Um Estudo Experimental sobre Ataques ao Sistema de Compartilhamento P2P BitTorrent

Carlos H. Schmitt[1], Marinho P. Barcellos[2], Rodrigo B. Mansilha[1]

[1]UNISINOS – Universidade do Vale do Rio dos Sinos
Av. Unisinos, 950 – São Leopoldo, RS – CEP 93.022-000

[2]PUCRS – Pontifícia Universidade Católica do Rio Grande do Sul
FACIN – Faculdade de Informática
Av. Ipiranga, 6681, Prédio 32
Porto Alegre, RS – CEP 90.619-900

carlos.schmitt@gmail.com, marinho@acm.org, rmansilha@gmail.com

Resumo. *BitTorrent é um dos principais protocolos par-a-par (P2P) para redes de distribuição de conteúdo. Este protocolo provou ser altamente escalável, porém há dúvida sobre a sua real robustez quando submetido a ataques com o objetivo de prejudicar a rede. Até então os estudos sobre o impacto de ataques explorando vulnerabilidades de segurança foram conduzidos através de simulações. Resultados experimentais obtidos avaliando-se implementações reais de BitTorrent permitem obter dados mais concretos do que via avaliação analítica ou simulações. Este artigo descreve o uso de um agente modificado para avaliação de impacto de ataques de Corrupção de Peças em um ambiente real, porém controlado, permitindo uma análise que é ao mesmo tempo fidedigna e reprodutível. Os resultados mostram que o ataque que descrevemos e suas variantes são efetivos, causando prejuízo significativo tanto em termos de taxa de download efetiva como em termos de sobrecarga de rede.*

Abstract. *BitTorrent is one of the main peer-to-peer (P2P) protocols for content distribution networks. Although found to be highly scalable in practice, there is still doubt on its robustness regarding attacks that aim to prejudice the network. To date, the studies on the impact of attacks exploiting security vulnerabilities were conducted through simulations. Experimental results obtained through actual implementations of BitTorrent allow more concrete data than analytical evaluation or simulations. This paper describes the use of a modified agent to evaluate the impact of Piece Corruption Attacks in a real, although controlled environment, leading to an analysis that is both accurate and reproducible. The results indicate that the attack we describe and its variants are effective against modern BitTorrent agents, causing significant harm both in terms of effective download rate and network cost.*

1. Introdução

BitTorrent é um protocolo de compartilhamento de arquivos Par-a-Par (P2P) que pode ser usado como tecnologia base de redes de distribuição de conteúdo em larga

escala. BitTorrent se tornou provavelmente a tecnologia de compartilhamento mais popular hoje em dia na Internet, e começa a ser adotado por empresas. Os estudos existentes sobre o impacto de ataques explorando vulnerabilidades de segurança no BitTorrent encontrados na literatura, [Konrath et al. 2007a, Mansilha et al. 2007], foram conduzidos através de simulações. Em tais análises, o protocolo foi modelado e implementado através de uma extensão do arcabouço de simulação Simmcast [Muhammad and Barcellos 2002]. O arcabouço foi empregado para avaliar o impacto de três ataques diferentes. Segundo aqueles estudos, o ataque que apresenta maior efetividade é o envio de blocos corrompidos, e por essa razão o trabalho enfatiza este ataque.

Embora esta simulação seja detalhada e se assemelhe à realidade, algumas premissas simplificatórias são inevitáveis. Na avaliação de sistemas computacionais em geral, é recomendado que a análise seja feita empregando metodologias complementares de maneira a aumentar a confiança nos resultados [Jain 1991]. Resultados experimentais obtidos avaliando-se implementações reais de BitTorrent permitem obter dados mais concretos e contrastar os resultados de simulação com os obtidos em cenários reais.

Este artigo descreve a implementação e avaliação de impacto de ataques ao BitTorrent em cenários reais. Outros trabalhos na literatura realizaram também estudos experimentais sobre o comportamento de BitTorrent, mas nenhum tratou especificamente de ataques que buscam prejudicar o andamento dos downloads dos usuários. Esta é a primeira contribuição nesse sentido.

O restante deste trabalho está organizado da seguinte maneira: a Seção 2 sintetiza os principais estudos relacionados à avaliação de ataques em BitTorrent. A Seção 3 apresenta o ataque implementado neste trabalho, Corrupção de Peças, como suas variantes (o conteúdo compartilhado via BitTorrent é organizado em peças). A implementação deste ataque bem como os resultados obtidos são apresentados na Seção 4 e 5, respectivamente. Por fim, a Seção 6 encerra o trabalho sintetizando as conclusões e descrevendo trabalhos futuros.

2. Trabalhos Relacionados

Os artigos relacionados a este podem ser enquadrados em duas categorias: aqueles que são similares em metodologia e aqueles que compartilham o mesmo objetivo. A primeira se refere a avaliações experimentais de BitTorrent. [Pouwelse et al. 2005] usa uma infraestrutura para coletar dados sobre enxames BitTorrent e inferir sobre seu compartamento típico. Os quatro trabalhos seguintes são similares em que usam uma implementação de novo agente ou agente alterado para demonstrar sua hipótese. [Shneidman et al. 2004] apresenta uma avaliação experimental em que um par egoísta busca aumentar sua taxa de download mentindo a posse de todas as peças do arquivo e enviando lixo aos pares remotos. [Locher et al. 2006] descreve um agente de usuário (BitThief) que atua como par carona puro e mesmo assim consegue boas taxas de download. [Piatek et al. 2007] apresenta BitTyrant, um agente que é usado para demonstrar limitações no mecanismo de justiça do BitTorrent. Este agente troca dados apenas com pares estrategicamente escolhidos e que possam maximizar sua taxa de download. Similarmente, os autores em [Sirivianos et al. 2007]

apresentam um novo agente que atua como um "par carona puro", além de se valer do *Large View Exploit*, *Sybil* e *Withewashing* como estratégias para lucrar mais.

Na segunda categoria estão os trabalhos que compartilham objetivo: investigam o impacto de ataques de segurança à rede BitTorrent, porém via simulação. Em [Konrath et al. 2007a], são apresentados os possíveis impactos de dois ataques (Mentira de Peças e Eclipse de Pares Corretos). Em [Mansilha et al. 2007], o modelo e a simulação foram aprimorados e usados para avaliar o ataque de Corrupção de Peças. Como resultado, estes estudos apontam que o protocolo BitTorrent está sujeito à ação de atacantes, sendo mais efetivo o ataque de corrupção de peças, descrito a seguir.

3. Ataque de Corrupção de Peças

Em uma rede BitTorrent ("enxame"), o conteúdo sendo compartilhado entre pares é dividido em "peças", e estas subdivididas em "blocos". A consistência dos dados compartilhados é assegurada pelo protocolo subjacente, TCP, e verificada através de um *hash* associado a cada peça[1]. O ataque de Corrupção de Peças consiste em um ou mais pares maliciosos enviarem blocos corrompidos aos pares da rede. Deste modo, o par atacado percebe a falha, descarta a peça a qual pertence o bloco corrompido, e deve obte-la novamente. Feito com muitas peças, prejudica o desempenho do download e aumenta o custo de rede, em termos de volume de dados transmitidos.

3.1. Contramedidas implementadas em agentes BitTorrent

Para defender-se desse ataque ou de pares defeituosos, certas implementações de BitTorrent possuem mecanismos de contramedida que localizam e bloqueiam/banem pares que freqüentemente enviam blocos corrompidos. Uma implementação simplista de contramedida utilizada por alguns agentes consiste em banir um par que tenha enviado integralmente uma peça corrompida. Apesar de eficaz, esta contramedida pode ser contornada de forma simples: nunca enviando a totalidade dos blocos. Neste caso, o par honesto será obrigado a obter o(s) bloco(s) restante(s) de outro par, e não terá certeza de qual par enviou as partes corrompidas.

O Azureus, assim como o μTorrent, implementa um método de defesa mais sofisticado. A cada falha de *hash*, todos os IPs dos pares que contribuíram com pelo menos um bloco desta peça são colocados em uma lista de suspeitos. O número de vezes que um IP se comporta como suspeito é monitorado, e quando chega a um limiar, o IP é colocado em uma lista negra. Diferentemente do método anterior, um par malicioso pode ser detectado mesmo se não enviar todos os blocos de uma peça.

Apesar de mais elaborado, este método tem duas limitações, conforme a seguir. Contribuindo para peças juntamente com pares maliciosos, pares honestos podem ser considerados suspeitos e eventualmente banidos. A probabilidade de banimento de pares corretos – falsos negativos – aumenta com a proporção de pares maliciosos no enxame. Mesmo que o par honesto não participe das mesmas peças que o par malicioso, ele poderá ocasionalmente participar de uma peça com outro par malicioso, e então passar de suspeito a banido.

[1]optamos por não repetir aqui explicações sobre BitTorrent constantes de nossas publicações anteriores; vide [Konrath et al. 2007b].

Além da imprecisão em detectar pares corruptores, esse mecanismo de banimento com base no IP pode levar à exclusão de pares honestos que compartilham IPs com pares corruptores, em função de NAT (*Network Address Translation*). Por exemplo, no caso de haver um par malicioso em meio à rede de um provedor de Internet, um conjunto de pares honestos que são usuários deste provedor podem ser banidos, o que aumenta o número de "falsos negativos" desta contramedida.

De forma mais ampla, estudamos implementações de agentes BitTorrent de forma a compreender que medidas de segurança estão disponíveis hoje. A documentação sobre contra-medida ao ataque de corrupção (popularmente denominados "filtros de IP") é escassa ou inexistente. Em virtude disto, foram feitos experimentos com as implementações mais populares, no sentido de averiguar *efetivamente* a presença de um mecanismo de contramedida. Os resultados obtidos são apresentados na Tabela 1. A mesma é auto-explicativa à exceção de um aspecto, a razão, que indica se a contramedida pesa a razão entre peças corrompidas e peças corretas de que um par participou. Nota-se ainda que apenas três agentes possuem algum mecanismo de contramedida. No caso do BitTornado, apesar de possuir a opção de banir pares que enviam dados corrompidos, a implementação não detectou nem baniu o par malicioso.

Percebe-se que a maioria dos agentes analisados não bane pares que enviam blocos corrompidos. Nas exceções estão o Azureus e μTorrent, mas que permitem o bloqueio de pares honestos em meio aos maliciosos, tanto por compartilharem uma mesma peça quanto por estarem atrás de um mesmo NAT. Por não haver um mecanismo de defesa padronizado ou amplamente aceito, neste trabalho não será considerada nenhuma contramedida para a realização dos experimentos.

Tabela 1. Medidas de Proteção por Filtro de IPs

Agentes	μTorrent	Azureus	Mainline	BitTornado	BitComet	BitLord	KTorrent
Filtro IP?	✓	✓	×	×	×	×	✓
Requer toda peça?	×	×	✓	✓	✓	✓	✓
Peças necessárias p/banir	6	3	×	×	×	×	1
Bane pares honestos?	✓	✓	×	×	×	×	×
Utiliza ratio?	×	✓	×	×	×	×	×

Nas próximas subseções serão apresentadas duas variações do ataque de Corrupção de Peças, Sufocamento e Desconexão.

3.2. Sufocamento

Nesta versão do ataque, o par malicioso anuncia-se como semeador a um par remoto e então habilita o mesmo a pedir peças. Quando requisições a blocos são recebidas, o malicioso envia um bloco apenas, e corrompido, e então envia uma mensagem de **Choke**. Assim o par atacado precisa buscar o restante da peça com outros pares. Quando a peça é completada, a mesma se mostra corrompida e precisa ser recarregada.

O diagrama de tempo na Figura 1 ilustra o ataque, assumindo-se para tal três simplificações. Primeiro, é mostrada apenas a troca de mensagens entre um rastreador e três pares, denominados Par Malicioso, Par Honesto 1 e Par Honesto 2. Segundo, foi

dado foco às mensagens trocadas pelo Par Honesto 1, que será o par atacado. Terceiro, a troca de mensagens realizada no **handshake** foi simplificada (**Handshake + Bitfield**). Os passos são os seguintes:

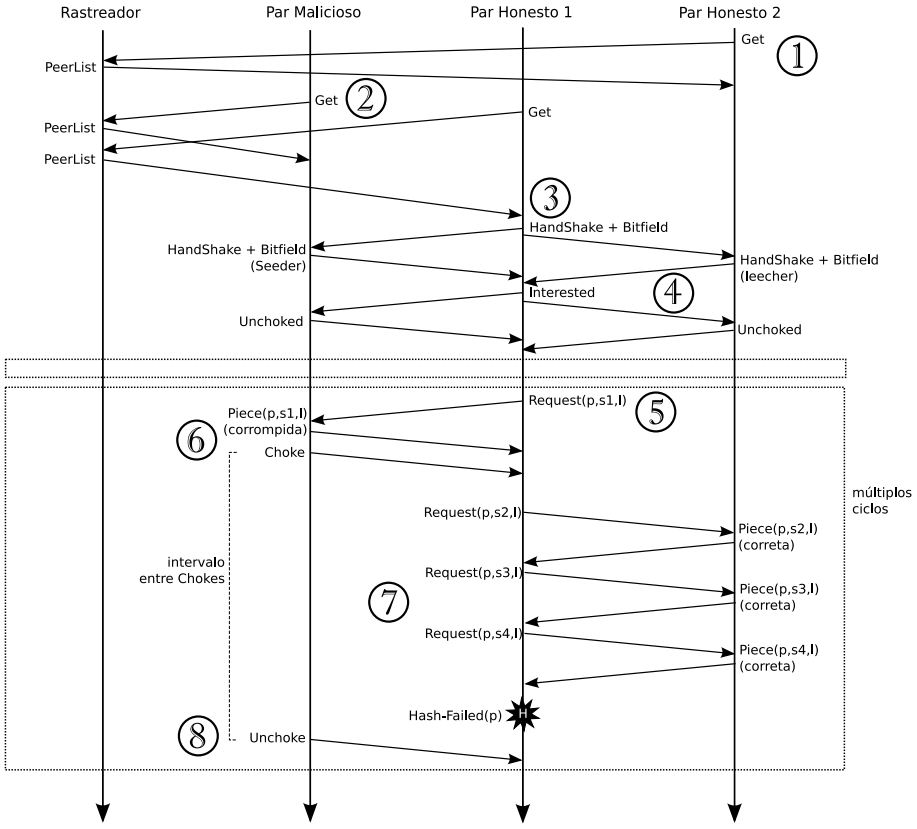


Figura 1. Diagrama de mensagens do ataque de Envio de Blocos Corrompidos.

(1-2) Par Honesto 2, e então Par Malicioso e Par Honesto 1, abrem conexões com o rastreador e obtém lista de endereços de pares que estão na rede, **PeerList**;

(3) no momento em que Par Honesto 1 recebe a **PeerList**, ele envia uma mensagem de **handshake** para Par Malicioso e logo em seguida a Par Honesto 2. Ao receberem o pedido de **handshake**, ambos os pares aceitam estabelecer a conexão respondendo também com a mensagem **Handshake + BitField**. Ao informar seu **BitField**, o Par Honesto 2 mostrou ser um sugador (*leecher*), enquanto o Par Malicioso mentiu a posse de todas as peças e informou ser um semeador (*seeder*);

(4) Par Honesto 1 envia mensagens de **Interested** para ambos os pares mostrando seu interesse em peças disponibilizadas por estes. Ela é respondida pelos dois pares remotos com **Unchoked**;

(5) após algum tempo, simbolizado pelo retângulo pontilhado vazio, o Par Honesto 1 solicita ao Par Malicioso o bloco 1 de uma peça, enviando **Request(*p*,*s*,*l*)**, onde p é a peça, s (*start*) é o byte inicial do bloco e l (*length*) o seu tamanho. Para este exemplo, foi considerado o compartilhamento de uma peça de 64Kbytes, ou seja, 4 blocos;

(6) ao receber a requisição, Par Malicioso envia para Par Honesto 1 uma mensagem contendo a peça corrompida (**Piece(**p,$s1$,l**)**) e logo em seguida **Choke** para que o par não faça mais nenhuma requisição até que seja dessufocado;
(7) a fim de completar o download da peça, o Par Honesto 1 inicia uma série de requisições ao Par Honesto 2, que responde a todas elas com os blocos requisitados. Ao completar a peça, o Par Honesto 1 confere o *hash* da mesma que, devido ao envio do bloco corrompido, irá falhar;
(8) neste momento, termina o tempo do intervalo definido para o ataque e então o Par Malicioso envia a mensagem de **Unchoke** ao Par Honesto 1. Em seguida, este recomeça o download da peça falha pedindo novamente o bloco 1 ao Par Malicioso, reiniciando o ciclo ilustrado pelo retângulo pontilhado maior.

3.3. Desconexão

No ataque anterior, após um **Choke**, o par malicioso aguarda um tempo antes de enviar um **Unchoke** e tentar corromper outra peça (ou a mesma peça, desde que a cópia anterior já tenha sido descartada). Descobriu-se, experimentalmente, que o agente (Azureus, no caso) não pede o restante dos blocos a outros pares imediatamente, mas sim continua pedindo blocos de outras peças a outros pares. Com isso, se o intervalo entre *chokes* é curto, quando o par malicioso retornar, o par honesto solicitará blocos da peça que já foi comprometida.

Visando evitar essa situação e tornar o ataque mais efetivo, nesta variação do ataque de Corrupção de Peças o malicioso fecha a conexão após enviar o bloco corrompido, para logo depois reabri-la. Além disso, ao invés de enviar o **BitField** completo como da primeira vez, o par malicioso configura em 0 o bit relacionado à peça que acabara de enviar, indicando que não possui mais esta peça para compartilhar. Com isso, o honesto solicitará blocos de uma outra peça, que poderá então ser alvo de ataque. Este ataque é efetivo porque, experimentalmente, comprovou-se que o par honesto não contrasta o mapa anterior com o novo mapa recebido; essa situação teria que ser resolvida mantendo-se uma cache de mapas de pares com quem o par local esteve conectado recentemente.

Um fator importante a ser considerado é que nem sempre o par malicioso será o primeiro a estabelecer a conexão com o par remoto. Na verdade, conexões provenientes de outros pares são muito freqüentes, visto que o par malicioso só toma conhecimento de novos pares que ingressaram no enxame após obter uma nova lista de endereços do rastreador (não mais frequente que 2 minutos). Neste caso, quando o par malicioso estabelece uma nova conexão não esperada pelo par honesto, estará se comunicando com este através de uma porta diferente daquela divulgada (pelo par remoto) no rastreador, porque a porta informada ao rastreador é aquela usada para receber pedidos de conexão. Por este motivo, de forma geral, o ataque torna-se ineficiente nos casos em que o par honesto estabelece a primeira conexão, pois é impossível o restabelecimento desta por parte do par malicioso.

Para contornar essa limitação, o par malicioso só aceita conexões com pares novos depois do recebimento do conjunto de endereços do rastreador, conectando-se na porta correta. Nos casos em que o agente modificado recebe uma conexão de outro par, ele irá realizar todos os passos normais do ataque até o momento em que

envia o bloco e fecha a conexão. Possivelmente o par remoto pode querer conectar-se novamente com o par malicioso e então o processo é reiniciado.

Por uma restrição de espaço, não apresentamos o diagrama de tempo ilustrando esse ataque. No entanto, ele é relativamente similar àquele na Figura 1.

Experimentalmente observamos que o ataque pode perder parte de seu desempenho em virtude da "sobreposição" de blocos em peças já corrompidas por outros atacantes, isto é, pelo envio de blocos corrompidos de uma mesma peça por diferentes pares maliciosos devido à ausência de conluio. Conforme já citado, ao escolher uma peça para requisitar de um par, pares dão preferência para continuar o download de peças já iniciadas e que não estão sendo mais obtidas de nenhum par. Como os pares maliciosos mantém seu **BitField** o mais completo possível, com exceção das peças enviadas por ele próprio, o par honesto pode optar por qualquer uma das peças restantes. Deste modo, caso o par honesto já tenha sido atacado por um par malicioso anteriormente, provavelmente terá peças inacabadas que foram corrompidas pelo primeiro par malicioso. Neste caso, quando o segundo par malicioso iniciar seu ataque com o par honesto, receberá requisições de peças *já corrompidas* pelo primeiro par malicioso. Sendo assim, ao enviá-las, poderá não estar causando dano adicional ao par honesto, posto que é parte de uma peça que já contém um ou mais blocos corrompidos e não teve seu *hash* verificado.

Visando superar esta limitação, o atacante pode adotar uma estratégia similar ao *superseeding*. Para evitar que o par remoto possa escolher qual peça deseja receber, o malicioso inclui uma única peça em seu mapa de bits. O malicioso não tem como saber quais peças estão pela metade no par honesto, e portanto sorteia uma peça dentre as que o honesto disse não ter. Desta forma, diminui-se a probabilidade de dois ou mais pares enviarem blocos corrompidos da mesma peça, pois a peça será sorteada e não escolhida pelo honesto. Caso o ataque fosse realizado em conluio, os pares maliciosos poderiam informar uns aos outros quais peças já foram corrompidas em um determinado par e então os outros não escolheriam nenhuma destas peças para divulgar em seus **BitFields**, podendo aumentar ainda mais o impacto deste método de ataque.

4. Implementações do Ataque

Para a implementação dos ataques, modificou-se o agente BitTorrent de código aberto jBitTorrent ([jBittorrent 2007]), parte de uma API em Java criada para auxiliar os estudos sobre o protocolo. A implementação é clara e bem documentada, e o código *multi-threaded* está bem organizado. Cada *thread* é responsável pela comunicação com um único par, em linha com a arquitetura e diagramas de estado que especificamos em [Konrath et al. 2007a].

Foram realizadas modificações neste agente a fim de implementar o ataque de Corrupção de Peças tanto com Sufocamento como com Desconexão. Diversas adaptações foram feitas na implementação, tal como aumentar o número de pares que estão aptos a receberem dados do par local: o par malicioso inicialmente habilita todos os pares a pedirem blocos, enviando um **Unchoke** a cada par honesto conectado.

Para a variação **Sufocamento**, foi implementado o método que controla o envio de **Choke** a pares honestos, cuidando para que o par malicioso envie todos

os blocos de uma mesma peça a um único par. Evita-se assim que este seja o único fornecedor de blocos daquela peça, o que tornaria sua descoberta trivial. Uma atenção especial é necessária para última peça do arquivo, que pode ser composta de apenas um único bloco.

Para a variação **Desconexão**, foi necessária a recriação de diversos métodos e a adaptação do ataque para o sistema de *threads* existente. É por meio delas que a conexão com o par remoto é fechada após o envio de um bloco. Após um período de sincronização (1 segundo), as *threads* responsáveis pela comunicação com o par honesto são novamente instanciadas, restabelecendo a conexão. A única exceção é quando a primeira conexão foi estabelecida a partir do par remoto. Neste caso, o par malicioso é incapaz de reiniciá-la e então aguarda por uma nova conexão ou até que obtenha nova lista de endereços junto ao rastreador.

5. Avaliação Experimental

O diferencial deste trabalho é a condução de um estudo experimental sobre o impacto dos ataques anteriormente identificados na literatura. Esta seção descreve o conjunto de experimentos realizado. Na seqüência, o ambiente é apresentado, seguido dos resultados obtidos com os experimentos envolvendo os ataques de Corrupção com Sufocamento e Corrupção por Desconexão.

5.1. Ambiente de Avaliação

Para avaliar os impactos dos ataques implementados neste trabalho, foi utilizado o TorrentLab[2], um ambiente que permite a avaliação experimental de redes BitTorrent. O TorrentLab utiliza uma infraestrutura distribuída (atualmente, é uma grade OurGrid) para instanciar remotamente os agentes BitTorrent em máquinas de uma rede e posteriormente gerar relatórios gráficos dos experimentos e logs individuais dos pares participantes.

Como agente de usuário, adotou-se o Azureus, versão 3.0.1.6. A escolha por este agente é justificada pela sua popularidade e funcionalidade sofisticada. Uma configuração *default* foi criada e empregada em todos os experimentos, baseada em valores mencionados na literatura ou tipicamente encontrados em configurações reais de agentes como Azureus e μTorrent. O arquivo compartilhado é composto de 60 peças com 61 blocos cada, totalizando 1 MB por peça e 60 MBs o arquivo completo. Os pares honestos foram configurados com razão de contribuição 1, o que significa que cada par contribuirá no mínimo com a mesma quantidade de recursos que obteve da rede. A taxa de download e upload adotadas para os pares honestos foi de 512 Kbps e 128 Kbps, respectivamente, refletindo-se a assimetria tipicamente encontrada em redes com banda larga. Para os pares maliciosos não foi imposto nenhum limite de velocidade, buscando-se analisar os resultados do ataque em sua melhor condição de impacto.

O objetivo dos experimentos é medir a efetividade dos impactos sobre os pares do enxame sem que este falhe globalmente. Os cenários contaram com 50 sugadores e 2 semeadores, todos honestos. O primeiro semeador (semeador inicial)

[2]um artigo descrevendo este ambiente foi submetido ao SBRC2008.

possui taxas de transferência 4 vezes maiores que os sugadores: 2 Mbps de download e 512 Kbps de upload. O objetivo deste semeador é participar apenas do início do enxame, distribuindo rapidamente as primeiras peças do arquivo na rede para então abandoná-lo; para tal, sua razão de contribuição foi configurada com valor 2. Em contraste, o segundo semeador (semeador permanente) foi configurado para que nunca abandonasse o enxame (razão infinita), garantindo que o enxame nunca falhará, pois uma peça jamais tornar-se-á indisponível no enxame. Entretanto, o semeador permanente possui uma taxa de upload bem menor, de 72 Kbps, para permitir que o espalhamento de peças dependa da participação dos pares em geral.

O ingresso dos pares no enxame segue um processo de Poisson, com tempos entre chegadas de pares seguindo uma distribuição exponencial [Eger and Killat 2006, Guo et al. 2005]. Nos experimentos realizados, a grande maioria dos pares chega nos primeiros 3 minutos, porém o último par ingressa no enxame no minuto 26.

Para os pares maliciosos, o tempo entre as requisições junto ao rastreador foram reduzidas para 120 segundos, o mínimo permitido por este. Diminuindo o tempo entre as conexões, crescem as chances do par atacante conectar-se mais rapidamente com pares recém ingressados no enxame, ou seja, de identificar e iniciar seu ataque com estes pares honestos o mais breve possível. Dependendo do tempo entre as conexões junto ao rastreador, a efetividade do ataque pode sofrer variações significativas, tornando-se menos efetivo caso este tempo seja elevado.

Na implementação de um agente, há três conjuntos/listas de pares que são chave. A primeira é **PeerList**, a lista de pares que o rastreador retorna ao ser consultado. O tamanho máximo dessa lista foi configurado para 100. Segundo, o **PeerSet**, o conjunto de pares que o par local conhece e pode usar para abrir novas conexões, e que é "alimentada" por instâncias de **PeerList**. O terceiro é **ActivePeerSet**, que representa o conjunto de pares ativos (conectados), um subconjunto de **PeerSet**. O tamanho máximo do **ActivePeerSet** representa o número máximo de pares com os quais o par pode se conectar. Pares honestos e maliciosos tiveram essa lista limitada em 25 e 100 pares, respectivamente (usualmente o valor é 50, mas alteramos esses valores para mostrar melhor o impacto de situações em que o número de pares fosse maior).

Apesar de conduzido em um ambiente controlado, por se tratar de um trabalho experimental, os resultados obtidos nas execuções podem sofrer interferências de variáveis externas, como problemas na rede ou escalonamento dos processos, ocasionando variações nos resultados. Por este motivo, todos os valores apresentados nesta seção foram obtidos através de repetidas execuções de experimentos, até se obter resultados estatisticamente válidos. O número de repetições necessário oscilou com o tipo de experimento e resultados colhidos.

Na análise dos experimentos a seguir, foram utilizadas duas métricas para medir o grau de impacto dos ataques na rede. A primeira é a taxa efetiva de download obtida, considerando apenas peças baixadas com sucesso, e calculada fazendo-se uma média entre todos os pares em função do tempo que estiveram como sugadores. A segunda é a sobrecarga em termos de peças que precisam ser baixadas repeti-

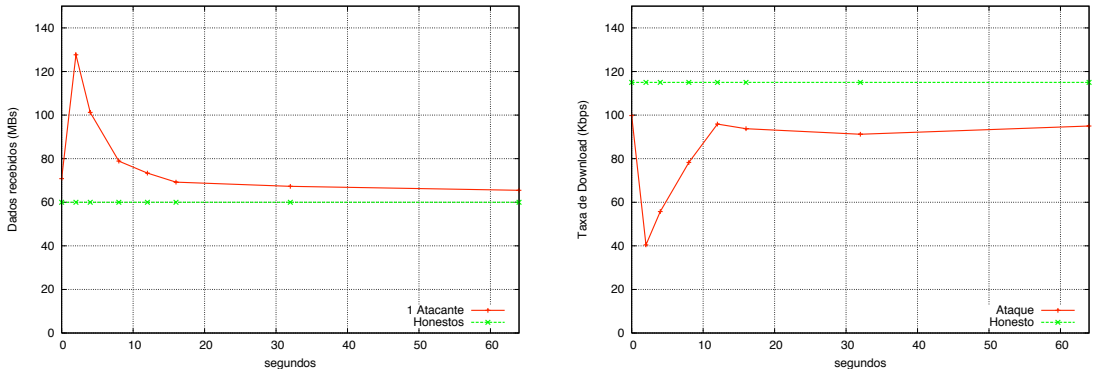
das vezes. Para exemplificar o impacto na dinâmica de um ataque, é mostrada a evolução de um enxame em termos de tipos de pares (sugadores ou semeadores) e downloads completados.

Cada execução individual de um experimento pode consumir várias horas, demandando o uso de um conjunto de máquinas. Para tornar viável a execução dos experimentos dentro de um tempo limitado, optou-se por configurar um tempo máximo para as execuções. Baseado no tempo médio total das execuções em ambientes honestos (menos que 90 minutos), este limite foi configurado em 180 minutos, ou seja, pelo menos o dobro. Sendo assim, ao final do período definido, as execuções são finalizadas e para os pares que não terminaram seus downloads as estatísticas são baseadas no tempo em que estiveram presentes no enxame.

5.2. Experimento com Sufocamento

Como explicado na Subseção 3.2, uma das maneiras de aplicar-se o ataque de Corrupção de Peças é através do envio de mensagens de **Choke** e **Unchoke**. Desse modo, consegue-se fazer com que o par atacante envie um bloco de cada vez, com intervalos de tempo entre eles, a fim de que o par remoto termine a peça falha e volte a pedir novamente blocos ao par local.

O presente experimento tem como objetivo descobrir qual o impacto do ataque, variando-se o tempo de intervalo entre as mensagens de **Choke** e **Unchoke**. Foram avaliados 8 intervalos diferentes: 0, 2, 4, 8, 12, 16, 32 e 64 segundos, onde 0 significa ausência de intervalos entre os envios. Os resultados obtidos foram comparados com as execuções realizadas sem ataque.



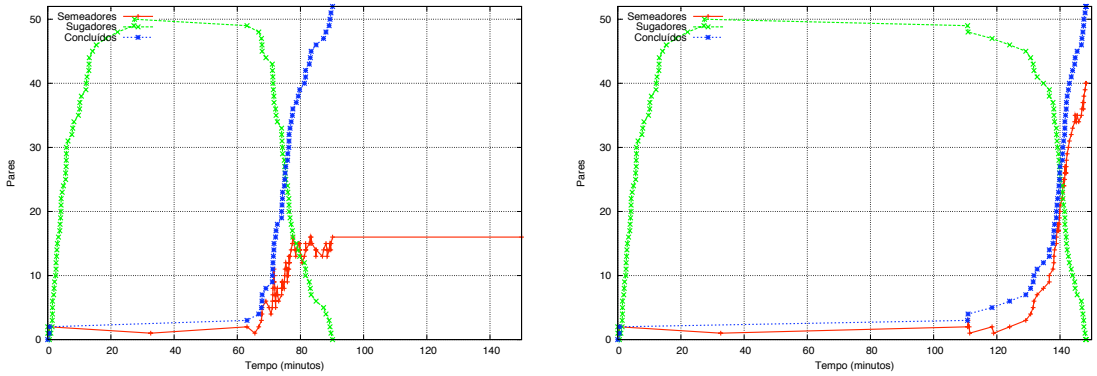
(a) Sobrecarga de Rede (b) Sobrecarga de Tempo ou Taxa

Figura 2. Variação de intervalo usado no Sufocamento.

A Figura 2(a) ilustra a sobrecarga de tráfego de rede (em Mbytes) causada pelo ataque em função do tempo entre mensagens de **Choke** e **Unchoke**, representado no eixo X em segundos. A reta "honestos" se refere ao caso sem ataque, e fica constante nos 60 Mbytes correspondentes ao download do conteúdo (ignora-se no cômputo as mensagens de controle, cujo tamanho é desprezível). Já a curva "1 Atacante" mostra a quantidade de Mbytes que o par precisa fazer download antes que complete o conteúdo. É fácil observar na Figura 2(a) que o melhor intervalo é aquele próximo de 2 segundos; um valor menor que este não é efetivo porque o

malicioso ainda não acabou a peça corrompida na maioria das vezes, enquanto que um valor maior oferece uma chance maior de o par obter peças de pares honestos sem ser importunado.

Já a Figura 2(b) representa a taxa efetiva de download, em Kbps, de peças corretas provenientes dos pares honestos durante os ataques. A reta "honestos" fica constante em torno de 117 Kbps, sendo este o valor esperado, pois o gargalo nesse cenário é a capacidade de upload dos pares, 128 Kbps. Consistentemente com os resultados da Figura 2(a), o ataque com 2 segundos demonstra ser o mais efetivo em termos de taxa. Em ambos os casos, a partir do intervalo 8 segundos ocorre uma variação muito pequena entre os experimentos, causando um baixo impacto.

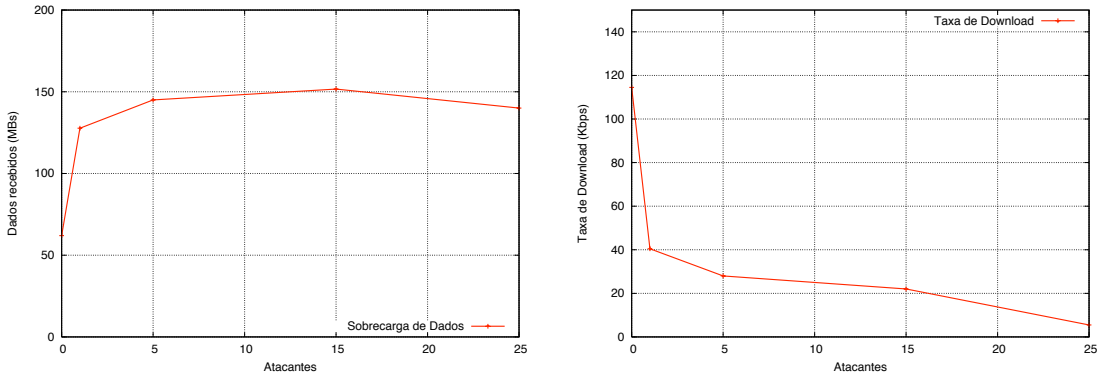


(a) Honestos (b) 8 segundos

Figura 3. Honestos e Sufocamento com 8 segundos de intervalo.

Para melhor compreender a evolução dos pares na rede, a Figura 3(a) apresenta o histórico da população de pares em um enxame sem atacantes, enquanto a Figura 3(b) faz o mesmo para um enxame com um atacante configurado com intervalo de 8 segundos (escolheu-se este valor porque o ataque mais efetivo não seria graficamente ilustrativo, com crescimento, ápice e queda). Em ambos os casos, a população de sugadores cresce bastante no início, demonstrando a chegada de uma *flash crowd* de pares ao enxame, até basicamente 26 minutos, quando todos os pares já entraram no enxame. Na Figura 3(a), os primeiros pares começam a terminar seu download logo depois de 60 min, onde então nota-se uma variação na população de semeadores. A curva referente ao número de downloads concluídos aponta quantos pares já terminaram o donwload até o momento. Observando-a no tempo 0, notamos que inicia com 2, porque os semeadores iniciais são incluídos na contabilização. A quantidade de semeadores cai de 2 para 1 no tempo de 32 min, quando a razão alvo do semeador inicial é alcançada e este abandona o enxame. Nesta comparação, pode-se perceber de forma mais nítida a diferença entre os dois gráficos, no que diz respeito ao tempo de donwload.

Adicionalmente, foram realizados experimentos com este ataque variando-se a quantidade de pares maliciosos no enxame. Para isto, adotou-se o intervalo igual a 2 seg, por ter sido o mais efetivo. O número de pares maliciosos empregados nos experimentos foi: 1, 5, 15, e 25 pares. Todos maliciosos ingressam no tempo 0, junto com o semeador inicial.



(a) Sobrecarga de Rede (b) Sobrecarga de Tempo ou Taxa

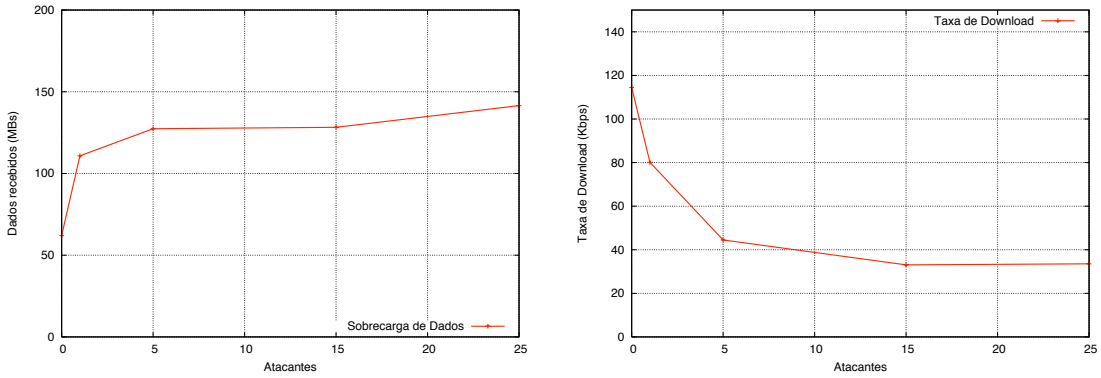
Figura 4. Sufocamento com múltiplos atacantes.

A Figura 4 apresenta gráficos de sobrecarga em função do número de pares maliciosos no enxame, enquanto a métrica é a mesma empregada na Figura 2(a), carga de rede em Mbytes. Devido ao forte impacto obtido com os ataques, em grande parte das execuções nenhum par chegou a terminar seu download, fazendo com que ocorresse uma "estagnação" no crescimento do volume de dados recebidos. Por essa razão, o impacto desses ataques pode ser melhor percebido na Figura 4(b), onde é apresentada a taxa útil (efetiva) de download dos pares. Sem atacantes, a taxa é 117 Kbps na média. Alguns poucos atacantes são suficientes para fazer a taxa despencar, mostrando a efetividade do ataque. Com 25 atacantes, a taxa média ficou próxima de 5 Kbps, uma considerável diferença frente ao caso sem atacantes.

5.3. Experimento com Desconexão

Para o terceiro experimento, avaliou-se o ataque de corrupção de peças com a variação de Desconexão. Como explicado na Subseção 3.3, este ataque pode ser executado de duas formas. A primeira é informando o **BitField** completo ao par remoto, retirando apenas as peças já enviadas a ele. A segunda alternativa é fazer a operação oposta, enviar o **BitField** com apenas uma peça, aquela que quer-se fornecer ao par honesto. De forma experimental, mediu-se o grau de efetividade dos dois métodos. Como resultado, o ataque com **BitField** completo mostrou uma efetividade inferior, embora atingindo níveis de impacto consideráveis. Por questões de espaço, são apresentados apenas experimentos em que o malicioso divulga apenas uma peça a cada conexão.

A Figura 5(a), que ilustra a quantidade de dados que cada par honesto deve carregar antes que complete o conteúdo compartilhado, mostrou-se bastante similar ao ataque de Sufocamento. Ocorre um crescimento rápido da sobrecarga até 5 atacantes, quando estabiliza e converge para próximo de 150 MB. Analisando o gráfico da Figura 5(b), pode-se notar que o impacto deste ataque não é tão efetivo quanto o anterior, mas apresenta uma significativa redução da taxa útil de download dos pares, caindo de 115 para 34 Kbps com 25 atacantes.



(a) Sobrecarga de Rede (b) Sobrecarga de Tempo ou Taxa

Figura 5. Desconexão com múltiplos atacantes.

6. Conclusões e Trabalhos Futuros

Este trabalho trata de ataques que exploram vulnerabilidades no protocolo BitTorrent com o objetivo de prejudicar o desempenho da rede, sem ganho próprio, apresentando duas contribuições principais. Primeiro, permite aumentar o entendimento sobre a implementação de mecanismos de contramedida nos principais agentes. Segundo, o ataque de Corrupção de Peças é explorado em maior nível de detalhe, variantes determinadas, e investigadas sob uma ótica experimental.

Conduzir avaliações experimentais medindo o prejuízo no desempenho de agentes BitTorrent modernos, como Azureus, causados por ataques identificados na literatura representa considerável desafio. Essa é uma das principais razões para que trabalhos anteriores se concentrassem sua análise em simulações. Este artigo representa a primeira contribuição em termos de implementação e experimentação de ataques em um ambiente real.

Os resultados dos experimentos confirmaram que BitTorrent é suscetível a ataques em que pares maliciosos enviam blocos corrompidos. Eles demonstraram que downloads em geral podem ser atrasados em mais de 100% mesmo quando os recursos do ataque são modestos.

Este trabalho pode ser estendido de várias formas. Primeiro e mais importante, gostaríamos de estender a avaliação experimental para cenários de maior escala, em número de pares e em conteúdo: pelo menos 200 pares Azureus compartilhando um conteúdo de maior volume, como 700 MB ou 4.4 GB, valores associados a mídias populares. Entretanto, enfrentamos o desafio de obter para uso exclusivo, por um período considerável, uma infraestrutura computacional em rede que permita realizar tais experimentos. Como alternativa, estamos trabalhando no sentido de alimentar os resultados em uma simulação e extrapolar os resultados para enxames de até 500 pares, o que muitas vezes torna-se inviável no ponto de vista experimental. Segundo, realizar experimentos a fim de avaliar os impactos efetivos da variação de Desconexão, apresentada neste trabalho, utilizando-se o método de envio do **BitField** completo. Por fim, pretendemos propor e avaliar experimentalmente contramedidas, pois além de mostrar a gravidade do problema de forma efetiva, real, o objetivo final de um trabalho como o nosso é tornar o BitTorrent mais seguro e confiável.

Referências

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

Guo, L., Chen, S., Xiao, Z., Tan, E., Ding, X., and Zhang, X. (2005). Measurements, analysis, and modeling of bittorrent-like systems. pages 35–48.

Jain, R. (1991). *The Art of Computer Systems Performance Analysis: techniques for experimental design, measurement, simulation, and modeling*. Wiley.

jBittorrent (2007). Java bittorrent api website. *http://sourceforge.net/projects/bitext/*.

Konrath, M. A., Barcellos, M. P., and Mansilha, R. B. (2007a). 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.

Konrath, M. A., Barcellos, M. P., Silva, J. F., Gaspary, L. P., and Dreher, R. (2007b). Atacando um enxame com um bando de mentirosos: vulnerabilidades em bittorrent. In *XXV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2007)*, volume 2, pages 883–896.

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., Konrath, M. A., and Barcellos, M. P. (2007). Corrupção, mentiras e isolamento: avaliação de impacto de ataques a bittorrent. In *VII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSEG 2007)*.

Muhammad, H. H. and Barcellos, M. P. (2002). Simulating group communication protocols through an object-oriented framework. In *The 35th Annual Simulation Symposium (ANSS 2002)*, pages 418–433.

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.

Pouwelse, J. A., Garbacki, P., Epema, D. H. J., and Sips, H. J. (2005). The bittorrent p2p file-sharing system: Measurements and analysis. In *4th International Workshop on Peer-to-Peer Systems (IPTPS)*.

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.