se tornam suspeitos quando constantemente deixam de fazer upload e download de dados. Um par p_i monitora a troca de dados com cada p_j desde o momento em que a conexão foi estabelecida com o mesmo. O total de blocos trocados com p_j durante a conexão atual, representado como d_j, é dividido pelo tempo decorrido desde o início da conexão do par, t_j, que gera uma taxa, $r_j = \frac{d_j}{t_j}$. Estes valores não persistem entre as conexões de um mesmo par. Um par pode permanecer conectado um tempo antes de sua taxa ser avaliada; denotado como t_{min}, este tempo é igual para todos os pares. r_{min} representa a taxa mínima que um par p_j deve honrar com p_i antes de tornar-se suspeito por p_i.

Portanto, quando $t_j \geq t_{min} \wedge r_j < r_{min}$, p_j é considerado suspeito. Como mencionado anteriormente, um par suspeito não necessariamente será desconectado. Mas quando isto acontecer, pares desconectados por p_i são colocados em quarentena, deno-

tado pelo conjunto Q_i (inicialmente vazio). Conexões recebidas por p_i de pares em Q são rejeitadas. O tempo que um par deve ficar em Q, dado por $cq_j \in \mathbb{R}, cq_j \geq 1$, é variável e cresce geometricamente de acordo com um fator f, conforme p_j tornar-se um suspeito. Inicialmente, cq_j é definido de acordo com o tempo global de quarentena, mas, assim como os demais parâmetros, é definido para cada par individualmente. O tempo restante de quarentena de p_j é dado por $q_j \in \mathbb{N}, q_j \geq 0$. Quando um par entra em Q, q_j assume o valor corrente de cq_j. Quando p_j deixa Q, p_i permite conexões com p_j, mas isto não necessariamente ocorre. O Algoritmo 1 mostra o pseudo-código da contra-medida.

Algoritmo 1 Rotação de Pares: a cada T, p_i avalia pares conectados, $p_j \in A$ (em A_i).

```
 1: for all p_j ∈ Q do
 2:     q_j ← q_j − 1
 3:     if q_j = 0 then
 4:         Q ← Q \ {p_j}
 5:     end if
 6: end for
 7: a ← |P \ (A ∪ Q)|
 8: for all p_j ∈ A, ordenado por r_j do
 9:     if (t_j ≥ t_min ∧ d_j/t_j < r_min) ∧ (|A| > A_min ∨ (|A| ≥ ⌊3/4 A_min⌋ ∧ a > 0)) then
10:         A ← A \ {p_j}
11:         Q ← Q ∪ {p_j}
12:         q_j ← ⌊cq_j⌋
13:         cq_j ← cq_j × f
14:         if |A| < A_min then
15:             a ← a − 1
16:         end if
17:     end if
18: end for
19: while |A| < A_min ∧ P \ (A ∪ Q) ≠ ∅ do
20:     p_k ← ∀p_j ∈ P \ (A ∪ Q)
21:     A ← A ∪ {p_k}
22:     t_k ← 0
23:     d_k ← 0
24: end while
```

Nas linhas 1-6 do Algoritmo 1, atualiza-se o tempo de quarentena restante para cada par $p_j \in Q$, removendo cada par p_j desse conjunto quando sua contagem chega a 0. Na linha 7, define-se a, uma variável que é inicializada com o número de pares que p_i "conhece" e que são "candidatos" a assumirem o lugar de pares que venham a ser rotacionados. Para tal, não podem estar já conectados a p_i nem podem constar da quarentena.

As linhas 9-18 correspondem ao processo de análise dos pares em A, potencialmente desconectando e colocando em quarentena uma parcela dos mesmos. Neste processo, A é percorrido de forma ordenada de maneira crescente de acordo com a taxa de dados trocada com cada par; ou seja, os primeiros pares são candidatos mais óbvios a serem rotacionados do que os últimos. Na linha 9, primeiro avalia-se a taxa de troca com p_j: se o par ainda não tem tempo de conexão suficiente ou contribui o necessário, então não deve ser rotacionado. Caso contrário, avalia-se ainda a situação em relação à quantidade de pares conectados, lembrando que rotações se tornam mais úteis a medida que o número de pares conectados cresce. Portanto, uma de duas condições suficientes para ir em frente

com a rotação do par: (i) enquanto o número de pares conectados superar A_{min}, pares podem ser desconectados (reduzindo $|A|$) de forma a oportunizar conexões iniciadas por outros pares remotos; (ii) enquanto houver em P um par candidato para o qual p_i poderia solicitar uma conexão ($a > 0$) e o número de pares conectados for igual ou superior a $\frac{3}{4}A_{min}$. As linhas 10 e 11 se referem à desconexão de p_j e sua inclusão na quarentena, respectivamente. A seguir, o tempo de quarentena a ser cumprido por p_j é configurado (linha 12) e aumentado segundo o fator f (linha 13). Nas linhas 14-16, decrementa-se a caso conte-se com um par candidato para assumir o lugar de p_j.

Nas linhas 19-24 do Algoritmo 1, p_i toma a iniciativa de abrir conexões de forma a completar A, até no máximo A_{min} pares. O restante das vagas em A_i permanece disponível a pares que solicitarem conexões com p_i. Não há garantia, contudo, que as conexões se concretizem, por uma série de razões, como o fato de pares $p_j \in P$ recusarem a conexão pois $|A_j| = A_{max}$ ou por já terem deixado o enxame. Na linha 20, escolhe-se um par conhecido qualquer mas que não esteja nem conectado nem na quarentena, conectando ao mesmo (linha 21) e reiniciando seus contadores (linhas 22-23).

Em resumo, o benefício desta contra-medida está em substituir pares remotos potencialmente maliciosos, não interessados em trocar dados com p_i, por outros pares possivelmente interessados. A eficácia e eficiência do algoritmo são avaliadas na Seção 5.

4. Algoritmo de Anti-corrupção

Esta seção descreve uma contra-medida adequada ao ataque de corrupção. Um mecanismo de contra-medida deve tentar identificar qual(is) par(es), dentre o conjunto de pares que contribuíram com blocos de uma determinada peça, foram os que enviaram bloco(s) corrompido(s).

Para identificar pares corruptores, emprega-se uma estratégia baseada em *reputação*. Quando os pares são identificados ou suspeitos de contribuírem com blocos corrompidos para uma peça, eles têm sua reputação diminuída. Por outro lado, quando um par contribui com blocos de uma peça íntegra, sua reputação é aumentada. Quando a reputação de p_j, na visão de p_i, atinge um limiar mínimo, p_j é desconectado de p_i e colocado em quarentena, assim como no caso anterior. No contexto deste trabalho, considera-se apenas o caso em que o par é enviado em definitivo à quarentena.

Diferentemente da contra-medida anterior, periódica, esta é orientada a evento, sendo ativada quando a transferência de uma peça é completada. Quando uma peça se mostra corrompida, para evitar que o par tenha de transferir novamente todos os blocos $b_{x,*}$ desta peça, o mecanismo tenta reconstruí-la buscando carregar novamente apenas os blocos corrompidos. No melhor caso, há apenas um único destes blocos na peça. Para identificar quais são os possíveis blocos corrompidos, o mecanismo assume as seguintes premissas básicas:

- um par honesto p_j, ao enviar blocos de uma peça b_x para p_i, tende a colaborar com mais blocos da mesma peça (pois p_j não está sufocando p_i e possui b_x);
- um par malicioso, por dispor de recursos limitados de largura de banda, tentará otimizar o ataque enviando apenas um único bloco corrompido por peça e em seguida irá sufocar p_i, ou voluntariamente se desconectar.

O Algoritmo 2 apresenta o pseudo-código para esta contra-medida. A reputação de um par p_j, na visão de um par p_i, é denotada como o_i^j (de "opinião"), com $o \in [0:1]$.

Algoritmo 2 Algoritmo de contra-medida para ataques de corrupção

```
1:  if h_x = hash(b_x) then
2:     for all p_i ∈ R do
3:        o_i ← min(o_i + δ_inc, 1)
4:     end for
5:  else
6:     b'_x ← b_x
7:     R ← U_x
8:     B ← {b_{x,y} | u_{x,y} ≠ p_c}
9:     while ¬h'_x ∧ B ≠ ∅ ∧ Unchoked(p_c) do
10:       b'_{x,y} ← requests any b'_{x,y} ∈ B from p_c
11:       B ← B \ {b'_{x,y}}
12:    end while
13:    if h'_x = hash(b'_x) then
14:       o_c ← min(o_c + δ_inc, 1)
15:       R' ← {u_{x,y} | b_{x,y} ≠ b'_{x,y}}
16:       for all p_i ∈ R' do
17:          o_i ← max(o_i − δ_dec, 0)
18:       end for
19:       b_x ← b'_x
20:    else
21:       if p_c ∉ A ∨ ¬Unchoked(p_c) then
22:          o_c ← max(o_c − δ_dec/2, 0)
23:          b_x ← b'_x
24:       else
25:          o_c ← max(o_c − 2 × δ_dec, 0)
26:       end if
27:    end if
28:    for all p_i ∈ R do
29:       if o_i = 0 then
30:          A ← A \ {p_c}
31:          Q ← Q ∪ {p_i}
32:       end if
33:    end for
34: end if
```

No algoritmo, o primeiro passo é verificar o *hash* h_x da peça b_x; caso consistente (linhas 2-4), a reputação dos pares que contribuíram para a peça é aumentada em um delta, definido como δ_{inc}, respeitando-se o valor máximo de opiniões. O caso de peça corrompida é comentado a seguir.

Na linha 6, a peça original é salva por duas razões: seus blocos ajudam na recuperação da peça e na identificação de quais blocos, especificamente, estavam corrompidos. Na linha 7, forma-se um conjunto (R) com todos os pares que contribuíram para a peça; para tal, denota-se como $u_{x,y}$ o par que enviou o bloco $b_{x,y}$, e U_x o conjunto de pares que contribuiram para a peça b_x. Dentre os pares desse conjunto, possui papel especial para a contra-medida o par que contribuiu com o bloco que completou a peça, a ser denotado como p_c. Na linha 8, forma-se um conjunto com todos os blocos que *não* foram enviados por p_c. A cada interação do laço das linhas 9-12, p_i escolhe um bloco em B, carrega o mesmo de p_c (linha 10) e então o remove de B (linha 11), até que uma de três condições se verifique: (i) a peça é recuperada, como indicado pela função de *hash* da peça; (ii)

todos os blocos originalmente em B foram (re)carregados de p_c, quando $B = \emptyset$; (iii) p_c sufocou p_i ou desconectou, impossibilitando a carga de novos blocos a partir de p_c.

Na linha 13 verifica-se se a peça foi recuperada. Caso sim, na linha 14 aumenta-se a reputação de p_c, por ter garantidamente fornecido apenas blocos corretos. Na linha 15, cria-se um conjunto com todos os pares que forneceram blocos para b_x diferentes do bloco correspondente na peça correta, $b_x^{'}$. Nas linhas 16-18, a reputação destes pares é diminuída. Na linha 19 o conteúdo da peça recuperada $b_x^{'}$ é copiado sobre a peça original. As linhas 21-26 representam a situação em que não foi possível recuperar a peça, conforme comentado a seguir.

A linha 21 verifica se o laço foi encerrado porque p_c desconectou ou sufocou p_i; nestes casos, a reputação de p_c é diminuida, embora mais suavemente pois significa uma suspeita (linha 22). Mesmo o par p_c tendo o direito de sufocar p_i, a "punição" é aplicada visto que a estratégia de não cooperação pode ser aplicada por um par malicioso. Mesmo que a peça não tenha sido recuperada, sua nova versão é restaurada sobre a peça original (linha 23), esperando-se que a antiga tenha um grande número de blocos corretos. A contra-medida procura neste caso ter menos pares contribuindo para determinada peça, mas cada par com mais blocos. Na linha 25, a reputação de p_c é fortemente diminuida, visto que as evidências são fortes: p_c foi o único par a contribuir com a peça e ela continua corrompida.

Por fim, nas linhas 28-33, o algoritmo verifica a reputação de todos os pares que que participaram da peça corrompida. Se algum par atingir o limiar 0 ele é desconectado e colocado em quarentena (como mencionado anteriormente, no presente contexto, isto significa banir o par).

Em resumo, a contra-medida busca identificar os pares responsáveis por corromper peças com blocos incorretos, associando uma reputação a cada par. Em princípio, um único evento não provoca a ida de um par à quarentena. Um par que contribui para peças corretas terá sua reputação aumentada. Em contraste, quando um par se comporta constantemente de forma (aparentemente) maliciosa, ou seja, sua participação está ligada apenas a peças corrompidas, sua reputação decrescerá e o par será desconectado.

5. Avaliação dos Algoritmos de Contra-medida

Os algoritmos apresentados nas Seções 3 e 4 foram implementados em um simulador de redes BitTorrent. Tal simulador imita em detalhe a comunicação em nível de mensagens e foi validado experimentalmente com enxames reais em ambiente controlado [Barcellos et al. 2008, Mansilha et al. 2008]. Através de uma campanha de simulações, avaliamos a *eficácia* das contra-medidas perante ataque e a *eficiência* em situações sem ataque. Esta seção apresenta os resultados obtidos com esses experimentos. O objetivo da avaliação é responder as seguintes questões fundamentais:

- Q1. Qual o impacto dos ataques de "mentira em massa" ou corrupção sobre um enxame, comparando-se então os casos *sem* ataque e *com* ataque?
- Q2. Qual a eficácia das contra-medidas propostas ao lidar com as situações de ataque citadas na questão anterior, comparando-se aos casos de ataque sem e com contra-medida? Além disso, comparando o caso com ataque e com contra-medida, quão distante ele fica do caso ótimo, representado pelo cenário sem ataque e sem contra-medida?
- Q3. Qual a eficiência das contra-medidas, em termos de atrasos potencialmente induzidos com a introdução de uma contra-medida?

Q4. Os mecanismos de detecção das contra-medidas conseguem logo identificar aqueles pares que são maliciosos e corretamente?

A seguir, são apresentados os cenários de avaliação, as métricas coletadas, as premissas assumidas e escolha de parâmetros para o protocolo.

5.1. Modelo de avaliação

O conjunto de cenários avaliados e discutidos neste artigo refere-se à fase inicial do enxame, quando a taxa de chegada é inicialmente alta mas decresce exponencialmente. Os experimentos consideram um semeador inicial e a chegada de 250 sugadores exponencialmente distribuídos em 60 min, com uma concentração por volta dos primeiros 10 min. O semeador é permanente e não possui recursos ou capacidades especiais.

As avaliações foram feitas tomando-se as seguintes métricas de interesse: "tempo de download gasto por cada par" e "tempo necessário para que todos os pares honestos obtenham peças". A primeira refere-se ao sucesso do download. Onde, $f(x)$ é uma função monotônica não-decrescente que mostra quanto tempo é necessário para que x pares completem download, com $x \in \mathbb{N}$ e $x \in [1:250]$. É possível que, sob condições de ataque, um ou mais pares falhem em completar o download. É considerado que o *enxame falhou* (parcialmente) se um ou mais pares falham em completar o download em "tempo hábil".

A segunda métrica avalia quanto de um download os pares conseguiram completar. Do ponto de vista do usuário, a falha com 1% de peças pode ser muito diferente de uma falha com 99% de peças. Se o conteúdo pode ser útil parcialmente (i.e. compressão de centenas de fotos em alta resolução), é melhor possuir 99% do que 1%. Em contraste, existem casos onde é necessário possuir o conteúdo completo (i.e. atualização do sistema operacional a ser aplicado a milhões de computadores) e 1% é melhor que 99% porque no segundo caso foram desperdiçados muito mais recursos. Para fins de avaliação, foi adotado o primeiro caso, mas isto não significa que este seja mais comum que o segundo caso. Logo, o tempo necessário para completar o download (y) e a quantidade global de peças (x) refletem a evolução do enxame, e definem uma função $f(x)$ monotônica não-decrescente. Resultados são apresentados para ambas as métricas apresentadas.

Os parâmetros da modelagem de simulação podem ser organizados em categorias: (i) ambiente, (ii) protocolo BitTorrent, (iii) ataque, e (iv) contra-medida. As duas primeiras categorias são comuns a todos os casos, pois independem de ataques e contra-medidas. As larguras de banda são apresentadas sempre em pares, representando download/upload. O *ambiente* é definido como: número de sugadores, 250; número de semeadores iniciais, 1; tempo de chegada/taxa de chegada sugadores, exponencial; larguras de banda dos semeadores iniciais e sugadores, 1024 Kbps/256 Kbps; e larguras de banda do rastreador, 4096 Kbps/4096 Kbps.

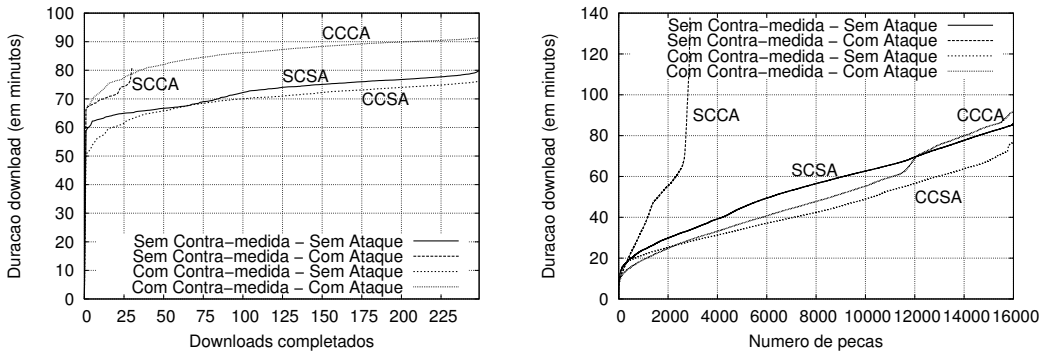
Por sua vez, a configuração do *protocolo BitTorrent* é definida como: número de peças, 64; tamanho de cada peça, 1 MB (64 blocos); taxa (*ratio*) alvo dos sugadores, 1.0; taxa (ratio) alvo dos semeadores iniciais, ∞; tempo máximo de semeadura para sugadores, ∞; número mínimo de conexões almejado (A_{min}), 30; número máximo de conexões (A_{max}), 50; número máximo de uploads concorrentes, 4; intervalo de execução do *optimistic unchoke*, 30s; número de entradas dedicadas ao *optimistic unchoke* (fora *anti-snubbing*), 1; intervalo de consulta ao rastreador, 10 min; número de pares retornados em L pelo rastreador, 50.

De acordo com o ataque "mentira em massa", os *sybils* são controlados por um único par malicioso, compartilhando a informação dos P entre o *sybils*. Cada novo *sybil* inicia uma conexão com o rastreador, registra-se e recebe uma nova lista L. Esta lista é adicionada ao conjunto de pares conhecidos, removendo os próprios *sybils* da lista: $P \leftarrow P \cup (L \setminus M)$. Os parâmetros que descrevem um *ataque* são definidos como segue. Para o ataque de "mentira em massa": número de pares atacantes, 500 *sybils*; tempo de chegada do atacante, 0.1s (logo após o semeador inicial); taxa de criação de *sybils* pelo atacante, 1 *sybil*/3s; larguras de banda do par malicioso (comum aos *sybils*), 8 Mbps/8 Mbps; número de peças mentidas, 16. Em relação ao ataque de corrupção, os pares maliciosos chegam a cada 3s a partir do tempo 0; o intervalo entre transmissões de mensagens UNCHOKE por cada par atacante é 4s. A escolha dos parâmetros baseou-se um estudo experimental [Schmitt et al. 2008] com enxames populados com agentes Azureus, visando aumentar o impacto dos ataques.

Por fim, os parâmetros assumidos para as *contra-medidas* são listados a seguir. A contra-medida de Rotação foi avaliada com as seguintes escolhas: periodicidade de avaliação do algoritmo (T), 1 min; tempo inicial de quarentena, 4 ciclos de avaliação; tempo de carência antes de avaliar taxa de troca com um par (T_{min}), 5 min; fator geométrico de crescimento quarentena (f), 2.0; taxa de transferência mínima exigida (R_{min}), 0.2 Kbps. Os parâmetros para a contra-medida Anti-corrupção são: reputação inicial de pares (o_i^j), 0.5; fator de decremento base a ser aplicado perante evidência negativa (δ_{dec}), 0.2; fator de incremento base para evidências positivas (δ_{inc}), 0.1.

5.2. Avaliação da Rotação de Pares

Esta subseção apresenta a avaliação da eficácia e eficiência da Rotação de Pares. A Figura 1 apresenta os resultados. Na Figura 1(a), os eixos x e y são, respectivamente, o número de *downloads completados* por pares honestos, e tempo de download em minutos. O enxame é considerado completado com sucesso quando todos os 250 pares acabam o download, ou seja, a curva alcança $x = 250$. Para ilustrar, a curva "Sem Contra-medida-Com Ataque" significa que o enxame falhou com 30 pares completos no tempo 80.8 min. A Figura 1(b) mostra no eixo x o número global de peças corretas já recebidas, enquanto o eixo y mostra o tempo necessário para fazer o download dessas peças. Em ambos existem quatro curvas refletindo os casos de interesse: "Sem Contra-medida – Sem Ataque" (SCSA), "Sem Contra-medida – Com Ataque" (SCCA), "Com Contra-medida – Sem Ataque" (CCSA), e "Com Contra-medida – Com Ataque" (CCCA).



(a) Pares com downloads completados (b) Contador global de peças corretas recebidas

Figura 1. Avaliação de Rotação de Pares em cenários com e sem ataque

A comparação de ambas as curvas sem contra-medidas, *com* vs. *sem* o ataque "mentira em massa", ajuda a responder a primeira questão, Q1, sobre o impacto de um ataque sem proteção. Como é mostrado na Figura 1(a), a curva SCSA indica que o primeiro par termina o download por volta de 60 min, e o último por volta de 80 min. Em contraste, a curva SCCA, ilustrando os efeitos de um ataque sem contra-medida, mostra que cerca de 30 pares completam o download entre 68 e 82 min; os 220 pares restantes falham por causa do ataque. Em termos de peças completas, a Figura 1(b) destaca, através de SCCA, o impacto do ataque. Os pares alcançam cerca de 3000 peças em 70 min, e depois disso é realizado pouco avanço. Por volta de 138 min, o enxame falha. Em resposta à questão Q1, existe um *grande* impacto quando é lançado um ataque "mentira em massa" com 500 *sybils* sobre um enxame com 250 pares honestos sem a proteção de uma contra-medida.

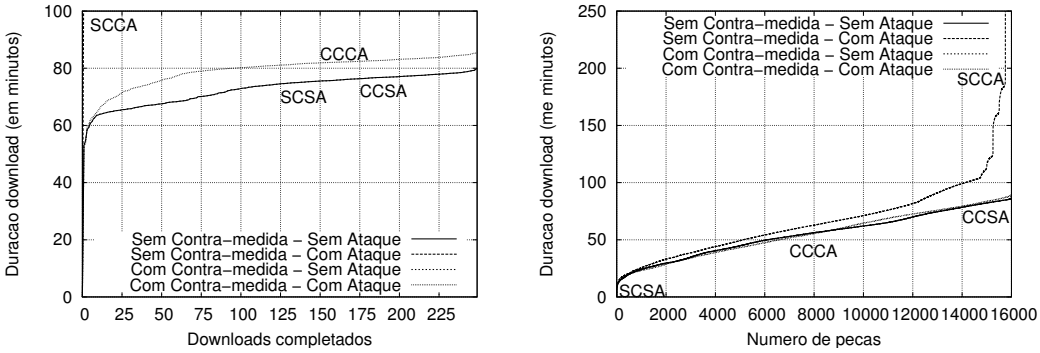
A questão Q2, sobre a eficácia da contra-medida, é respondida comparando-se na Figura 1(a) as curvas que representam cenários com ataque: SCCA e CCCA. Em CCCA todos os 250 pares completam o download; o 250^o par termina aos 93 min, apenas 13 min depois do caso ideal SCSA. Examinando o mesmo par de curvas, SCCA e CCCA, na Figura 1(b), fica claro que a contra-medida (CCCA) mitiga a maior parte do impacto negativo do ataque. Todos os downloads são concluídos em tempo hábil, antes dos 93 min. Em comparação com o caso ideal, SCSA (80 min), o algoritmo proposto (CCCA) atua bem (16% mais lento quando o número de *sybils* é igual ao dobro do número de pares honestos).

Para responder a Q3, sobre uma possível sobrecarga gerada pela contra-medida de Rotação, compara-se na Figura 1(a) as curvas sem ataque (SCSA e CCSA). É possível antecipar que o algoritmo da Rotação pode introduzir alguns atrasos, devido a decisões erradas levando a rotação de pares honestos que apenas estavam inativos. Contudo, surpreendentemente, os resultados obtidos com a Rotação (CCSA) são um pouco (3-5 min), mas constantemente, superior ao BitTorrent sem a contra-medida (SCSA). O motivo para esta vantagem nos downloads pode ser confirmada na Figura 1(b): a curva CCSA é mais baixa (mais rápida) que SCSA por uma margem de até 20 min. Outros cenários foram explorados e resultados similares foram encontrados. A explicação para esta melhoria reside no fato de que o algoritmo pode rotacionar até $A_{max} - \frac{3}{4}A_{min}$ piores pares (com menor taxa de envio). As vagas livres em A são destinadas aleatoriamente a novos pares e/ou deixadas em aberto para novas conexões, permitindo que novos e melhores pares sejam encontrados.

5.3. Avaliação da Anti-corrupção

As três questões da subseção anterior são discutidas aqui em relação à contra-medida Anti-corrupção. A Figura 2 apresenta um par de gráficos onde um ataque de corrupção é lançado com 15 atacantes, cada um enviando um bloco corrompido a cada 4s. Os eixos *x* e *y* possuem o mesmo significado que nos gráficos anteriores.

O primeiro ponto que se destaca na Figura 2(a) é a curva representando o tempo de downloads completados (SCCA): ela cresce rapidamente para mais de 100 min já após os primeiros downloads (cerca de 3). Todos os outros 247 pares tentam por 600 min mas falham em completar o download. Em contraste, em um enxame "normal" (SCSA) todos os pares completam seus downloads em até 80 min. Em outras palavras, respondendo a Q1, o ataque é *devastador*. Em relação às peças, o contraste entre SCSA e SCCA não é tão surpreendente na Figura 2(b). Apesar do ataque, os pares honestos conseguem receber



(a) Pares com downloads completados (b) Peças corretas recebidas

Figura 2. Avaliação da Anti-corrupção

quase todas as 16000 peças, embora em uma taxa inferior (lembre que os pares descartam peças corrompidas e executam o download novamente). Os pares conseguem obter as 12000 primeiras peças em uma taxa razoavelmente boa; contudo quando há menos peças para serem obtidas, os pares honestos tornam-se "alvos" fáceis para os pares maliciosos. O resultado final é que as últimas 1000 peças no enxame não podem ser obtidas corretamente; os pares honestos não fazem progresso, e os downloads convergem para 96-99% de conclusão.

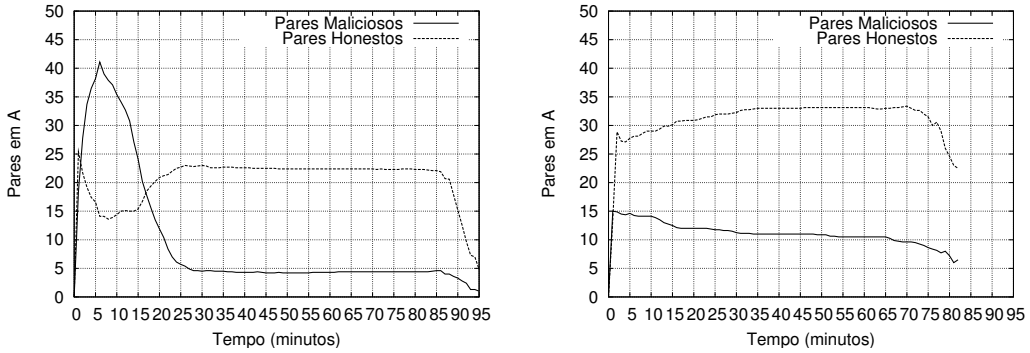
Para responder a Q2, compara-se na Figura 2(a) ambas as curvas com ataque: SCCA – já discutida – e CCCA. Enquanto pouquíssimos pares completam SCCA, já em CCCA todos os pares completam com sucesso em até 86 min. Também observa-se que a curva CCCA permanece muito próxima (cerca de 10 min) de SCSA. A comparação demonstra que o algoritmo proposto é tanto eficaz (com 100% de downloads) como eficiente (completando com tempo próximo do ideal).

Finalmente, em relação a Q3, nota-se na Figura 2(a) que as curvas referentes aos cenários sem ataques (SCSA e CCSA) são sobrepostas. Em outras palavras, apesar de algumas variações com respeito ao download de peças mostrado na Figura 2(b), as taxas de sucesso e os tempos de download são idênticos. Em resumo, a Anti-corrupção é eficiente e não introduz sobre-carga quando não há ataque. Este era o resultado esperado, uma vez que o algoritmo Anti-corrupção não entra em ação a menos que peças corrompidas sejam recebidas.

5.4. Precisão dos mecanismos de detecção

Esta seção procura responder a quarta e última questão, Q4: na Rotação e na Anti-corrupção, os algoritmos conseguem identificar os pares maliciosos corretamente? Em cada contra-medida, foi registrado a cada minuto, para cada par honesto ($p_i \in H$), o número de pares maliciosos bem como de pares honestos em A_i, e obteve-se as médias que foram divididas pelo número de pares. A Figura 3 ilustra a população média de pares em A ao longo do tempo, sendo a média obtida entre todos pares honestos (no caso do ataque de corrupção, semeadores são ignorados).

Em relação ao ataque "mentira em massa", os resultados na Figura 3(a) indicam que nos primeiros momentos do enxame o número médio de maliciosos conectados aumenta rapidamente até 42 pares. Em contraste, o número de honestos cresce para 25 pares no primeiro minuto, mas cai para 13 pares nos minutos subseqüentes. Por volta dos 7 min,



(a) Rotação de Pares (b) Anti-corrupção

Figura 3. Eficácia das contra-medidas em detectar pares maliciosos

a proporção de maliciosos para honestos atinge seu nível mais alto. A partir do 7 min, até os 30 min, a Rotação corretamente identifica e desconecta os pares maliciosos. A Rotação de Pares é eficaz uma vez que mantém em 5, ou menos, a média de pares maliciosos conectados aos honestos. Por volta dos 85 min, os pares honestos começam a deixar o enxame em massa, levando a uma queda acentuada no número de honestos. Contudo, o número de pares maliciosos também diminui, pois os pares que tinham conexões com os maliciosos também deixam o enxame.

A Figura 3(b), por sua vez, mostra curvas para a média de pares maliciosos que estão conectados com cada par honesto através do tempo. Como o ataque de corrupção é realizado com um número menor de atacantes, diferente do caso anterior, inicialmente o número de honestos ultrapassa o número de maliciosos (há um total de 15 atacantes). A partir do terceiro minuto, a Anti-corrupção detecta e isola pares maliciosos. A curva de maliciosos diminui suavemente até 7 pares, por volta de 82 min. Inversamente, as vagas em A são liberadas pela contra-medida, novas conexões são estabelecidas (recebidas e/ou solicitadas) com pares honestos, enquanto os maliciosos permanecem em quarentena. Note que nenhum par honesto foi colocado em quarentena, de modo que a contra-medida proposta é bastante eficaz em detectar e isolar pares corruptores.

6. Conclusões e Trabalhos Futuros

BitTorrent é um protocolo de compartilhamento de arquivos que permite disseminar arquivos para um grande número de usuários. Em [Konrath et al. 2007] foi identificado e avaliado o impacto de ataques DoS em BitTorrent, mostrando que atacantes com poucos recursos podem causar danos significativos em enxames. Neste trabalho, foram propostos dois algoritmos de contra-medidas para atenuar o impacto desses ataques. Para tal, primeiro foi revisado o protocolo BitTorrent através de uma visão orientada a conjuntos e então descritas e discutidos os algoritmos propostos.

O princípio de funcionamento das contra-medidas é o mesmo: permitir que o par local possa detectar pares maliciosos (com os quais o par local esteja conectado) e desconectá-los, dando oportunidade para outros pares. Foram propostas as contra-medidas de Rotação de Pares e Anti-corrupção. Avaliou-se ambas as contra-medidas sob um conjunto de cenários. Os resultados mostraram que ambos os algoritmos propostos são eficazes e eficientes contra os ataques de “mentira em massa” e de corrupção.

Como trabalhos futuros, são propostas duas linhas de investigação. Primeira-

mente, será estudada a combinação das contra-medidas, avaliação de suas eficácias e eficiências em cenários sem ataque, com um dos ataques e com ambos os ataques. Em seguida, apesar do TorrentSim – usado em nossas avaliações – ter sido validado e prover resultados próximos da vida real, é imperativo que procure-se confirmar através de comparações com experimentos em redes reais. Já têm-se implementados ataques em agentes de usuários e atualmente se está implementando as contra-medidas.

Referências

(2007). Azureus website. http://azureus.sourceforge.net/.

Barcellos, M. P., Mansilha, R. B., and Brasileiro, F. V. (2008). Torrentlab: investigating bittorrent through simulation and live experiments. *IEEE Symposium on Computers and Communications (ISCC'08)*, pages 1–6.

Christin, N., Weigend, A. S., and Chuang, J. (2005). 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. ACM Press.

Cohen, B. (2003). Incentives build robustness in bittorrent. In *Proceedings of the 1st Workshop on the Economics of Peer-to-Peer Systems*, pages 116–121, Berkeley, CA.

Douceur, J. R. (2002). The sybil attack. In *1st International Workshop on Peer-to-Peer Systems*, pages 251–260, Cambridge, MA, USA.

Konrath, M. A., Barcellos, M. P., and Mansilha, R. B. (2007). Attacking a swarm with a band of liars: evaluating the impact of attacks on bittorrent. In *The Seventh IEEE International Conference on Peer-to-Peer Computing (IEEE P2P 2007)*. IEEE.

Liogkas, N., Nelson, R., Kohler, E., and Zhang, L. (2006). Exploiting bittorrent for fun (but not profit). In *5th International Workshop on Peer-to-Peer Systems (IPTPS 2006)*.

Locher, T., Moor, P., Schmid, S., and Wattenhofer, R. (2006). Free riding in bittorrent is cheap. In *Fifth Workshop on Hot Topics in Networks (HotNets-V)*, Irvine, CA, US.

Mansilha, R. B., Barcellos, M. P., and Brasileiro, F. V. (2008). Torrentlab: Um ambiente para avaliação do protocolo bittorrent. *XXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2008)*, pages 1–14.

Piatek, M., Isdal, T., Anderson, T., Krishnamurthy, A., and Venkataramani, A. (2007). Do incentives build robustness in bittorrent? In *Proceedings of 4th USENIX Symposium on Networked Systems Design & Implementation (NSDI 2007)*, Cambridge, MA. USENIX.

Schmitt, C., Barcellos, M. P., and Mansilha, R. B. (2008). Um estudo experimental sobre ataques ao sistema de compartilhamento p2p bittorrent. *XXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2008)*, pages 1–14.

Shneidman, J., Parkes, D., and Massoulie, L. (2004). Faithfulness in internet algorithms. In *Proc. SIGCOMM Workshop on Practice and Theory of Incentives and Game Theory in Networked Systems (PINS'04)*, Portland, OR, USA. ACM SIGCOMM.

Sirivianos, M., Park, J. H., Chen, R., and Yang, X. (2007). Free-riding in bittorrent with the large view exploit. In *6th International Workshop on Peer-to-Peer Systems (IPTPS 2007)*, Bellevue, WA, US.