UNIVERSIDADE DO VALE DO RIO DOS SINOS
CIÊNCIAS EXATAS E TECNOLÓGICAS
PROGRAMA INTERDISCIPLINAR DE PÓS-GRADUAÇÃO EM
COMPUTAÇÃO APLICADA
Daniel R. Bauermann

Experimentação de Contramedidas para Ataques em Arquitetura BitTorrent

São Leopoldo, abril de 2009

UNIVERSIDADE DO VALE DO RIO DOS SINOS
CIÊNCIAS EXATAS E TECNOLÓGICAS
PROGRAMA INTERDISCIPLINAR DE PÓS-GRADUAÇÃO EM
COMPUTAÇÃO APLICADA

Experimentação de Contramedidas para Ataques em Arquitetura BitTorrent

por

DANIEL R. BAUERMANN

Dissertação submetida a avaliação como requisito parcial para a obtenção do grau de Mestre em Computação Aplicada

Orientador: Prof. Dr. João Carlos Gluz

São Leopoldo, abril de 2009

"Saber ouvir, é possuir, além do seu, o cérebro do outro."
"Savoir écouter, c'est posséder, outre le sien, le cerveau des autres."
Leonardo da Vinci

Agradecimentos

Primeiramente preciso agradecer à minha esposa Francis e meu filho Carlos Henrique, dos quais fiquei muito afastado nestes 2 últimos anos. Vocês sempre trouxeram-me e continuam trazendo muitas alegrias. O apoio de vocês é o alicerce que me permite pisar em solo firme. Amo vocês!

Preciso agradecer à minha mãe Lidia e meus irmãos Vitor e Carlos, que iniciaram a construção da base sólida que necessitei para chegar onde estou. Vocês sempre apoiaram-me, encorajam-me e acreditaram em mim. Sou muito grato a vocês!

Toda a família que me apoiou e de quem abdiquei meu tempo para dedicar-me a esta empreitada, meu muito obrigado pela compreensão e apoio. Aos amigos de quem abdiquei o convívio, permanecendo durante muitos finais de semana trabalhando, obrigado!

O que falar de meu orientador Prof. Dr. Marinho Pilla Barcellos? Nossa, exemplo de dedicação, esforço e garra! Apesar de todas adversidades que o destino impôs em teu caminho, seguistes de cabeça erguida e destes a volta por cima. Sem teus conselhos, incentivos, exemplos e perseverança eu não teria chegado ao fim desta caminhada vitorioso. Tenho a convicção que o melhor ensinamento ocorre através de exemplos. Tenhas certeza Marinho que fostes um ótimo exemplo e que jamais esquecerei os valiosos ensinamentos!

Prof. Dr. João Carlos Gluz, obrigado pelo apoio e suporte oferecido, especialmente neste último ano.

Muitas pessoas colaboraram no sucesso deste projeto. Matheus, um muito obrigado pela ajuda e disposição. Ah, sem esquecer as boas piadas! Giovani, pelo eterno bom humor, pelas palavras de motivação e paciência nos diversos ensinamentos. Rodrigo, sua presteza e dedicação. Flávio, obrigadão pelo conhecimento compartilhado. Marlom, o cara que deu o "pontapé" inicial nesta linha e sempre se mostrou prestativo em colaborar.

Por fim, não posso deixar de agradecer ao Gilberto, que deu me deu o empurrão que precisava para entrar "nesta fria". Obrigado pelo apoio , incentivo e trocas de ideias.

Resumo

BitTorrent é, atualmente, umas das mais populares tecnologias para troca de arquivos na Internet. Embora bastante disseminado, estudos anteriores já demonstraram ser possível explorar as vulnerabilidades do protocolo com o objetivo de prejudicar a distribuição de conteúdo. Em um estudo recente foram apresentados os primeiros algoritmos para defesa de ataques em redes BitTorrent. Embora resultados satisfatórios tenham sido encontrados, a avaliação de tais algoritmos foi feita em ambiente simulado, com uma série de premissas simplificatórias. Nesta dissertação são apresentados os primeiros resultados da avaliação dos algoritmos de contramedidas em um ambiente controlado, usando agentes de usuário reais. Além de validar os resultados encontrados através de simulação, demonstrou-se que é possível recuperar-se de ataques através de tais algoritmos.

Palavras-chave: P2P, BitTorrent, Segurança.

TITLE: "EXPERIMENTAL EVALUATION OF COUNTERMEASURES AGAINST ATTACKS IN THE BITTORRENT ARCHIITECTURE"

Abstract

Currently BitTorrent is one of the most popular technologies for file sharing in the Internet. Although its use is quite disseminated, several studies have already showed that it is possible to explore the protocol's vulnerabilities to hamper content sharing. A recent study presented the first defense algorithms against such attacks on BitTorrent networks. Albeit the results where satisfactory, the evaluation of those algorithms was made by simulation, with several simplifying assumptions. This dissertation presents the first results about the evaluation of countermeasure algorithms using real agents in a controlled environment. Besides the validation of simulation results, it was shown that it is possible to recover from attacks using these algorithms.

Keywords: BitTorrent, P2P, Security.

Sumário

Lista de Abreviaturas

DoS	*Denial of Service*
IP	*Internet Protocol*
LRF	*Local Rarest First*
NAT	*Network Address Translation*
P2P	*Peer-to-Peer* ou Par-a-Par
RFC	*Request for Comments*
SHA	*Secure Hash Algorithm*
TCP	*Transmission Control Protocol*
TFT	*Tit-for-Tat*

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A Internet desenvolveu-se baseada no modelo cliente/servidor, no qual servidores centralizados executam tarefas para clientes distribuídos. Ainda hoje, este é o modelo comumente encontrado em aplicações de Internet. A partir do final dos anos 90 a tecnologia *Peer-to-Peer* (P2P) surge para mudar este paradigma. Através desta nova tecnologia, todos os participantes da rede podem atuar tanto como servidores como clientes, não dependendo de uma organização central ou hierárquica [Sadok et al., 2005].

Nesse contexto de aplicações de rede P2P, o compartilhamento de arquivos tem tornado-se uma das aplicações mais significativas. Diferentes tecnologias de compartilhamento de arquivo, tais como Gnutella, KaZaa e BitTorrent têm sido utilizadas. Dentre estas, o BitTorrent é uma das mais populares, sendo já responsável por uma considerável fração do tráfego de Internet [Barbera et al., 2005]. O BitTorrent faz uso dos recursos dos pares na rede para distribuir de forma eficiente grandes volumes de dados. Em virtude de suas qualidades, aplicações baseadas em BitTorrent estão sendo desenvolvidas para distribuição de conteúdo digital gerado por companhias de mídia, como BBC e Twentieth Century Fox.

Assim como a popularidade do BitTorrent, têm crescido os riscos e os impactos de um ataque que explore as suas vulnerabilidades. Dois tipos diferentes de ataques foram analisados em [Konrath et al., 2007], demonstrando a possibilidade de exploração das vulnerabilidades existentes no BitTorrent. Tais estudos foram executados através de simulações, utilizando como arcabouço de simulação o Simmcast [Barcellos et al., 2006].

Já é possível encontrar na literatura artigos que demonstraram a possibilidade real de ataques em redes BitTorrent na Internet. Em [Schmitt et al., 2008] são apresentados os resultados de experimentos conduzidos em implementações do protocolo que exploram ataques de negação de serviço em BitTorrent, especificamente ataques de Corrupção, implementados e avaliados em um ambiente controlado. De forma semelhante ao estudo apresentado em [Konrath et al., 2007],

mas em redes controladas de longo alcance (usando PlanetLab), os autores em [Dhungel et al., 2008] investigam ataques denominados "bloco falso" e "par não cooperativo". Os resultados indicam que os ataques de fato ocorrem, mas que seu impacto não chega a dobrar o tempo de carga médio. Já em [Dhungel et al., 2009], os mesmos autores identificam um novo ataque, denominado "ataque ao semeador", onde o atacante, agindo no início do enxame contra o semeador inicial, obtem grande sucesso em impedir a distribuição do conteúdo.

O ataque de Eclipse no BitTorrent foi apresentado pela primeira vez em [Konrath et al., 2007]; em [Zip, 2009], descreve-se um ataque com características semelhantes. No entanto, este último ocorre em uma rede pública, sendo inclusive apresentado um agente de usuário, denominado ZipTorrent, cujo objetivo é atrasar a distribuição de conteúdos populares. Fazendo uso de centenas de agentes que não contribuem no enxame, os atacantes geram impacto significativo na distribuição do conteúdo.

Esta dissertação está inserida em um contexto maior de pesquisa sobre segurança em P2P. Com o objetivo de criar mecanismos de defesas para os ataques em rede BitTorrent, o autor desta dissertação participou efetivamente da elaboração de algoritmos de contramedida, especificamente para os ataques de Mentira em Massa e Corrupção de Peças. Como fruto deste trabalho, em [Barcellos et al., 2008a, Bauermann et al., 2008] foram apresentados os algoritmos de defesa para os ataques citados. Através de um modelo criado em simulação, foram comprovadas a eficácia e a eficiência dos algoritmos criados. Os resultados reportados naqueles artigos serviram de base para comparação com aqueles encontrados no presente trabalho.

Embora resultados satisfatórios já tenham sido obtidos no ambiente simulado, é fundamental a avaliação dos algoritmos de defesa já propostos em agentes de usuário reais. Conforme [Dhungel et al., 2009], muitos agentes, em sua implementação, desviam significativamente do protocolo original do BitTorrent.

A principal contribuição desta monografia é a avaliação experimental, em ambiente controlado, das contramedidas para ataques na arquitetura BitTorrent. São resultados deste trabalho, ainda:

- estudar o comportamento de agentes de usuário BitTorrent em ambiente controlado;
- implementar em um agente real os principais ataques e avaliar parâmetros e condições ambientais que afetam sua efetividade;
- implementar em um agente real os algoritmos contramedidas apresentados em [Bauermann et al., 2008];
- avaliar a eficácia desses algoritmos em ambiente controlado.

O restante deste documento está organizado em sete capítulos. O Capítulo 2 revisa brevemente os conceitos de BitTorrent sob o ponto de vista de arquitetura e protocolo. Também descreve sucintamente a diversidade de agentes de usuário. No Capítulo 3 são descritos de forma geral os ataques alvos das contramedidas avaliadas neste trabalho. O Capítulo 4 descreve os algoritmos de contramedidas propostos no trabalho prévio a essa dissertação. A escolha dos agentes e o detalhamento das implementações podem ser encontrados no Capítulo 5. Enquanto o Capítulos 6 descreve os resultados obtidos através de simulação, o Capítulo 7 descreve o ambiente, o modelo de avaliação e os resultados encontrados através de experimentação com agentes reais. Por fim, o Capítulo 8 encerra com as conclusões e trabalhos futuros.

Capítulo 2

BitTorrent

Existem diversos trabalhos na literatura [Cohen, 2003, Qiu and Srikant, 2004, Izal et al., 2004, Konrath, 2007] (e fontes menos confiáveis[1]) sobre o funcionamento do protocolo BitTorrent. Pode-se assumir seguramente que trata-se de um protocolo razoavelmente conhecido. Entretanto, não existe uma definição padronizada, ou aceita de forma unânime, do protocolo, levando a interpretações distintas sobre seu funcionamento. Na Seção 2.1 são apresentados os principais componentes de um sistema BitTorrent. Detalhes do protocolo são explicados na Seção 2.2. Por fim, na Seção 2.3 são definidos o papel de agentes de usuário do trabalho.

2.1 Arquitetura

BitTorrent é uma aplicação P2P cujo objetivo é distribuir de forma eficiente arquivos entre diversos pares de uma rede. Um único arquivo compartilhado pode estar associado a centenas de pares em processo de *download*, chegando à casa de várias dezenas ou até mesmo centenas de milhares de pares durante seu ciclo de vida [Qiu and Srikant, 2004]. O conjunto formado pelos pares trocando partes de um mesmo arquivo é denominado "enxame".

A ideia básica do BitTorrent é dividir em "peças" o arquivo a ser compartilhado, sendo cada peça de tamanho fixo (exceto a peça final). Peças são subdivididas em blocos de 16KB. O número de blocos por peça é configurado tipicamente de acordo com o conteúdo sendo compartilhado, mas constante para todas as peças)exceto a última).

[1] Embora apresente informações relevantes sobre o protocolo, como o próprio título indica, o conteúdo encontrado em [Cohen, 2009b] não é fruto de um processo de padronização tal como uma *Request for Comments* (RFC), podendo induzir a erros na implementação.

Um arquivo com extensão ".torrent" possui informações sobre o arquivo[2] compartilhado e sobre o conjunto de peças que o compõem, incluindo o *hash* SHA1 de cada peça. Esse *hash* é validado ao término do *download* da peça. Para obter um arquivo disponibilizado através do .torrent, um par precisa se conectar com o "rastreador" (*tracker*), que é um elemento centralizador do BitTorrent, responsável pelo conhecimento do "enxame" (*swarm*). O enxame, por sua vez, é apenas uma visão da composição atual do grupo de pares, pois ela é mantida de maneira bastante "frouxa": pares podem sair silenciosamente do enxame, o que será mais cedo ou mais tarde detectado pela falta de contato do par com o rastreador. Se um par deixa de conectar o rastreador no período esperado consecutivas vezes, o rastreador assume que ele saiu do enxame.

Ao se conectar com o rastreador o par informa seu endereço *Internet Protocol* (IP) e a porta *Transmission Control Protocol* (TCP) em que estará esperando por conexões, que são registrados pelo rastreador (e posteriormente informado aos pares do enxame que necessitem conexões com novos pares). Após a conexão com o rastreador, um par tipicamente solicita uma lista de outros pares participantes do enxame. Os pares dessa lista são escolhidos pelo rastreador aleatoriamente com base na visão atual do enxame.

Um par que possui todas as peças de um arquivo e está contribuindo com os outros pares é dito "semeador" (*seeder*). Já o par que ainda não possui uma cópia completa do arquivo e continua fazendo *download*, ao mesmo tempo que distribui as peças que já possui, é chamado "sugador" (*leecher*). A lista de pares distribuída pelo rastreador é composta tanto de semeadores como de sugadores.

2.2 Protocolo

Na escolha da peça que um par deve solicitar, o BitTorrent emprega uma política chamada "Peça Local Mais Rara Primeiro" (*Local Rarest First* (LRF)). Para cada peça anunciada, o par verifica qual a peça menos replicada conforme o anúncio de peças dos pares com os quais o par local está conectado. Essa política é usada para tentar garantir uma maior homogeneidade na distribuição de peças na rede, diminuindo os riscos de faltar a peça caso o(s) par(es) quem contem(êm) aquela peça deixe(m) o enxame. Além disso, o par local aumenta a chance de outros pares trocarem dados com ele, pois possuirá uma cópia de uma peça menos replicada.

Uma exceção para essa política é quando o *download* é iniciado. Nesde ponto, o par local ainda não possui nenhuma peça para compartilhar e o mais importante

[2]BitTorrent suporta o compartilhamento de mais de um arquivo simultaneamente, mas para facilitar a compreensão, será descrito o caso com apenas um arquivo.

é completar sua primeira peça o mais rápido possível. Assim, a seleção das peças é feita de forma aleatória até que o par complete sua primeira peça. Depois disso, a política utilizada será a LRF [Cohen, 2003].

Para estimular a cooperação dos pares no enxame, o protocolo BitTorrent emprega uma política denominada *Tit-for-Tat* (TFT): o par local colabora preferencialmente com os pares que estão lhe proporcionando as melhores taxas de *download*. O objetivo do uso desta política é eliminar os pares-carona[3] (*free-riders*) do enxame.

Novamente existem exceções para o uso dessa política: (a) quando o par está iniciando seu *download* e portanto não possui nenhuma peça para compartilhar; (b) no caso dos semeadores, que não realizam *download* dos pares, apenas contribuem para o enxame. Neste último caso, são avaliadas as taxas de *upload* dos pares a fim de distribuir mais rapidamente as peças, aumentando, assim, o número de semeadores.

A comunicação dos pares no BitTorrent acontece através da troca de mensagens. Para melhor compreensão desse mecanismo, as mensagens serão tratadas conforme sua função, sem detalhar parâmetros adicionais. Detalhes do protocolo e das mensagens podem ser encontrados em [Cohen, 2009b, Bit, 2009c] e também no trabalho [Konrath, 2007].

Ao obter um arquivo .torrent, o par encontra o endereço do rastreador, dentre outras informações. Após estabelecer uma conexão com o rastreador, o par envia um mensagem GET com informações que identificam de forma única o arquivo que deseja receber. Como resposta, o rastreador envia uma mensagem contendo uma lista, PEERLIST, formada por pares do enxame sorteados aleatoriamente. O par fecha a conexão e novas comunicações com o rastreador só serão feitas periodicamente para avisar que o par continua no enxame ou quando o par necessitar uma nova PEERLIST (por questões de segurança e desempenho o rastreador só responde a novas requisições do par após um intervalo de tempo definido pelo próprio rastreador).

Após obter uma lista de pares com os quais pode trocar dados, o par local passa a se comunicar diretamente com estes pares (sem necessitar da interferência do rastreador). Para iniciar a comunicação com um par remoto, o par local deverá estabelecer uma conexão com ele e então enviar uma mensagem de HANDSHAKE. Ao receber uma mensagem desse tipo, o par verifica se deseja efetivar a conexão com o novo par solicitante, pois há no agente um limite máximo de conexões que podem ser mantidas (usualmente configurável). Podendo efetivar esta nova conexão, uma mensagem de HANDSHAKE é enviada como resposta pelo par remoto. Caso contrário, uma mensagem de REFUSE é enviada como retorno e a conexão é fechada.

Após efetivada a conexão entre dois pares, são trocadas mensagens entre ambos com o mapa de peças de cada par. Denominado BITFIELD, consiste em um

[3]Pares que consomem recursos de rede mas não contribuem para a rede.

mapa de bits representando todas as peças do arquivo onde cada bit é marcado caso o par local possua aquela peça, ou não marcado caso ainda não a tenha obtido.

Ao analisar o BitField de um par remoto, o par local procura por peças que ainda não possua e, portanto, que sejam de seu interesse. No caso de encontrar peças nestas condições, o par local envia uma mensagem de Interested para o par remoto a fim de avisá-lo sob seu interesse em determinada peça. Sempre que não houver mais interesse em alguma peça (caso em que o par local tenha recebido a peça de um outro par remoto, por exemplo) que anteriormente estava marcada como "interessante", o par local deve avisar os pares que não está mais interessado na peça, através do envio de uma mensagem Not_Interested.

Além do controle de interesse por peça que cada par possui, ele também controla quais pares estão "sufocados" (*choked*) ou "não sufocados" (*unchoked*). Inicialmente todos os pares estão no estado sufocado. Conforme descrito na seção anterior, o uso da política TFT define quais pares não serão mais sufocados. Assim, ao ser eleito como par não sufocado e tendo o par remoto interesse em alguma das peças do par local, este envia uma mensagem de Unchoked para o par remoto.

Um par local mantém a troca de dados com até no máximo um certo número de pares remotos (4 é um padrão *de facto*). A cada ciclo de 30 segundos ele aplica um algoritmo chamado "dessufocamento otimista" (*optimistic unchoking*). Seu objetivo é tentar descobrir um novo par que possa ter melhores taxas de *download* em relação aos pares atuais. Caso exista um par para preencher esta vaga, o par com a pior taxa de *download* recebe uma mensagem de Choked e é substituído pelo novo par eleito.

Ao receber um mensagem de Unchoked, o par local escolhe uma peça de seu interesse (respeitando a política da LRF) e envia uma mensagem de Request para solicitar os blocos da peça eleita. O par remoto, ao receber esta mensagem, responde com uma mensagem de Piece enviando os blocos solicitados. Assim os pares permanecem trocando solicitações e respostas até que a peça se complete. Ao receber uma peça por completo, o par verifica se a peça está íntegra (através da informação de seu *hash*) e anuncia para os demais pares com os quais está conectado que possui a peça. O anúncio é feito através da mensagem Have. Essa mensagem é importante para os demais pares, pois influencia a seleção de peças feita através da política LRF dos pares remotos.

A Figura 2.1 sintetiza a descrição acima no caso ideal de troca de dados entre dois pares, desconsiderando eventos de falha (*hash* inválido ou uma falha de comunicação, por exemplo) ou negação (recebimento de sufocamento ou negação de conexão, por exemplo). Os retângulos menores em cinza representam intervalos de tempos variados. O diagrama representa a troca de dados apenas entre o Par 1 e Par 2, considerando que as peças que cada um possui são complementares às peças que o outro par necessita. Além disso, a mensagem de resposta do Par 2 (Handshake

e BitField) foram unificadas em uma mensagem única por questões de espaço.

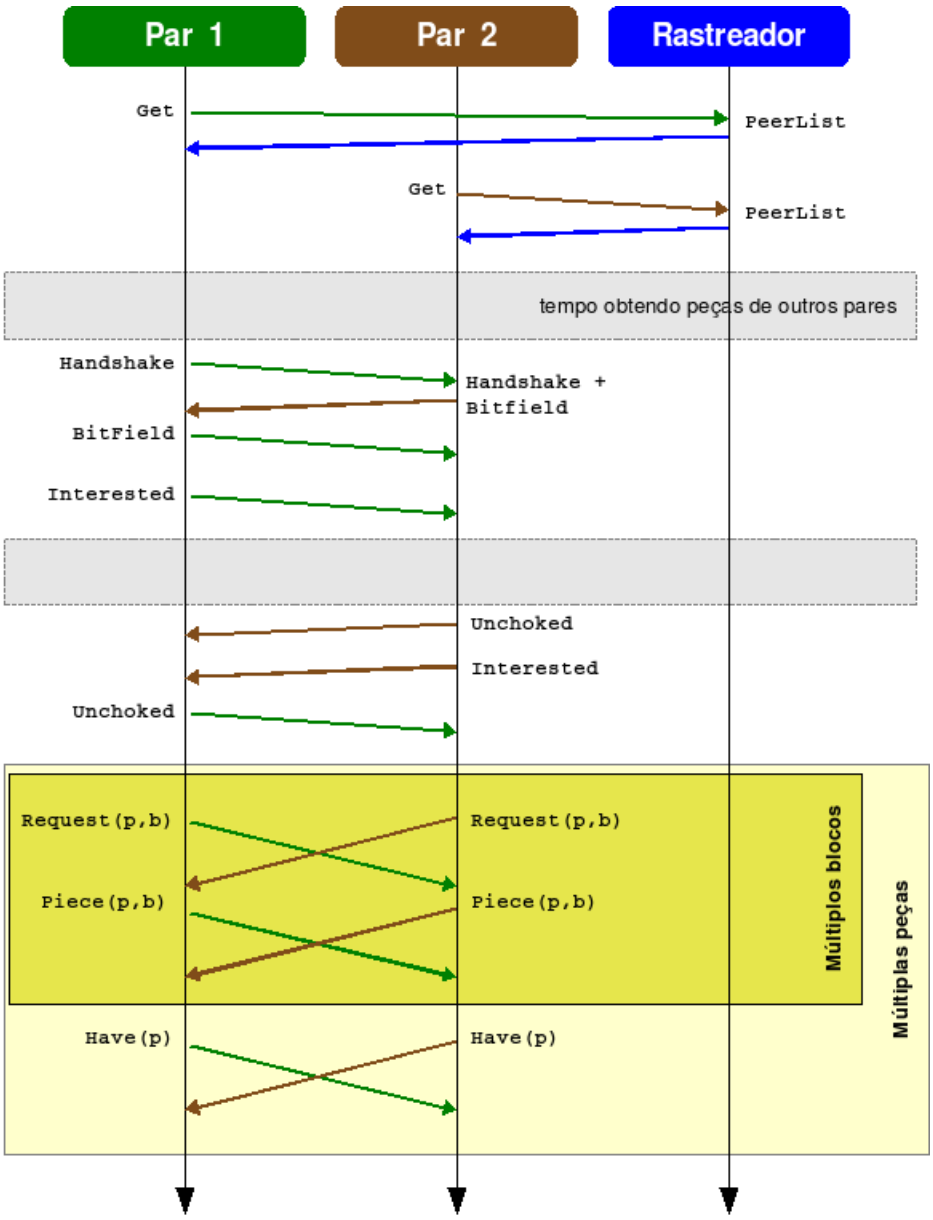


Figura 2.1 – Diagrama de tempo ilustrando troca de dados entre pares no BitTorrent.

2.3 Agentes de usuário

No contexto deste trabalho, agentes de usuário são programas que permitem o compartilhamento de conteúdo usando o protocolo BitTorrent. Existem agentes escritos em diversas linguagens de programação e que executam em uma variedade de plataformas. Em [Bit, 2009b] há uma lista com mais de 50 agentes diferentes, demonstrando a diversidade de opções disponíveis.

O primeiro agente de usuário foi desenvolvido por Bram Cohen, criador do protocolo BitTorrent [Cohen, 2009a]. Esse agente também é conhecido como Mainline em referência à sua origem principal e precursora. A partir da versão 6.0 o código do BitTorrent passou a ser baseado no μTorrent [UTo, 2009] (escrito em C++) e deixou de ser código aberto, sendo seu executável disponibilizado apenas para Windows. Antes dessa versão, o BitTorrent era desenvolvido em Python e disponibilizado de forma aberta para diferentes plataformas.

Ao criar um agente de usuário, não é necessário implementar o protocolo de forma completa. Há diversos exemplos de agentes que estão baseados somente em uma parte do BitTorrent. No contexto do presente trabalho, foram considerados apenas agentes que implementam o protocolo por completo.

Capítulo 3

Segurança em BitTorrent

Como a arquitetura do BitTorrent não oferece um sistema de controle de identidade de usuários, é possível a um usuário instanciar vários agentes na mesma máquina. Além disso, é possível a um agente gerar um conjunto de identidades falsas na rede, passando-se por diferentes nós. Esse ataque é conhecido na literatura pelo nome de *Sybil* e foi introduzido mais detalhadamente em [Douceur, 2002]. Apesar de ser tratado como um ataque, *Sybil* por si só não causa grande impacto à rede, mas pode aumentar significativamente os danos de outros ataques caso utilizado em conjunto. Dos ataques descritos nesta seção, apenas o último deles não faz uso de *Sybils* para alavancar sua eficácia de ataque.

A seguir são descritos os casos de ataque à arquitetura BitTorrent. Os ataques de Eclipse, Mentira e Corrupção de Peças são apresentados nas Seções 3.1, 3.2, 3.3 e 3.4, respectivamente. Já a Seção 3.5 descreve as principais contramedidas nativas encontradas nos principais agentes de usuário.

3.1 Ataque de Eclipse

No ataque Eclipse [Singh et al., 2006] o objetivo é cercar um par honesto com um conjunto suficientemente grande de pares maliciosos, de forma a isolá-lo da rede sem que o mesmo perceba. Se o atacante tiver recursos suficientes, pode alvejar o enxame como um todo, potencialmente levando-o ao colapso.

A Figura 3.1 ilustra um exemplo dessa estratégia de ataque. Pode-se observar um atacante controlando 4 identidades *Sybil*. Tanto o Honesto 1 como o Honesto 2 estão conectados com os *Sybils*, mas supõem estar conectados com outros pares honestos. A título de simplificação, o exemplo assume que um par honesto mantém apenas 4 conexões. Neste caso, os pares honestos não conectam-se entre si e não efetivam troca de dados, permanecendo "eclipsados" um em relação ao outro pelos pares controlados pelo atacante.

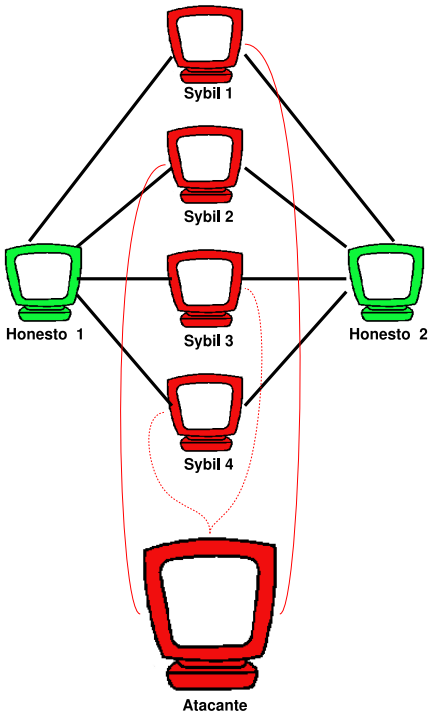


Figura 3.1 – Exemplo de ataque de Eclipse.

3.2 Ataque de Mentira de Peças

Neste tipo de ataque o objetivo é 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 suas identidades, a posse de uma peça que não possui. Assim, o atacante está artificialmente aumentando o grau de replicação da mesma. Com isso, consegue interferir na política de escolha das peças por pares honestos (os que são seus vizinhos), induzindo-os a selecionarem primeiro outras peças menos replicadas. Portanto, a peça sobre a qual o atacante mente terá sua disponibilidade real reduzida gradativamente e poderá mesmo desaparecer do enxame, com a saída dos nós que realmente a possuem.

Como os pares maliciosos não desejam entregar uma peça sobre a qual estão mentindo, duas estratégias são possíveis: (a) manter sempre os outros pares como sufocados; (b) deixar de sufocar um par mas não responder caso o mesmo solicite a peça (ou conjunto de peças) sendo anunciada falsamente. No primeiro caso, o par malicioso nunca realizará *upload* para os pares conectados, mas, assim como qualquer par correto, ele poderá ser selecionado para receber conteúdo

via dessufocamento otimista. No caso de não responder a requisições, o dano aumenta, porque pares maliciosos terão informado pares remotos que estes podem solicitar *download*, mas o par malicioso jamais responderá às solicitações, levando a *timeouts* no par correto [Konrath, 2007].

3.3 Ataque de Mentira em Massa

Para que o ataque de Mentira de Peças seja efetivo, é necessário que cada par honesto receba de um grande número de anúncios falsos sobre uma peça. Ao utilizar um número elevado de pares mentirosos, o ataque também atua como Eclipse. Por essa razão, pode ser visto como um Eclipse avançado. Batizado de "mentira em massa", o ataque é uma combinação de Eclipse com Mentira de Peças [Barcellos et al., 2008a, Bauermann et al., 2008]. Por sua efetividade, o mesmo é adotado como estratégia de ataque no restante do trabalho.

3.4 Ataque de Corrupção de Peças

Neste tipo de ataque o par malicioso pode optar por enviar um ou mais blocos corrompidos de uma peça ao par honesto. Conforme o protocolo BitTorrent (discutido no Capítulo 2), não é possível ao par local descobrir o conjunto de blocos incorretos de uma peça não íntegra. Assim, recebendo apenas um bloco corrompido o par honesto necessita descartar a peça inteira e fazer o *download* de todos os seus blocos novamente. No pior caso, o tamanho da peça pode chegar a 4MB e o envio de um único bloco errado (16KB) provocará o desperdício dos outros 255 blocos (4080KB) recebidos de forma correta, pois não há como o par saber qual dos 256 blocos está inconsistente com o conteúdo da peça [Mansilha et al., 2007].

Ao contrário dos ataques descritos anteriormente, no caso da corrupção de peças é necessário ao atacante utilizar mais de seus recursos computacionais (principalmente largura de banda) para efetivar o ataque. Sendo necessário apenas um bloco para corromper uma peça integralmente, o desafio do atacante é encontrar o intervalo entre ataques a fim de maximizar a eficácia do ataque e melhor aproveitar seus recursos computacionais.

A Figura 3.2, bem como os passos descritivos da mesma, foram apresentados originalmente em [Schmitt et al., 2008] e auxiliam a compreensão da sistemática do ataque, sendo reproduzidos a seguir.

(1,2,3) os nós Honesto 2, Malicioso e Honesto 1 iniciam conexões com o Rastreador e obtém uma lista de endereços de pares que estão no enxame, a PeerList;

(4) ao receber a PeerList, o Honesto 1 envia uma mensagem de Handshake para o

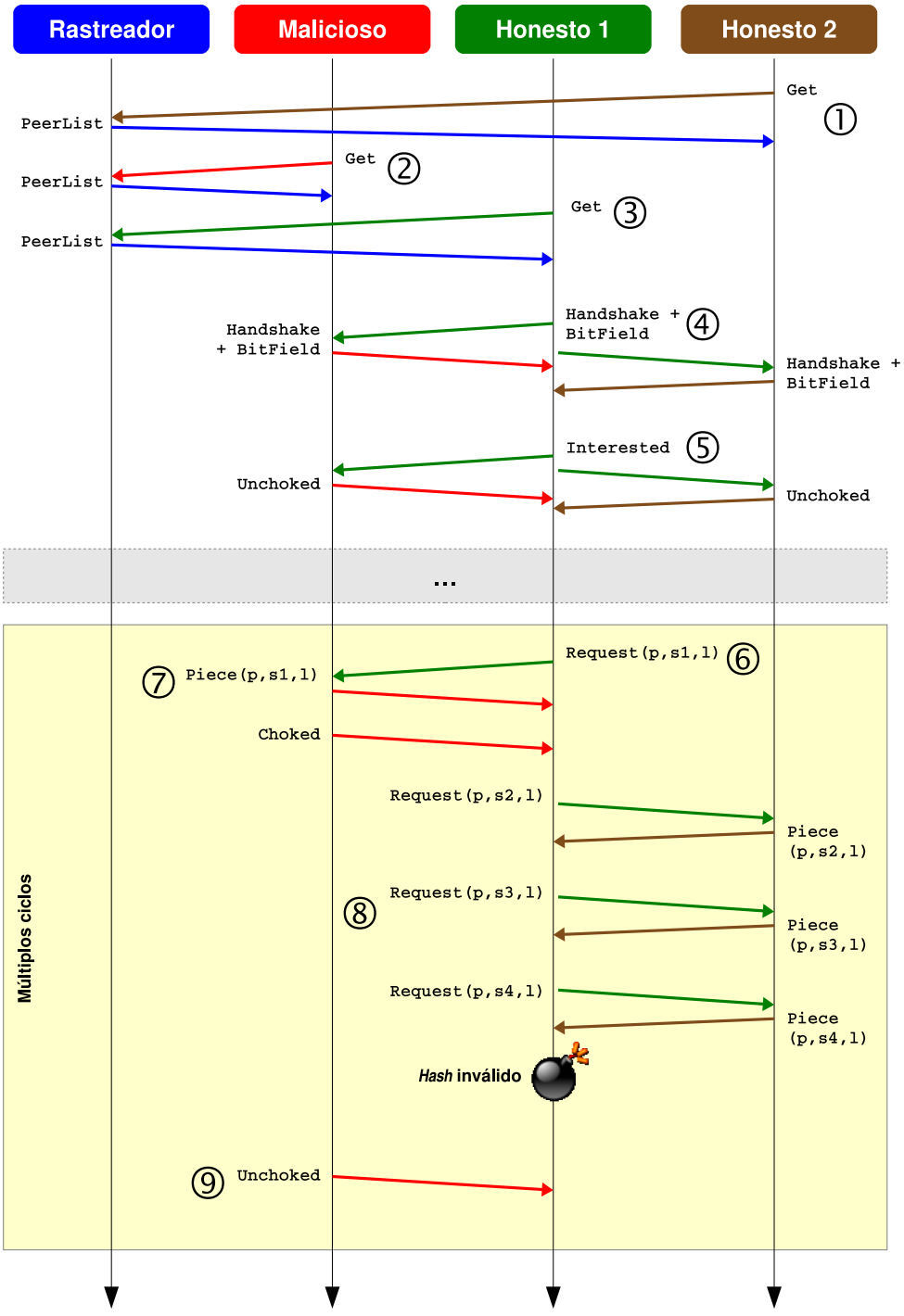


Figura 3.2 – Diagrama de mensagens do ataque de Corrupção de Peças.

Malicioso e para o Honesto 2 (conjunto de pares conhecidos neste exemplo). Ao receber o HANDSHAKE, ambos os pares aceitam efetivar a conexão, respondendo com uma mensagem de HANDSHAKE + BITFIELD. Ao informar seu BITFIELD, o Honesto 2 mostrou ser um sugador, enquanto o Malicioso mentiu sobre a posse de todas as peças e anuncia-se com um semeador;

(5) o Honesto 1 envia mensagens INTERESTED para ambos os pares, mostrando seu interesse nas peças disponibilizadas por estes. A mensagem é respondida pelos pares com um UNCHOKED;

(6) após algum tempo variável (simbolizado pelo retângulo menor em cinza pontilhado), o Honesto 1 solicita ao Malicioso o bloco 1[1] de uma peça, enviando uma mensagem REQUEST(P, S, L), onde p é a peça, s (*start*) é o byte inicial do bloco e l (*length*) o seu tamanho. Neste exemplo, foi considerado o compartilhamento de uma peça composta de 4 blocos;

(7) ao receber a requisição, o Malicioso responde para o Honesto 1 com uma peça corrompida através da mensagem PIECE(P, S1, L) e logo em seguida uma mensagem CHOKED para que o par não faça mais nenhuma requisição até que receba uma mensagem de UNCHOKED;

(8) para completar o *download* da peça, o Honesto 1 inicia um série de requisições ao Honesto 2. Este responde a todas as requisições com os blocos solicitados íntegros. Após completar a peça, o Honesto 1 verifica o *hash* da mesma que irá falhar devido ao bloco corrompido enviado inicialmente pelo atacante;

(9) após um intervalo de tempo definido na lógica do ataque, o Malicioso envia uma mensagem UNCHOKED para o Honesto 1. Havendo disponibilidade e interesse, o Honesto 1 começa o *download* de alguma peça com o Malicioso, iniciando o ciclo ilustrado no retângulo maior.

Ao serem forçados a realizar o *download* de peças consecutivas vezes, os pares honestos geram um grande desperdício de sua banda.

3.5 Contramedidas existentes

Implementações de agentes BitTorrent incluem algumas contramedidas básicas. A definição original do protocolo BitTorrent já prevê um mecanismo denominado "anti-snubbing", que tem o objetivo de lidar com pares cuja comunicação se

[1] O bloco 1 é usado a título de ilustração, podendo os pares honestos solicitarem qualquer bloco.

encontra estagnada. Não existe, no entanto, consenso quanto à sua forma de funcionamento. Segundo [Cohen, 2003], o *anti-snubbing* aumenta o número de pares que são concorrentemente beneficiados com *optimistic unchoking*. Em princípio, o esquema de *optimistic unchoking* seleciona um (único) par de forma aleatória, para o qual enviará blocos quando solicitado. Em contraste, com *anti-snubbing* o par local monitora a quantidade de dados contribuída pelos pares beneficiados por *upload* e, caso não seja recebido nenhum bloco completo do par remoto em 1 minuto, então esse é sufocado e utiliza-se a vaga dele para um *optimistic unchoking* adicional. Portanto, é possível que em um dado momento, múltiplos *optimistic unchoking* sejam acionados porque os pares dos quais se pode esperar contribuição ficaram inativos.

Em relação a peças corrompidas, os agentes BitTorrent mais populares possuem "IP filters", mecanismos de contramedida que punem pares que enviam peças corrompidas. A alternativa mais agressiva é banir todos os pares que contribuem para uma peça. Uma limitação desse esquema é que um par não malicioso que contribui para o par local com grande quantidade de dados tem maior chance de ser punido. Isso prejudicaria o próprio par local. Um mecanismo mais sofisticado, adotado no Azureus/Vuze [Azu, 2009], consiste em monitorar quantas vezes cada par contribui para peças corrompidas e comparar com a quantidade de peças corretas que este par tem enviado.

Uma outra possibilidade é permitir ao usuário final que faça esse controle manualmente, adicionando e obtendo pares de uma lista global de pares banidos. Tal lista pode, em tese, ser alimentada a partir de informações registradas em um servidor público que concentra informações de pares maliciosos. No entanto, isso traz a tona outras questões, como por exemplo a dinamicidade de IPs, pares atrás de *Network Address Translation*s (NATs), o problema de escalabilidade e a garantia de veracidade das informações lá mantidas (ataques de falsa denúncia podem ser facilmente executados). Conforme demonstrado em medições realizadas por [Dhungel et al., 2008], essa estratégia de carregar IPs que foram "negativados" não é suficiente.

Capítulo 4

Algoritmos de contramedidas propostas

As contramedidas descritas neste Capítulo foram primeiramente apresentadas em [Barcellos et al., 2008a] e [Bauermann et al., 2008]. Embora esses trabalhos tenham sido anteriores a esta dissertação eles foram a base inicial do presente trabalho, servindo tanto de base de implementação como de modelo de avaliação comparativa. Assim, a descrição dos algoritmos de contramedidas é reproduzido aqui para clareza da presente dissertação.

Com o objetivo de auxiliar o entendimento da lógica dos algoritmos de contramedidas, a arquitetura e protocolo BitTorrent descritos no Capítulo 2 é revisada na Seção 4.1 em uma visão orientada a conjuntos. A seguir, as Seções 4.2 e 4.3 apresentam os algoritmos propostos, Rotação de Pares e Anti-corrupção, respectivamente, e discutem seu funcionamento.

4.1 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.

4.2 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 p_i: só vale a pena rotacionar pares inativos se 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, denotado 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** $p_j \in Q$ **do**
2: $q_j \leftarrow q_j - 1$
3: **if** $q_j = 0$ **then**
4: $Q \leftarrow Q \setminus \{p_j\}$
5: **end if**
6: **end for**
7: $a \leftarrow |P \setminus (A \cup Q)|$
8: **for all** $p_j \in A$, ordenado por r_j **do**
9: **if** $(t_j \geq t_{min} \wedge \frac{d_j}{t_j} < r_{min}) \wedge (|A| > A_{min} \vee (|A| \geq \lfloor \frac{3}{4} A_{min} \rfloor \wedge a > 0))$ **then**
10: $A \leftarrow A \setminus \{p_j\}$
11: $Q \leftarrow Q \cup \{p_j\}$
12: $q_j \leftarrow \lfloor cq_j \rfloor$
13: $cq_j \leftarrow cq_j \times f$
14: **if** $|A| < A_{min}$ **then**
15: $a \leftarrow a - 1$
16: **end if**
17: **end if**
18: **end for**
19: **while** $|A| < A_{min} \wedge P \setminus (A \cup Q) \neq \emptyset$ **do**
20: $p_k \leftarrow \forall p_j \in P \setminus (A \cup Q)$
21: $A \leftarrow A \cup \{p_k\}$
22: $t_k \leftarrow 0$
23: $d_k \leftarrow 0$
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 dar oportunidade a 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 no Capitulo 6.

4.3 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, foi(ram) o(s) que enviou(aram) 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 $o \in [0:1]$. 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

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
```

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 alocação de um par em 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.

Capítulo 5

Implementação de ataques e contramedidas em agentes de usuário

O foco deste trabalho é a implementação e avaliação experimental dos algoritmos de contramedida apresentados nos artigos [Barcellos et al., 2008a] e [Bauermann et al., 2008]. Assim, nas seções a seguir, serão descritos aspectos da implementação desses algoritmos em um dos agentes de usuário BitTorrent existentes. Na Seção 5.1, são apresentados os critérios para escolha e características do agente que foi selecionado. O modelo original dos agentes escolhidos é brevemente tratado na Seção 5.2. Nas Seções 5.3 e 5.4 são apresentados detalhes dos ataques implementados. Encerrando, as Seções 5.5 e 5.6 detalham a implementação das contramedidas.

5.1 Escolha dos agentes de usuário

Conforme descrito na Seção 2.3, a diversidade de agentes disponíveis é grande. Por isso, o primeiro passo no trabalho foi eleger um agente que pudesse ser modificado de maneira a incorporar ataques e contramedidas. Como critérios que auxiliaram no processo de seleção, foram considerados:

- implementação completa do protocolo;
- popularidade do agente;
- não obsolescência;
- disponibilidade de código fonte;

- disponibilidade de documentação;
- possibilidade de vincular um endereço IP ao agente (opção geralmente denominada *bind*).

As cinco primeiras características são auto-explicativas. A última se refere aos cenários em que uma máquina executa vários agentes. É importante que cada agente possua um IP diferente, pois as contramedidas empregam o IP como um identificador do par.

Para a implementação dos ataques foi utilizada a Java BitTorrent API versão 1.1 [Dubuis, 2009]. Ela é uma API criada por Baptiste Dubuis que permite o desenvolvimento de aplicações que façam uso do protocolo BitTorrent. A escolha pela Java BitTorrent API baseou-se nos seguintes critérios: (a) boa documentação disponibilizada em seu código fonte; (b) possibilidade de execução em linha de comando; e (c) baixo consumo de recursos. O primeiro critério de escolha contribuiu para o entendimento e rápido desenvolvimento do ataque, permitindo explorar ataques mais eficientes. O segundo critério foi fundamental para automatização e execução em maior escala executados neste trabalho. Já o terceiro critério foi fator importante para instanciar ataques. Embora a lógica do ataque não seja sofisticada, o número elevado de atacantes no caso de Mentira em Massa exige uma aplicação leve, que consuma poucos recursos. Assim, considerando que nem todas as mensagens são tratadas pelo atacante, optou-se por um agente real, mas com uma implementação minimalista do protocolo.

Para a implementação das contramedidas escolheu-se um outro agente: BitTorrent versão 4.4 [Cohen, 2009a], doravante tratado como Mainline. Sua escolha levou em consideração (a) a maior aproximação com o protocolo original, posto que foi desenvolvido pelo autor do protocolo; (b) possibilidade de execução em linha de comando; e (c) um agente "leve" em termos de execução de software. As justificativas para o segundo e terceiro critérios são as mesmas apresentadas para escolha do primeiro agente.

Além dos agentes escolhidos, também foram considerados e avaliadas as seguintes implementações: (a) Azureus [Azu, 2009], (b) BitTornado [Bit, 2009a] e (c) Transmission [Tra, 2009]. O primeiro foi descartado em virtude de sua necessidade de recursos para execução. Para criação do enxame proposto neste trabalho e considerando os recursos computacionais disponíveis, o Azureus não se apresentou como uma alternativa viável. Já o BitTornado mostrou-se bastante promissor em quesitos de recursos de sistema. Além disso, por basear-se no agente BitTorrent original, enquadraria-se adequadamente aos critérios de seleção do agente. No entanto, o recurso para definição do IP para o agente (*bind*) não funcionou como o esperado, invalidando o uso do agente. O último agente considerado não possuía, até a versão avaliada, o recurso para definição de IP local. Somado a isso, seu

código possui uma documentação bastante escassa, dificultando a compreensão de sua estrutura.

5.2 Algoritmos originais

Esta seção descreve de forma sucinta os algoritmos originais dos agentes escolhidos para a implementação dos ataques e contramedidas criados neste trabalho. Esta visão é necessária para compreender as alterações implementadas e descritas nas seções conseguintes.

5.2.1 Java BitTorrent API

A fim de auxiliar a compreensão do algoritmo do Java BitTorrent API, foi criado um modelo simplificado das suas principais classes, conforme ilustrado na Figura 5.1. A execução é disparada através de uma classe *main*, recebendo como parâmetro o arquivo .torrent. Então é criada uma instância de **DownloadManager**, a classe responsável pelo gerenciamento do *download* e/ou compartilhamento do arquivo[1]. Entre os métodos disparados na classe, o startListening é responsável por criar um processo "ouvinte" para receber os pedidos de conexões dos pares remotos. Para isto, o método cria uma instância da **ConnectionListener**, para gerenciamento desses pedidos de conexões. Já o método startTrackerUpdate, de forma similar, cria um instância de **PeerUpdater**, tratando toda a comunicação realizada entre o par local e o rastreador.

Para cada conexão estabelecida com um par remoto, é criada uma instância da classe **DownloadTask**, sendo esta responsável pela troca de dados entre os pares. Para o tratamento das mensagens recebidas, é criada uma instância de **MessageReceiver**. De forma análoga, para o gerenciamento das mensagens enviadas, é instanciada a classe **MessageSender**.

Dentre as principais classes pode-se ainda destacar **Piece** e **Peer**. Estas classes possuem as principais propriedades que caracterizam as peças e os pares, bem como os métodos que permitem a interação com estas classes.

[1]Se o par local já possuir uma cópia completa do arquivo, ele atuará como semeador.

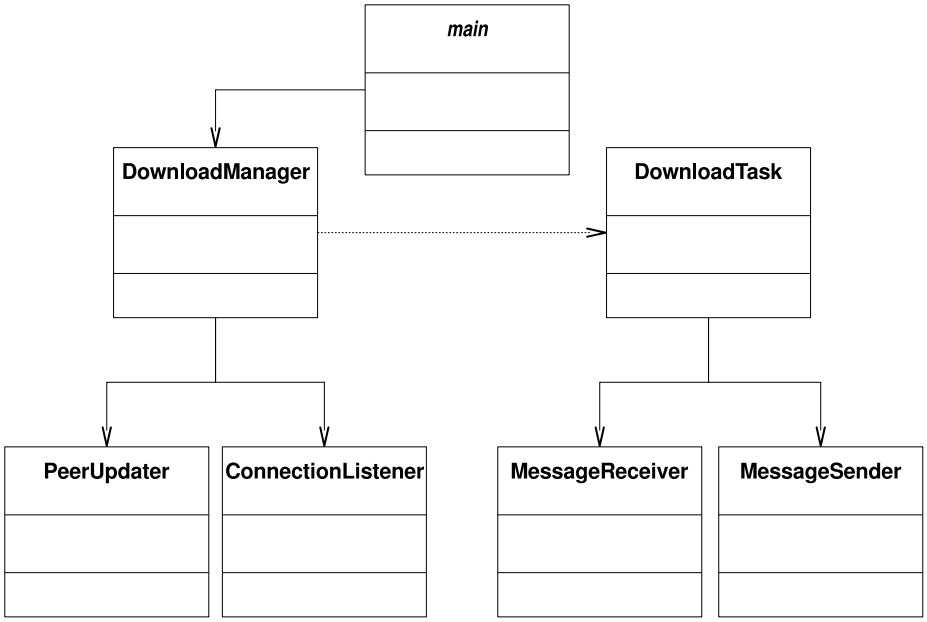


Figura 5.1 – Diagrama de classes simplificado da Java BitTorrent API.

5.2.2 Mainline

Sendo o Mainline um agente completo, a complexidade de seu código é um pouco maior se comparada à Java BitTorrent API. O núcleo principal do Mainline possui 116 classes e quase 13000 linhas de código. A descrição detalhada de suas classes e métodos não é o objetivo deste trabalho; assim, na Figura 5.2 são apresentadas apenas as classes principais do agente, que permitem seu entendimento e auxiliam na compreensão das implementações descritas nas seções seguintes.

A classe principal, pela qual se inicia a execução do Mainline, é a **Multitorrent**. Nela é instanciada a classe **RawServer**, responsável pelo gerenciamento da comunicação em nível de *sockets*. Através dos métodos desta classe é criado o "ouvinte" que receberá os pedidos de comunicação dos pares remotos. Na classe principal também é criada uma instância de **_SingleTorrent**, sendo esta responsável pelo gerenciamento do processo de *download*/*upload* do conteúdo.

A comunicação com o rastreador fica a cargo da classe **Rerequester**. Já a classe **Encoder** fica responsável pelo gerenciamento da comunicação entre os pares, controlando o total de conexões estabelecidas, substituindo conexões perdidas e enviando mensagens de KeepAlive. O gerenciamento de peças solicitadas dos pares remotos é responsabilidade da classe **Upload**, que trata mensagens como Interested ou Not_Interested enviadas pelos pares.

A classe **SingleDownload** realiza o gerenciamento de requisições de peças

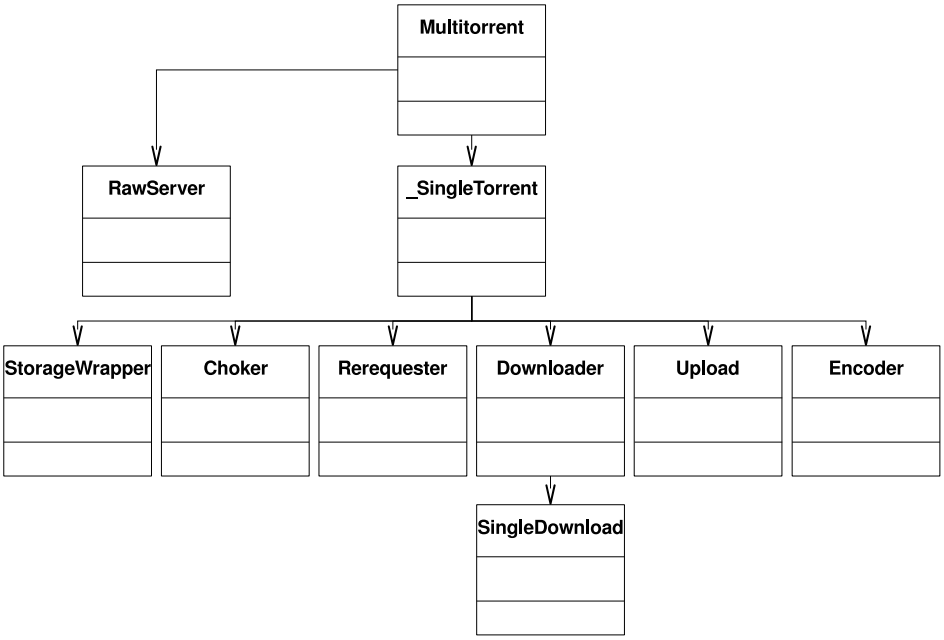


Figura 5.2 – Diagrama de classes simplificado do Mainline.

que são do interesse do par local. Ela trata mensagens de CHOKED, UNCHOKED e HAVE e requisita novas peças para os pares remotos. No entanto o tratamento do bloco recebido é executado através de **StorageWrapper**, sendo esta responsável pelo armazenamento em arquivo e verificação de *hash* em peças completadas. Por fim, o controle de melhores pares, feito através do *optimistic unchoking*, é de responsabilidade de **Choker**.

A exploração do código do Mainline permitiu ainda a compreensão da contramedida nativa existente nesse agente, cujo objetivo é eliminar pares que contribuam com dados incorretos. O mecanismo de defesa nativo só entra em ação quando um par remoto foi o único contribuidor para a peça. Como primeira medida, o par local desconecta o par remoto que enviou os dados da peça corrompida e armazena estatísticas sobre o par remoto (acumulador de peças falhadas). Se o mesmo par remoto voltar a contribuir com 3 ou mais peças corrompidas, e caso não tenha contribuído com um número suficiente de peças íntegras[2], então este par será banido e não poderá conectar-se novamente com o par local.

Para execução dos cenários sem ataque e sem contramedida foi utilizado o Mainline como agente. Salvo pequenas modificações criadas para geração de "logs"

[2]Conforme o algoritmo, a razão de contribuição é elevada: `len(self.stats.bad) > self.stats.numgood // 30`

de peças recebidas, o código utilizado é o mesmo disponibilizado pelo criador do agente.

5.3 Ataque de Mentira em Massa

Como descrito na Seção 3.3, o segredo deste ataque é o elevado número de pares conectados no enxame, através de identidades falsas. Se por um lado este tipo de ataque não demanda um grande consumo de recursos de rede por par atacante, por outro o gerenciamento de um número elevado de conexões que podem ser estabelecidas representou um desafio a ser tratado. Na implementação original da Java BitTorrent API, cada conexão com um par remoto cria 3 *threads* para o gerenciamento desta comunicação. Se o par local estiver conectado com 50 pares, por exemplo, seriam pelo menos 150 *threads* criadas. No entanto, este número ainda precisaria ser multiplicado pelo número de *Sybils* que o atacante deseja utilizar. Por isto, o gerenciamento das conexões foi modificado em relação à Java BitTorrent API, como descrito a seguir.

Inicialmente foram criados 2 novos parâmetros que são passados na chamada do ataque desenvolvido: o IP inicial do atacante e o número de *Sybils* que deverão ser criados. Desta forma, cada novo *Sybil* terá um IP sequencial a partir do IP inicial.

A classe **DownloadManager** teve seu construtor alterado, passando a receber o número de *Sybils* que serão executados. Os objetos que antes eram instâncias únicas passaram a ser vetores, com uma instância para cada identidade falsa criada. Uma característica do ataque de mentira é o atacante mentir sobre a posse de N peças. Assim, ao ser criada a instância da classe **DownloadManager**, o construtor cria N falsas peças, para despertar o interesse dos atacados. O método startTrackerUpdate, responsável por estabelecer a comunicação entre o atacante e o rastreador, foi alterado para que todos os *Sybils* criassem a conexão com o rastreador. Além disso, cada novo atacante deve chegar no enxame em um dado intervalo de tempo, ficando a cargo deste método tal controle. Outro método que foi modificado nesta classe foi o updatePeerList, que é responsável por receber a PeerList do rastreador e tentar estabelecer a conexão com os pares ainda não conectados. Para maximizar o efeito do ataque, sempre que algum *Sybil* recebe a lista de pares do rastreador, para cada par nesta lista, o atacante faz com que seus Sybils tentem conectar com esse par[3]. Desta forma, a visão de pares do atacante é ampliada. Dependendo do número de *Sybils*, o atacante pode ter uma visão completa do enxame. Ao identificar um par ainda não conectado, o método dispara a chamada de outro,

[3]Exceções são os IPs dos demais *Sybils*, com os quais não serão disparadas solicitações de conexão.

denominado connect. Este método foi modificado para que a identidade responsável pela comunicação pudesse ser identificada. A inicialização de um "ouvinte" com o objetivo de receber os pedidos de conexão é feita pelo método startListening, que também foi modificado a fim de criar-se uma *thread* para cada *Sybil* do atacante. Por fim, os pedidos de conexões dos pares para os atacantes são tratados pelo método connectionAccepted, que foi modificado para identificar de qual identidade foi solicitada a conexão.

Na implementação original, a comunicação entre os pares e o par local é realizada através das classes **DownloadTask**, **MessageReceiver** e **MessageSender**. Para o ataque de mentira, foi criada uma nova classe, chamada **SocketData**, para substituí-las. Além de unificar as classes em uma única, esta nova não é mais uma *thread* em modo de execução permanente, mas sim uma *thread* executada sob demanda. Assim, ela assume o papel de criar os objetos de comunicação, ler as mensagens recebidas e enviar somente as mensagens características deste ataque. A leitura das mensagens trata apenas o recebimento do HANDSHAKE e do KEEPALIVE. E as mensagens enviadas pelo atacante são: HANDSHAKE, BITFIELD e KEEPALIVE. Isto se deve à característica do ataque, que apenas deseja manter a conexão da vítima ocupada, não intencionando nenhuma troca de dados.

Outra classe acrescentada foi **SocketManager**, responsável por manter um vetor das conexões estabelecidas. Esta nova classe cria também as instâncias das classes **KeepAliveSender** e **MessageIgnoreReader**. Esta última tem a responsabilidade de ler os dados dos *sockets* e enviá-los para serem tratados pelo método readMessage da classe **SocketData**, que por sua vez é responsável por filtrar as mensagens desejadas. Já a classe **KeepAliveSender** tem o papel de enviar uma mensagem KEEPALIVE para todos os pares conectados com todas as identidades falsas do atacante, mantendo, assim, ativa a conexão dos pares atacados.

5.4 Ataque de Corrupção de Peças

A implementação do ataque de Corrupção de Peças criada neste trabalho difere um pouco da apresentada em [Schmitt et al., 2008]. Nesse estudo anterior, o atacante obteve sucesso gerando sobrecarga na rede.

Ao criar uma nova estratégia de ataque para o corruptor o atacante, além de gerar sobrecarga na rede, consegue também comprometer o enxame com resultados próximos aos apresentados em [Bauermann et al., 2008]. Esta nova estratégia difere do ataque anterior pois a cada intervalo entre *unchokeds*, antes de enviar a mensagem para deixar de sufocar o par atacado, o corruptor envia novamente uma mensagem CHOKED e em seguida uma mensagem UNCHOKED. A alteração justifica-se pelo comportamento observado no agente honesto avaliado. Ao receber uma mensagem CHOKED, o par local limpa as requisições para este par e busca em outras

pares disponíveis a continuação do *download* da peça. Em seguida o atacante se coloca disponível para oferecer as peças novamente.

Como já comentado na Seção 5.1, uma característica fundamental para a condução dos experimentos é a possibilidade de atribuição de um IP local ao agente. Embora disponível em alguns agentes, esta característica não era suportada na versão original da biblioteca escolhida para o ataque. Para implementar tal característica, as classes **ConnectionListener**, **DownloadTask** e **PeerUpdater** (responsáveis pela criação dos *sockets* de comunicação) tiveram seus métodos alterados. Além destes, os métodos connect, connectAccepted, startListening e startTrackerUpdate da classe **DownloadManager** tiveram sua lógica alterada para suportar a atribuição de IP para o agente.

Para atender à lógica do ataque, a classe **DownloadManager** teve alguns de seus métodos reescritos. O testComplete, responsável por verificar se as peças estão completas, foi modificado para que sempre retornasse verdadeiro. Assim, o arquivo é criado integralmente (mesmo não existindo fisicamente), para que o atacante se anuncie como semeador (já que possui uma cópia completa do arquivo). Já no método getPieceBlock a mudança do código foi feita para criar um bloco com dados aleatórios, para forçar a corrupção da peça. O método unchokePeers, originalmente destinado a escolher os pares com melhores contribuições, foi modificado para que enviasse UNCHOKED para todos os pares. Por fim, no método peerRequest, responsável por tratar as requisições dos pares remotos, foi modificado para que enviasse apenas 1 bloco corrompido. Após o envio do bloco, é enviada também uma mensagem CHOKED.

Encerrando as modificações, a classe **MessageSender**, cuja tarefa é o envio de mensagens, teve seu tratamento de mensagem KEEPALIVE alterado. No lugar desta mensagem, em intervalos definidos na estratégia de ataque, são enviadas mensagens CHOKED e UNCHOKED para os pares conectados. Esta modificação considerou que o intervalo entre envio destas mensagens é inferior ao intervalo utilizado para envio de KEEPALIVE, cujo valor padrão é 120 segundos.

5.5 Contramedida Rotação de Pares

A fim de minimizar o impacto das alterações sob o código original do Mainline, foi criada uma nova classe, denominada **PeerRotation**, específica para a implementação do algoritmo da contramedida de Rotação, cuja descrição encontra-se em 4.2. Nesta classe foram definidas as estruturas de dados necessárias para avaliação dos pares remotos, como tempo de conexão, contribuição e tempo em quarentena. Na inicialização da classe **Multitorrent** é instanciada a nova classe criada. Assim, todas as classes que precisam interagir com a contramedida passam a receber a referência do objeto criado.

A contramedida se baseia na quantidade de dados que é enviada ou recebida de cada par remoto. Para controlar o envio de dados a pares remotos, na classe responsável pelo gerenciamento de *uploads* (**Upload**), sempre que houver envio de blocos, os mesmos serão contabilizados através do método inc_remote_peer da classe da contramedida. De forma análoga, tal controle é realizado no gerenciamento de *downloads* do par local (classe **SingleDownload**). Note-se que nesse caso não há distinção entre blocos enviados e recebidos.

Pares que não colaboram e insistem em permanecer ociosos podem ser rotacionados e colocados em quarentena. A classe **Encoder**, responsável pelo gerenciamento de conexões, foi modificada para inicializar contadores dos pares, substituir pares e verificar se novas conexões não são de pares em quarentena. Os métodos start_connection e singleport_connection passam a verificar se o par com o qual a conexão está sendo estabelecida não está em quarentena. No método connection_completed, responsável por registrar o estabelecimento da conexão, são inicializados os contadores de controle da contramedida. E o método replace_connection foi modificado para criar a lógica de conexão com os "pares conhecidos" pelo par local, no intuito de substituir conexões rotacionadas.

O algoritmo principal de avaliação e eventual rotação dos pares foi implementado na classe **PeerRotation**. O algoritmo é executado a cada intervalo de tempo, definido conforme os parâmetros do modelo de avaliação a ser executado.

Durante a implementação desta contramedida, percebeu-se que uma premissa adotada em trabalhos anteriores baseados em simulação não poderia ser considerada no agente real. Na simulação, quando um par remoto conecta com o par local, aquele par é acrescentado ao conjunto de "pares conhecidos" do par local. Embora seja "conhecido" pelo par local, este par que inicia a conexão com o par local não fornece informações sobre sua porta de "escuta" (pois a iniciativa de conexão foi dele), impossibilitando uma reconexão com o mesmo. Assim, o conjunto de pares conhecidos na implementação real é menor que o encontrado em simulação. Apesar desta modificação, não foram percebidas diferenças significativas nos resultados das execuções conduzidas.

Outra modificação introduzida em relação à contramedida proposta foi o critério de desconexão dos pares que serão rotacionados. Pares são desconectados conforme suas taxas de troca de dados. Originalmente, pares com as mesmas taxas eram selecionados em ordem aleatória. Foi criado um critério de desempate para desconexão: tempo de conexão. Desta forma, dentre os pares remotos com as menores taxas de contribuição, são desconectados primeiro aqueles que estão a mais tempo conectados.

5.6 Contramedida Anticorrupção

De forma semelhante à contramedida anterior, a Anticorrupção foi criada em uma nova classe, denominada **AntiCorruption**. Diferentemente do caso anterior, onde o algoritmo de avaliação consta na própria classe criada, nessa contramedida as ações de recuperação são executadas sempre que o par local receber uma peça corrompida, conforme a lógica da contramedida detalhadamente descrita em 4.3.

Dessa maneira, a classe **StorageWrapper**, responsável pelo controle de requisições e recebimento de peças, teve seu método piece_came_in alterado. O algoritmo Anticorrupção foi incorporado ao método. Assim, ao receber completamente uma peça, íntegra ou não, este método interage com a nova classe criada, **AntiCorruption**, para ajustar a reputação de pares e/ou criar as estruturas de controle necessárias para recuperação da peça. Além disso, a contramedida nativa existente no método foi desabilitada.

A classe **SingleDownload** é responsável pelo gerenciamento de *downloads*. Seus métodos got_piece e _request_more foram modificados para que durante o processo de recuperação de uma peça falhada fossem disparadas requisições somente para o último contribuidor da peça em questão. Também foi necessário criar um novo método para limpar as requisições de uma peça não íntegra de todos os pares que não sejam o último contribuidor. Os métodos disconnect e got_choke foram alterados para decrementar a reputação de um par que envie tais mensagens e possua peça(s) em estado de recuperação.

Finalizando, de forma similar ao realizado na outra contramedida, a classe **Encoder** teve os métodos start_connection e singleport_connection alterados para não permitir a conexão de pares banidos pelo algoritmo Anticorrupção.

Capítulo 6

Avaliacao através de simulação

Os algoritmos apresentados nas Seções 4.2 e 4.3 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., 2008b, Mansilha et al., 2008]. Através de uma campanha de simulações, foi avaliada a *eficácia* das contramedidas 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 contramedidas propostas ao lidar com as situações de ataque citadas na questão anterior, comparando-se aos casos de ataque sem e com contramedida? Além disso, comparando o caso com ataque e com contramedida, quão distante ele fica do caso ótimo, representado pelo cenário sem ataque e sem contramedida?

Q3. Qual a eficiência das contramedidas, em termos de atrasos potencialmente induzidos com a introdução de uma contramedida?

Q4. Os mecanismos de detecção das contramedidas 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.

6.1 Modelo de avaliação

O conjunto de cenários avaliados e discutidos neste trabalho 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) contramedida. As duas primeiras categorias são comuns a todos os casos, pois independem de ataques e contramedidas. 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, simétrica.

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*, 30 s; 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.

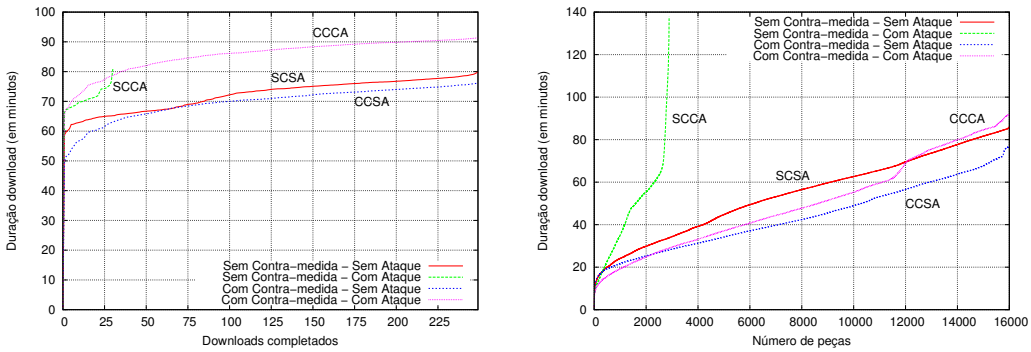
De acordo com o ataque "mentira em massa", as identidades falsas são controladas por um único par malicioso, compartilhando a informação dos P entre o *Sybils*. Cada novo *Sybil* inicia uma conexão com o rastreador, registra-se e recebe uma nova lista L. Esta lista é adicionada ao conjunto de pares conhecidos, removendo os próprios *Sybils* da lista: $P \leftarrow P \cup (L \setminus M)$. Os parâmetros que descrevem um *ataque* são definidos a seguir. Para o ataque de "mentira em massa": número de pares atacantes, 500 *Sybils*; tempo de chegada do atacante, 0,1 s (logo após o semeador inicial); taxa de criação de *Sybils* pelo atacante, 1 *Sybil*/3 s; larguras de banda do par malicioso (comum aos *Sybils*), 8 Mbps, simétrica; número de peças mentidas, 16. Em relação ao ataque de corrupção, os pares maliciosos chegam a cada 3 s a partir do tempo 0; o intervalo entre transmissões de mensagens UNCHOKE por cada par atacante é 4 s. A escolha dos parâmetros baseou-se em 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 *contramedidas* são listados a seguir. A contramedida 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 contramedida 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.

6.2 Avaliação da Rotação de Pares

Esta subseção apresenta a avaliação da eficácia e eficiência da Rotação de Pares. A Figura 6.1 apresenta os resultados. Na Figura 6.1(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 Contramedida-Com Ataque" significa que o enxame falhou com 30 pares completos no tempo 80,8 min. A Figura 6.1(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 Contramedida – Sem Ataque" (SCSA), "Sem Contramedida – Com Ataque" (SCCA), "Com Contramedida – Sem Ataque"

(CCSA), e “Com Contramedida – Com Ataque” (CCCA).



(a) Pares com downloads completados

(b) Contador global de peças corretas recebidas

Figura 6.1 – Avaliação de Rotação de Pares em cenários com e sem ataque

A comparação de ambas as curvas sem contramedidas, *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 6.1(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 contramedida, 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 6.1(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 contramedida.

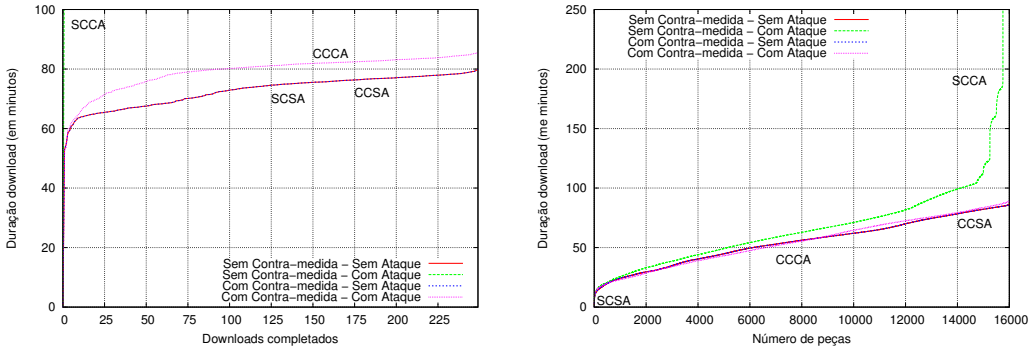
A questão Q2, sobre a eficácia da contramedida, é respondida comparando-se na Figura 6.1(a) as curvas que representam cenários com ataque: SCCA e CCCA. Em CCCA todos os 250 pares completam o download; o 250^o par termina aos 93 min, apenas 13 min depois do caso ideal SCSA. Examinando o mesmo par de curvas, SCCA e CCCA, na Figura 6.1(b), fica claro que a contramedida (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 contramedida de Rotação, compara-se na Figura 6.1(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 contramedida (SCSA). O motivo para esta vantagem nos downloads pode ser confirmada na Figura 6.1(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.

6.3 Avaliação da Anti-corrupção

As três questões da subseção anterior são discutidas aqui em relação à contramedida Anti-corrupção. A Figura 6.2 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.



(a) Pares com downloads completados

(b) Peças corretas recebidas

Figura 6.2 – Avaliação da Anti-corrupção

O primeiro ponto que se destaca na Figura 6.2(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

6.2(b). Apesar do ataque, os pares honestos conseguem receber 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.

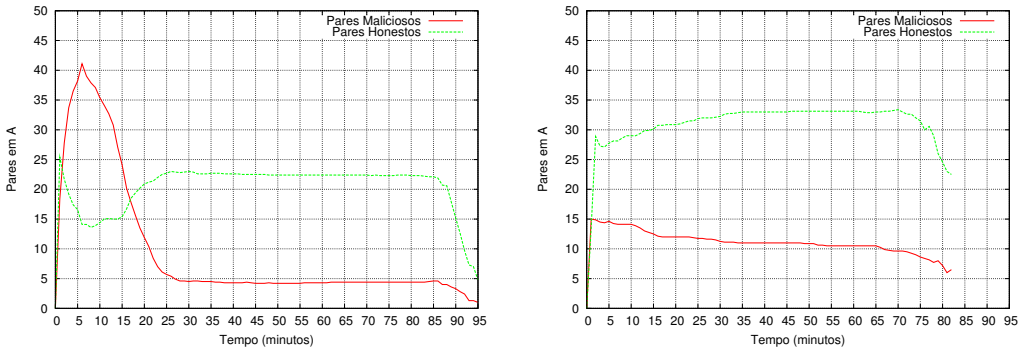
Para responder a Q2, compara-se na Figura 6.2(a) ambas as curvas com ataque: SCCA – já discutida – e CCCA. Enquanto pouquíssimos pares completam SCCA, já em CCCA todos os pares completam com sucesso em até 86 min. Também observa-se que a curva CCCA permanece muito próxima (cerca de 10 min) de SCSA. A 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 6.2(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 6.2(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.

6.4 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 contramedida, 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 6.3 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 6.3(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 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 hones-



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

Figura 6.3 – Eficácia das contramedidas em detectar pares maliciosos

tos. 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 6.3(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 contramedida, 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 contramedida proposta é bastante eficaz em detectar e isolar pares corruptores.

Capítulo 7

Experimentos de avaliação

Neste capítulo são apresentados os resultados da implementação dos ataques e contramedidas em agentes de usuário BitTorrent existentes. Através de uma campanha de execuções, foram avaliadas tanto a eficácia das contramedidas em situações com ataque, bem como sua eficiência em situações sem ataque. O objetivo da avaliação, de forma análoga às questões consideradas no estudo por simulação [Barcellos et al., 2008a, Bauermann et al., 2008] e apresentadas no Capítulo 6, é responder as seguintes questões fundamentais:

Q1. Qual o impacto dos ataques sobre um enxame em relação ao caso sem ataques?

Q2. Qual a eficácia das contramedidas frente aos ataques, comparando aos casos de ataque sem contramedida? E quão distante este caso com ataque e com contramedida fica do caso ótimo, representado pelo cenário sem ataque e sem contramedida?

Q3. Qual a eficiência das contramedidas, em termos de atrasos potencialmente induzidos com a introdução de uma contramedida?

Além disso, foi avaliado o grau de eficiência dos ataques desenvolvidos em comparação com o caso ótimo. Nas seções seguintes são descritos o ambiente e modelo de avaliação para condução dos experimentos. Na sequência são apresentados e discutidos os resultados obtidos desta avaliação.

7.1 Ambiente

Os experimentos apresentados neste trabalho foram todos executados no laboratório do Programa Interdisciplinar de Pós-Graduação em Computação Aplicada da Universidade do Vale do Rio dos Sinos. Para a execução foram utilizados

12 computadores PC ligados em rede *Fast Ethernet*. As máquinas utilizadas possuem cada uma um processador Intel Pentium 4 1,8 GHz, 1 GB de memória RAM, disco rígido IDE de 40 GB e placa de rede de 100 Mbps. Todas utilizam o Gentoo Linux como sistema operacional com *kernel* 2.6.

Para execução dos experimentos, utilizou-se como ponto de partida um ambiente criado por Flávio Santos[1]. Esse ambiente permite a execução de múltiplos agentes em uma mesma máquina, controlando o tempo entre chegadas dos pares e suas opções de inicialização, conforme o tipo de agente a ser executado. A execução de forma dinâmica em diferentes cenários é feita através da criação de um arquivo de parâmetros que define a quantidade e as características dos pares que compõem o enxame. A Tabela 7.1 lista as propriedades dos pares descritas no arquivo de parâmetros.

Tabela 7.1 – Parâmetros para caracterização de pares do enxame.

	Descrição
1	Tempo de chegada do par no enxame (em segundos)
2	IP definido para o par
3	Tipo de agente que será executado
4	Taxa de *upload* do par
5	Taxa de *download* do par
6	Razão upload:download (*contribution ratio*) alvo

A Figura 7.1[2] apresenta o diagrama de classes do ambiente para a criação do enxame. A execução é iniciada através da passagem do arquivo de parâmetros para a classe **Main**. Ela faz a carga do arquivo, lendo as características de cada par e sua adição no enxame, através do método addPeer da classe **Torrent**. Além de criar os pares (instâncias de **Peer**), a classe **Torrent** é responsável por controlar o tempo de entrada do par no enxame, bem como o tipo de agente a ser executado. O tipo de agente é uma propriedade da classe **Peer**, que permite fazer a chamada do agente padrão, dos agentes com contramedidas ou ainda dos atacantes. É criado então um novo processo para cada par, respeitando suas características, tais como taxas de *upload* e *download*. As mensagens de retorno do processo criado são filtradas pela classe **BitTorrentStreamConsumer**. Desta forma, são processadas e armazenadas somente as mensagens de saída que são importantes para traçar o comportamento dos pares no enxame, diminuindo consideravelmente o volume de "logs" das execuções.

[1] Aluno de mestrado da UFRGS e orientando dos Profs. Marinho Barcellos e Luciano Gaspary.

[2] Para melhor visualização, os construtores das classes **Peer** e **Torrent** tiveram sua assinatura omitida.

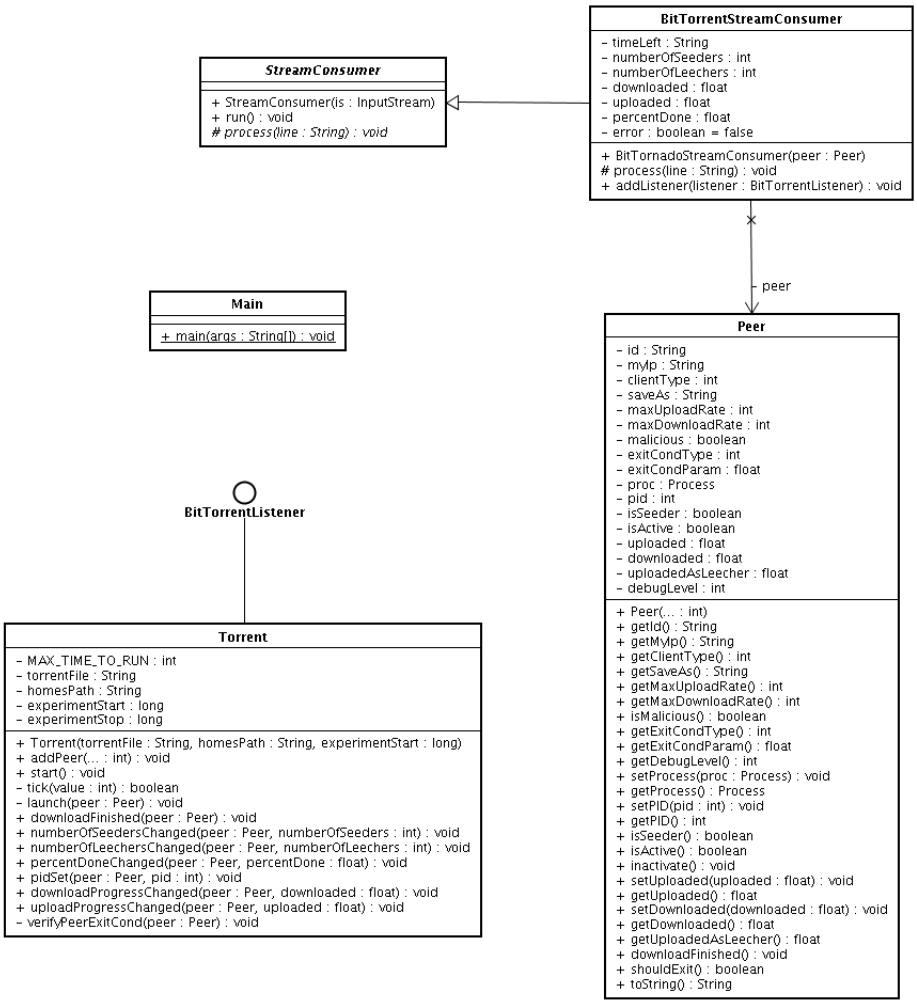


Figura 7.1 – Diagrama de classes do ambiente de execução.

Observa-se pelo arquivo de parâmetros descrito pela Tabela 7.1 que cada par no enxame pode possuir um IP único. Embora fosse possível rodar os pares no mesmo IP, usando portas diferentes, optou-se por uma configuração onde cada par possui um IP diferente, conforme necessidade das contramedidas criadas. Além disso, esta configuração visa também aproximar o ambiente controlado de um ambiente real.

Para tornar isto possível, empregou-se o mecanismo de "interfaces de redes virtuais" disponível no *kernel* do sistema operacional, criando um cenário onde

podem ser executados até 254 agentes com IPs diferentes em cada máquina do ambiente. Com exceção do ataque de Mentira em Massa (que utilizou 250 IPs em cada máquina), todos os demais cenários executados ocuparam, no máximo, 25 IPs em cada máquina.

Como rastreador para o enxame, optou-se pelo BNBT EasyTracker 7.7 Release 3 [Hogan, 2009]. A escolha pelo software considerou sua baixa exigência de recursos, bem como a possibilidade de execução através de linha de comando, sem a necessidade de uma interface gráfica. Foram utilizadas as configurações padrões do BNBT, alterando-se apenas a interface (IP) de comunicação do rastreador e o número máximo de pares (parâmetro *bnbt_max_peers_display*) para 1000.

Embora o ataque de Mentira em Massa, descrito na Seção 3.3, não exija um grande volume de recursos de banda por parte do atacante, sua implementação comprovou a necessidade de recursos computacionais superiores aos disponíveis em uma configuração padrão de sistema operacional. Em virtude do número elevado de identidades falsas criadas pelo atacante, o conjunto de *sockets* criados para comunicação entre os pares maliciosos e os atacados é bastante elevado. Desta forma, foi necessário incrementar o número de arquivos abertos (*open files*) para que este ataque pudesse ser executado com sucesso. Para os experimentos conduzidos neste trabalho, o parâmetro foi incrementado para 65536, pois o limite padrão (*open files* = 1024 nas máquinas utilizadas) não foi suficiente para execução do ataque.

7.2 Modelo de avaliação

As características do enxame empregado nesta avaliação experimental são as mesmas usadas em simulação. Conforme apresentado em [Barcellos et al., 2008a, Bauermann et al., 2008], há 250 pares honestos e 1 semeador inicial e permanente; a razão de contribuição alvo é 1,0 para sugadores e ∞ para o semeador; o conteúdo tem 64 peças, sendo cada peça composta por 64 blocos de 16KB; as larguras de banda do semeador inicial e dos sugadores é de 1024 Kbps para *download* e 256 Kbps para *upload*; o número mínimo de conexões foi definido como 30 e o máximo como 50. O tempo máximo para execução de experimentos é de 5 horas (300 minutos) reais.

O mentiroso chega logo após o semeador inicial, em 0,3 s; o atacante cria um *Sybil* a cada 3s, de forma a aproveitar os diferentes relatórios enviados pelo rastreador; larguras de banda do par malicioso (comum aos *Sybils*), 8 Mbps, simétrica; número de peças mentidas, 16. No ataque de Corrupção, os pares maliciosos chegam a cada 3s a partir do tempo 0; o intervalo entre transmissões de mensagens UNCHOKED por cada par atacante é de 8 s.

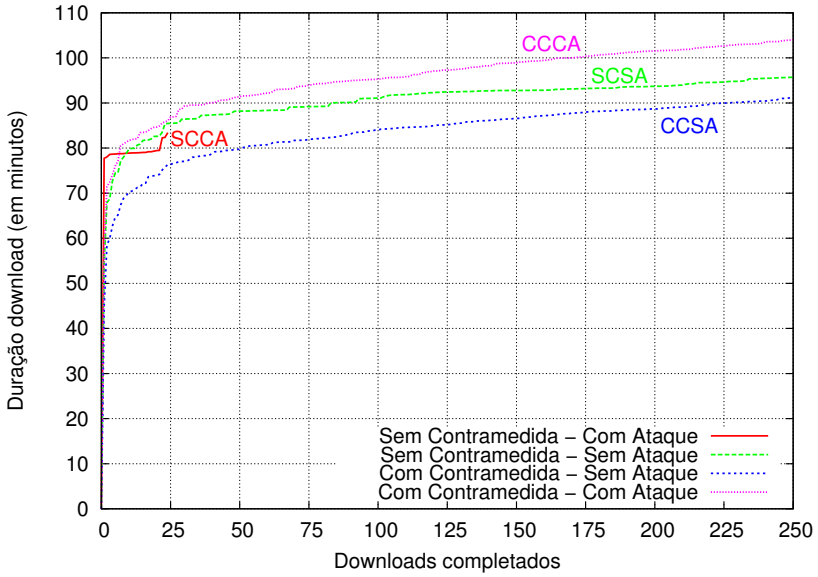
A contramedida de Rotação foi avaliada com as seguintes escolhas: periodicidade de avaliação do algoritmo, 1 minuto; tempo inicial de quarentena, 4 ciclos de

avaliação; tempo de carência antes de avaliar taxa de troca com um par, 5 minutos; fator geométrico de crescimento da quarentena, 2,0; taxa de transferência mínima exigida, 0,2 Kbps. Os parâmetros para a contramedida Anticorrupção são: reputação inicial de pares, 0,3 (na simulação o mecanismo foi mais permissivo, com reputação inicial igual a 0,5); fator de decremento base a ser aplicado perante evidência negativa, 0,2; e fator de incremento base para evidências positivas, 0,1.

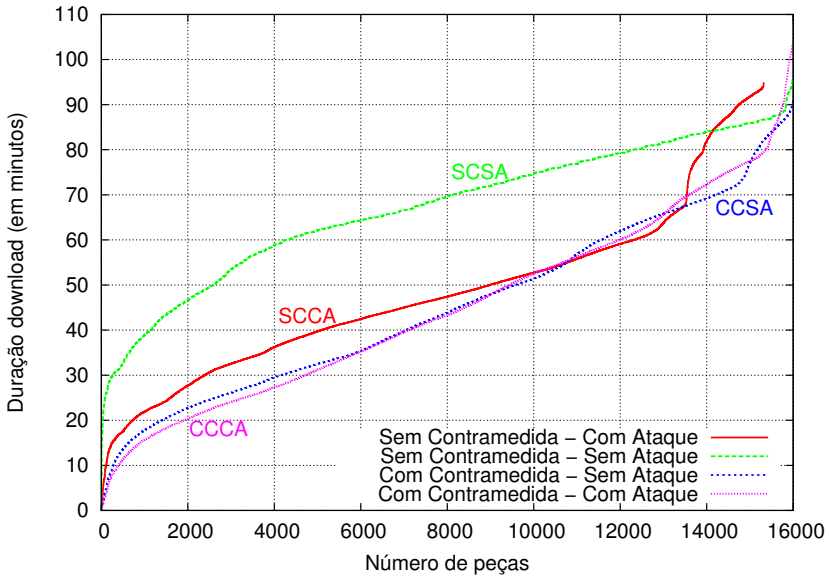
7.3 Avaliação da Rotação de Pares

A Figura 7.2 apresenta os resultados da avaliação da eficácia e eficiência para o caso da Rotação de Peças. Na Figura 7.2(a) os eixos x e y são, respectivamente, o número de pares que possuem seus *downloads* completados e o tempo em minutos que o download demandou. Para considerar um enxame completado com sucesso, todos os 250 pares precisam completar seus *downloads*, ou seja, a curva deve alcançar o ponto $x = 250$ (e quanto menor o valor em y, melhor). Já na Figura 7.2(b) o eixo x mostra o número global de peças corretas já recebidas e o eixo y mostra o tempo em minutos necessário para fazer o *download* desse número de peças. Como no gráfico anterior, o enxame é considerado completado com sucesso quando o eixo x atingir o ponto $x = 16000$ (e quanto menor o valor em y, melhor). Em ambas as figuras existem quatro curvas refletindo os casos de interesse: "Sem Contramedida – Sem Ataque" (SCSA), "Sem Contramedida – Com Ataque" (SCCA), "Com Contra-medida – Sem Ataque" (CCSA), e "Com Contramedida – Com Ataque" (CCCA).

Para responder a questão Q1, sobre o impacto de um ataque sem proteção, pode-se comparar as curvas com e sem o ataque de mentira em massa. Como pode ser visto na Figura 7.2(a), a curva SCSA indica que o primeiro par completa seu *download* por volta de 70 minutos e o último por volta de 95 minutos. Já a curva SCCA mostra que apenas 24 pares do enxame sob ataque conseguem completar seu *download* entre cerca de 78 e 84 minutos. Os demais 226 pares falham ao tentar baixar seu conteúdo. Tais resultados confirmam as conclusões encontradas através de simulação, pois as curvas possuem a mesma tendência. A diferença notável com as curvas encontradas em simulação é que essas foram um pouco mais baixas, indicando que o *download* foi mais rápido. O presente trabalho mostra que, na realidade, o término do *download* é um pouco mais demorado.



(a) Pares com *downloads* completados



(b) Número de peças corretas recebidas

Figura 7.2 – Avaliação da Rotação de Pares.

Em relação às peças completadas, a Figura 7.2(b) mostra que o enxame sob o efeito do ataque, apesar de conseguir baixar a quase totalidade das peças, falha na recepção das peças finais. Em aproximadamente 95 minutos são obtidas cerca de 15300 peças. Mas a partir deste ponto não existe mais evolução no enxame,

impedindo assim que 226 pares consigam completar seu conteúdo. Isto acontece porque o semeador permanente foi quase totalmente eclipsado por atacantes, e os pares que conseguiram conectar-se com o semeador deixaram o enxame ao receber suas últimas peças (pois atingiram sua razão alvo). Desta forma, não existem mais pares que possam fornecer as peças finais para os demais sugadores. Embora este cenário seja diferente do encontrado através de simulação, onde um número significativamente menor de peças é obtido, ele demonstra que o ataque implementado não deixou de ser efetivo, pois ainda consegue impedir o sucesso de 90% do enxame. Assim, em resposta à questão Q1, pode-se afirmar que há um grande impacto de um ataque de "mentira em massa" com 500 *Sybils* sobre um enxame com 250 pares honestos rodando sem a proteção de uma contramedida.

Para responder a questão Q2, sobre a eficácia da contramedida, analisa-se na Figura 7.2(a) as curvas de cenários com ataque: SCCA e CCCA. Observa-se primeiro que com a contramedida ativa, todos os 250 pares conseguem completar seu conteúdo. Além disso, que o 250^o conclui seu *download* aos 104 minutos, apenas 9 minutos depois do caso ideal, representado pela curva SCSA. Ao examinar as mesmas curvas na Figura 7.2(b), percebe-se a eficácia da contramedida desenvolvida. Em comparação ao caso ótimo, sem ataque e sem contramedida, o algoritmo implementado atua bem, apenas 9% mais lento quando o número de *Sybils* é o dobro do número de pares honestos.

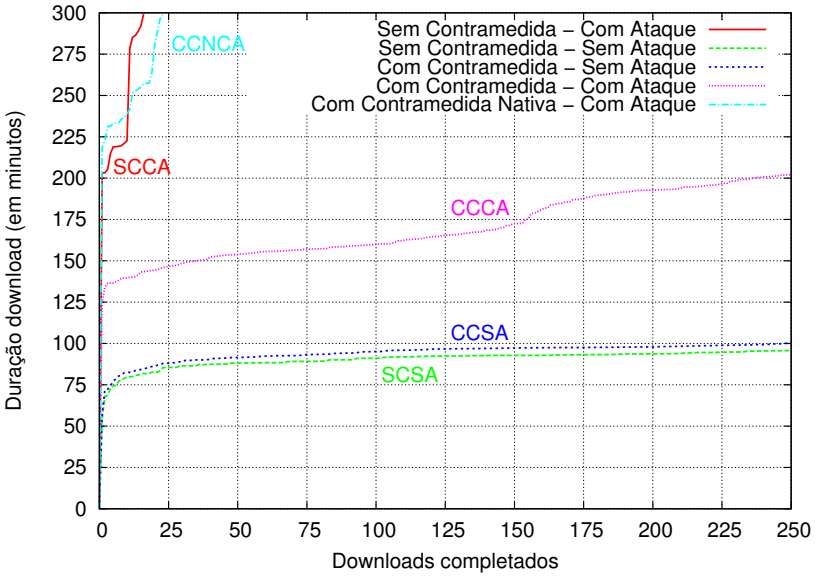
Conforme já mencionado, na comparação com as curvas correspondentes no caso simulado, pode-se perceber que há um número maior de peças carregadas no caso com ataque e sem contramedida. Observa-se também que a curva SCSA tem um início mais lento, enquanto as demais curvas iniciam mais cedo, mas acabam enfrentando dificuldades para completar suas últimas peças, aproximando-as da curva SCSA.

Respondendo a questão Q3, sobre uma possível sobrecarga gerada pela contramedida, compara-se na Figura 7.2(a) as curvas que representam os cenários sem ataque: SCSA e CCSA. O comportamento encontrado em simulação foi reproduzido na implementação em agentes reais, ou seja, o tempo para conclusão do enxame em cenários com contramedida é ligeiramente superior comparado ao caso ótimo. Tal deve-se à lógica do algoritmo, que pode rotacionar uma parcela dos piores pares (com menor taxa de contribuição). Esta vagas disponibilizadas podem ser ocupadas por pares melhores, auxiliando na evolução do *download*. O mesmo comportamento pode ser observado na Figura 7.2(b), onde a curva representada pelo cenário com contramedida chega a apresentar até 20 minutos de vantagem, mas se aproximando do caso ótimo nas peças finais.

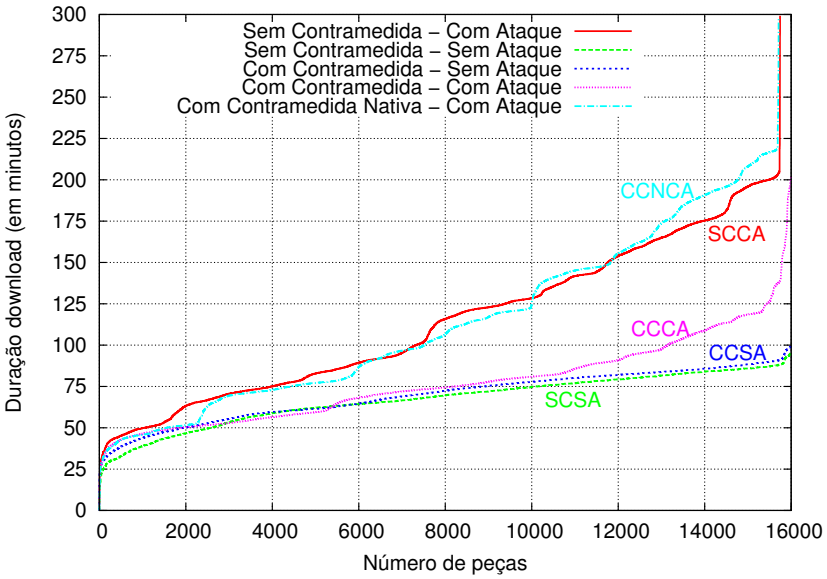
7.4 Avaliação da Anticorrupção

As questões da seção anterior são aqui discutidas em relação à contramedida Anticorrupção. A Figura 7.3 apresenta os gráficos de um cenário com ataque de corrupção de peças com 15 atacantes. Os eixos x e y possuem o mesmo significado dos gráficos da Seção 7.3. Entretanto, diferentemente da simulação, nestes experimentos há uma quinta curva: "Com Contramedida Nativa – Com Ataque" (CCNCA). Esta curva reflete o mecanismo "nativo" de proteção contra ataque de corrupção existente no agente de usuário Mainline. Embora simples, o mesmo serve de comparação com a contramedida implementada neste trabalho.

Ao analisar as curvas com ataque e sem contramedida e com contramedida nativa, representadas na Figura 7.3(a) por SCCA e CCNCA, respectivamente, destaca-se o impacto do ataque. No caso sem contramedida, apenas 16 pares conseguem completar seu *download*, e no outro caso apenas 23 pares. Além da eficácia do ataque em comprometer o enxame, pode-se perceber o atraso gerado pelo ataque nestes dois cenários: o primeiro par, entre os que completam seu conteúdo, o faz somente depois de 200 minutos, em contraste ao caso ótimo (curva SCSA), onde o último par completa seu *download* por volta de 95 minutos. Assim, respondendo à questão Q1, o impacto do ataque é bastante elevado. Esse cenário assemelha-se ao encontrado em simulação, validando nossos resultados anteriores. Por outro lado, as curvas de peças completadas desses cenários avaliados, mostradas pela Figura 7.3(b), divergem das curvas encontradas em simulação. Enquanto na simulação o ataque foi especialmente efetivo sobre as peças finais (especialmente sobre as últimas 4000), nas avaliações deste trabalho o ataque consegue atrasar já as peças iniciais, gerando um afastamento gradativo das curvas SCCA e CCNCA em relação ao caso ótimo. Pode-se perceber também o efeito devastador do ataque sobre a última peça de cada par. Ao analisar os casos com ataque, percebe-se que a partir do ponto $x = 15700$, tanto em SCCA como CCNCA, o recebimento de peças aproxima-se de nulo. Observou-se que ao entrar na fase final do *download*, os pares solicitam as peças para o maior número de pares disponíveis, ficando mais suscetíveis aos atacantes presentes no enxame.



(a) Pares com *downloads* completados



(b) Número de peças corretas recebidas

Figura 7.3 – Avaliação da Anticorrupção.

Em relação à eficácia do algoritmo implementado, respondendo à questão Q2, ao se comparar as curvas SCCA e CCCA da Figura 7.3(a), pode-se perceber que todos os pares conseguem se recuperar do ataque, em oposição ao caso sem a contramedida implementada. Mas em contraste aos resultados encontrados via

simulação, a contramedida não foi tão eficiente, uma vez que os pares levaram entre 135 e 202 minutos para completar seus *downloads*. Já ao comparar o recebimento de peças na Figura 7.3(b), as 10000 primeiras peças chegam em uma taxa próxima do caso ótimo. A partir deste ponto os atacantes passam a gerar uma maior sobrecarga para a contramedida. Isto acontece porque o algoritmo ainda não gerou evidências suficientes para banir os atacantes.

Por não possuir evidências suficientes para punir os atacantes, os pares honestos começam a baní-los de seus conjuntos tardiamente, conforme demonstrado através da Figura 7.4. No eixo x pode-se ver o número total de pares banidos, enquanto o eixo y representa o tempo de banimento do atacante. Percebe-se que o primeiro atacante somente é banido por volta dos 55 minutos. E de um total de 465 atacantes banidos, a metade deles leva aproximadamente 150 minutos até o banimento, ou seja, em $\frac{3}{4}$ do tempo de vida do enxame apenas metade dos atacantes são detectados. Surpreendentemente, ao computar-se o total de conexões de atacantes estabelecidas, encontrou-se que de 2023 conexões de atacantes estabelecidas, apenas 465 foram detectadas, ou seja, menos de 25% dos atacantes são banidos.

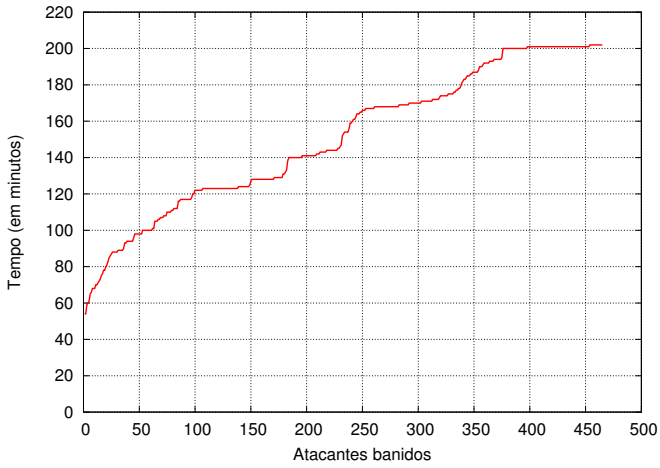


Figura 7.4 – Número de pares atacantes detectados pela Anticorrupção.

A avaliação da sobrecarga do algoritmo, referente à questão Q3, pode ser feita através da análise dos cenários sem ataques na Figura 7.3(a), SCSA e CCSA. Percebe-se que as curvas são bastante similares, quase sobrepostas, indicando não existir sobrecarga na implementação da contramedida. Também em relação às peças, na Figura 7.3(b), percebe-se tendência semelhante. Este resultado também foi encontrado em simulação e é esperado, uma vez que o algoritmo Anticorrupção não entra em ação a menos que peças corrompidas sejam recebidas.

7.5 Avaliação da Mentira em Massa

Como descrito na Seção 3.2, o objetivo do ataque de Mentira de Peças é causar o desequilíbrio na distribuição das peças no enxame. Como pode ser visto através da Figura 7.5, o ataque de mentira sob um cenário sem contramedida ativa atinge tal objetivo. No eixo x cada ponto representa uma das 64 peças, enquanto o y mostra a quantidade de pares que tiveram sucesso em obter cada peça. Para o cenário avaliado, foram mentidas as 16 primeiras peças. Pode-se notar na Figura 7.5 que as peças 1, 3 e 16 foram obtidas por apenas 24 pares, sendo estes os pares que conseguiram completar seus *downloads*, conforme descrito na Seção 7.3. O desequilíbrio na distribuição de peças, aliado com o eclipse do semeador, são a razão do sucesso deste ataque.

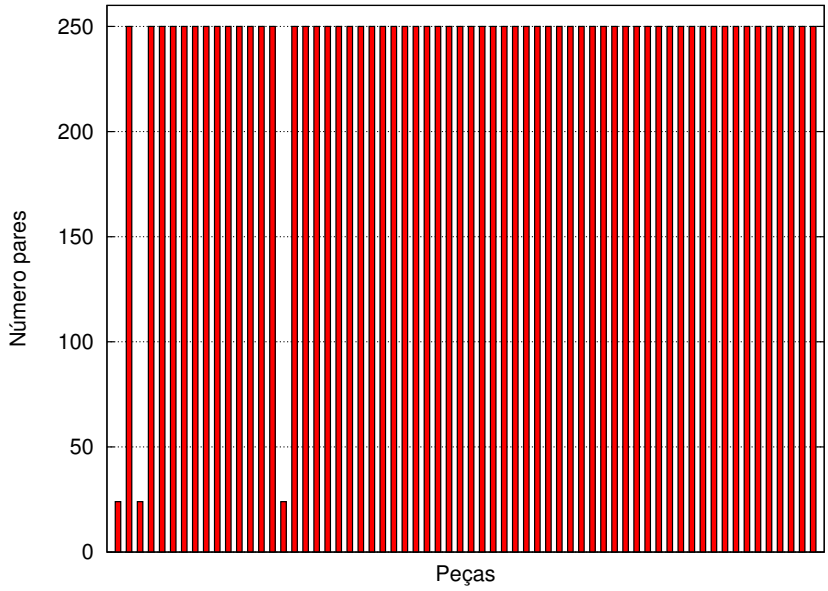


Figura 7.5 – Número de pares que obtiveram peças.

Durante a execução dos cenários de avaliação, foram encontrados casos onde o ataque de mentira não obteve sucesso diante do enxame. Concluímos que a eficácia do ataque está fortemente ligada à possibilidade de eclipse do semeador, tendo maior probilidade de ocorrer na fase de "nascimento" do enxame. Esta conclusão está consonante com os resultados recentemente reportados em [Dhungel et al., 2009].

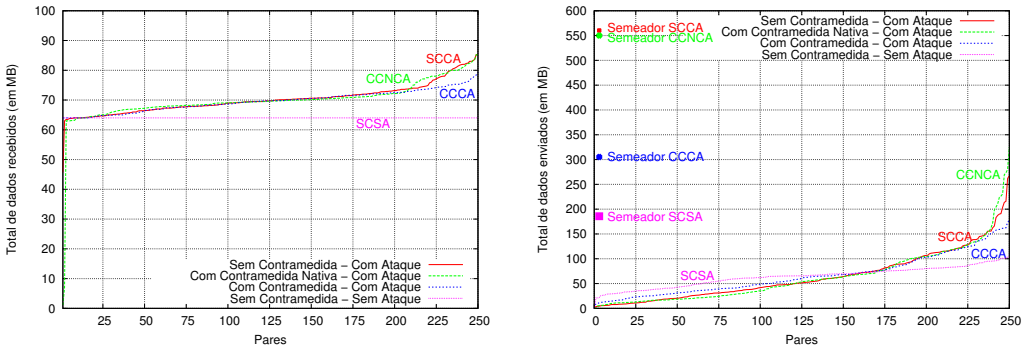
Outro fator que detectamos para o sucesso do ataque consiste na permanência dos pares sugadores no enxame após obterem uma cópia completa do conteúdo.

Nos cenários onde o ataque obteve sucesso, observou-se que os sugadores conectados ao semeador deixaram o enxame logo após completarem seu *download*, pois atingiram sua razão alvo de permanência. Entretanto, alguns poucos pares que permanecem no enxame conectados ao semeador e distribuindo as últimas peças (afetadas pelo desequilíbrio causada pelo ataque) são suficientes para mitigar o impacto do ataque.

7.6 Avaliação da Corrupção de Peças

Para avaliar o impacto do ataque de Corrupção de Peças em relação à sobrecarga de dados gerados pelo enxame, foram computados os dados enviados e recebidos pelos pares em cenários com ataque. Os resultados obtidos são apresentados nos gráficos da Figura 7.6. No eixo x de ambos os gráficos encontram-se os pares, ordenados conforme sua razão de *upload*/*download*. No eixo y, por sua vez, o total de *upload*/*download*. As curvas apresentas no gráfico correspondem aos cenários sobre ataque: SCCA, CCNCA e CCCA. Como curva de referência, utilizou-se o caso ótimo, SCSA.

Na Figura 7.6(a), como esperado, a curva SCSA (representando o cenário ótimo) é uma linha horizontal, refletindo o fato que todos os pares recebem o mesmo volume de dados (igual ao tamanho total do conteúdo). Ao observar as curvas representando os casos de ataque, destaca-se a sobrecarga gerada pelo ataque em praticamente todo o enxame, com exceção de 25 pares. Mas, contrariando nossa expectativa, a contramedida avaliada não gerou um ganho significativo sobre o volume de dados recebidos pelos pares. No entanto, vale destacar que enquanto muitos pares falham (após estagnarem) no caso com ataque e sem contramedida, na curva com contramedida todos os pares foram até o fim em seus *downloads*, carregando todas as peças necessárias.



(a) Sobrecarga de *download* (b) Sobrecarga de *upload*

Figura 7.6 – Avaliação da sobrecarga gerada pela Corrupção de Peças.

Para análise do volume de dados enviados pelos pares, foram registrados os dados do semeador permanente. A Figura 7.6(b) mostra os resultados. Para dintiguir o semeador permanente dos demais pares, seus dados foram traçados como pontos, próximos de $x = 0$. Na comparação do caso ótimo com os casos com ataque, observa-se que para os primeiros 175 pares não existem diferenças significativas em seu volume de dados enviados. Já para os demais 75 pares do enxame, o ataque gera uma sobrecarga de dados a enviar. Já ao comparar os cenários com a contramedida avaliada frente aos casos sem contramedida e com contramedida nativa, percebe-se uma relativa vantagem sob o conjunto dos últimos 20 pares.

No entanto, diferença significativa pode ser percebida ao avaliar o total de *upload* realizado pelo semeador no 4 casos. No caso ótimo, o semeador distribui 185 MB em dados. Em contraste, no caso com ataque e sem contramedida (e, também, quando com contramedida nativa) ele distribui um valor bem mais alto, 560 MB, no intervalo máximo avaliado. Já no caso CCCA, o semeador distribui 305 MB, ou seja, cerca de 65% mais de dados em relação ao caso ótimo, contra mais de 300% nos casos sem a contramedida avaliada.

Capítulo 8

Conclusões

O compartilhamento de arquivos através de redes P2P tem se tornado uma das mais significativas aplicações na Internet. Dentre as tecnologias utilizadas, o BitTorrent vem se destacando como um dos mais populares. A possibilidade exploração de ataques *Denial of Service* (DoS) em BitTorrent já foi descrita em [Konrath et al., 2007] e também já foi identificada em enxames reais [Zip, 2009], causando prejuízos significativos para os participantes.

Em busca de mecanismos de defesa para ataques em BitTorrent, o autor desta monografia participou da elaboração de algoritmos de contramedidas, especificamente para ataques de Mentira de Massa e Corrupção de Peças. Os algoritmos criados foram avaliados através de um modelo em simulação e apresentados em [Barcellos et al., 2008a, Bauermann et al., 2008].

Conforme [Dhungel et al., 2009], muitos agentes desviam do protocolo original do BitTorrent em sua implementação. Assim, ainda que resultados satisfatórios tenham sido encontrados nesta avaliação através de simulação, é necessário avaliar tais resultados em agentes reais, complementando e validando os resultados prévios.

A implementação em agentes reais não se mostrou uma tarefa trivial. Além da magnitude do código a ser modificado, foram necessárias mudanças em lógicas já estabelecidas, como no caso do ataque de Mentira em Massa, ao criar um conjunto grande de conexões. A montagem do ambiente e a avaliação de múltiplos agentes em uma mesma máquina foram desafios enfrentados com a adequação do sistema operacional para suporte à carga criada.

A principal contribuição deste trabalho é a avaliação da eficácia e da eficiência das contramedidas em um ambiente real, porém controlado. Somado a isto, os resultados aqui apresentados contribuem para aproximar o ambiente de simulação com o ambiente real, aumentando assim a confiabilidade em simulações futuras. Além disso, as implementações criadas no trabalho serão disponibilizadas para a comunidade, podendo ser adotadas por agentes reais.

Este trabalho pode ser estendindo de diferentes formas. As avaliações se restrigiram a um conjunto de cenários, a fim de validar os resultados previamente apresentados em simulação. Assim, é possível explorar os ataques e contramedidas em diferentes cenários, estendendo a análise de sensibilidade dos parâmetros. Também pode-se avaliar o comportamento do enxame criando-se cenários maiores, através do uso do PlanetLab.

Embora a contramedida Anticorrupção tenha apresentado sucesso na recuperação do enxame sob ataque, percebe-se que existe margem para melhorias. Desta forma, outra possibilidade seria a exploração de uma possível melhoria desta contramedida, aumentando sua eficiência e diminuindo o tempo de *download* dos pares no enxame.

Bibliografia

[Azu, 2009] (2009). Azureus website. http://azureus.sourceforge.net/.

[Bit, 2009a] (2009a). Bittornado. http://www.bittornado.com.

[Bit, 2009b] (2009b). Bittorrent client. http://en.wikipedia.org/wiki/Comparison_of_BitTorrent_software.

[Bit, 2009c] (2009c). Bittorrent protocol specification v1.0. http://wiki.theory.org/BitTorrentSpecification.

[Tra, 2009] (2009). Transmission. http://www.transmissionbt.com.

[UTo, 2009] (2009). utorrent website. http://www.utorrent.com/.

[Zip, 2009] (2009). ZipTorrent Pollutes and Slows Down Popular Torrents. http://torrentfreak.com/ziptorrent-pollutes-and-slows-down-popular-torrents/.

[Barbera et al., 2005] Barbera, M., Lombardo, A., Schembra, G., e Tribastone, M. (2005). A markov model of a freerider in a bittorrent P2P network. Em *IEEE Global Telecommunications Conference (GLOBECOM '05)*, volume 2, páginas 985–989, St. Louis, MO, USA.

[Barcellos et al., 2008a] Barcellos, M. P., Bauermann, D., Sant'anna, H., Lehmann, M., e Mansilha, R. (2008a). Protecting bittorrent: design and evaluation of effective countermeasures against dos attacks. Em *27th International Symposium on Reliable Distributed Systems (IEEE SRDS 2008)*.

[Barcellos et al., 2006] Barcellos, M. P., Facchini, G., Muhammad, H. H., Bedin, G. B., e Luft, P. (2006). Bridging the gap between simulation and experimental evaluation in computer networks. Em *Simulation Symposium, 2006. 39th Annual*, páginas 8 pp.+.

[Barcellos et al., 2008b] Barcellos, M. P., Mansilha, R. B., e Brasileiro, F. V. (2008b). Torrentlab: investigating bittorrent through simulation and live experiments. Em *IEEE Symposium on Computers and Communications (ISCC'08)*, páginas 1–6.

[Bauermann et al., 2008] Bauermann, D., Lehmann, M., Mansilha, R., e Barcellos, M. P. (2008). Protegendo bittorrent: projeto e avaliação de contra-medidas eficazes para ataques dos. Em *VIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg 2008)*, páginas 215–228.

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

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

[Cohen, 2009a] Cohen, B. (2009a). Bittorrent. http://www.bittorrent.com/.

[Cohen, 2009b] Cohen, B. (2009b). The bittorrent protocol specification. http://www.bittorrent.org/beps/bep_0003.html.

[Dhungel et al., 2009] Dhungel, P., Heiz, X., Wu, D., e Ross, K. W. (2009). The seed attack: Can bittorrent be nipped in the bud? Technical report.

[Dhungel et al., 2008] Dhungel, P., Wu, D., Schonhorst, B., e Ross, K. W. (2008). A measurement study of attacks on bittorrent leechers. Em *International Workshop on Peer-to-Peer Systems (IPTPS)*.

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

[Dubuis, 2009] Dubuis, B. (2009). Java bittorrent api. http://sourceforge.net/projects/bitext/.

[Greer, 2009] Greer, G. (2009). Limiting download speed. http://osdir.com/ml/network.bit-torrent.general/2004-04/msg00142.html.

[Hogan, 2009] Hogan, T. (2009). Bnbt easytracker. http://bnbteasytracker.sourceforge.net/.

[Izal et al., 2004] Izal, M., Urvoy-Keller, G., Biersack, E., Felber, P., Al-Hamra, A., e Garcés-Erice, L. (2004). Dissecting bittorrent: Five months in a torrent's lifetime. Em Chadi, editor, *5th International Workshop, PAM 2004*, volume 3015 / 2004, páginas 1–11. Springer Berlin / Heidelberg.

[Konrath, 2007] Konrath, M. A. (2007). Estudo das vulnerabilidades da arquitetura bittorrent, ataques e contramedidas possíveis. Dissertação de mestrado, São Leopoldo - RS - Brasil.

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

[Mansilha et al., 2008] Mansilha, R. B., Barcellos, M. P., e Brasileiro, F. V. (2008). Torrentlab: Um ambiente para avaliação do protocolo bittorrent. Em *XXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2008)*, páginas 357–370.

[Mansilha et al., 2007] Mansilha, R. B., Konrath, M. A., e Barcellos, M. P. (2007). Corrupção, mentiras e isolamento: avaliação de impacto de ataques a bittorrent. Em *VII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSEG 2007)*.

[Qiu and Srikant, 2004] Qiu, D. e Srikant, R. (2004). Modeling and performance analysis of BitTorrent-like peer-to-peer networks. *SIGCOMM Comput. Commun. Rev.*, 34(4):367–378.

[Sadok et al., 2005] Sadok, D., Kamienski, C. K., Souto, E., Rocha, J. a., Domingues, M., e Callado, A. (2005). Colaboração na internet e a tecnologia peer-to-peer. Em *XXV Congresso da Sociedade Brasileira de Computação (SBC 2005)*, páginas 1407–1454.

[Schmitt et al., 2008] Schmitt, C., Barcellos, M. P., e Mansilha, R. B. (2008). Um estudo experimental sobre ataques ao sistema de compartilhamento p2p bittorrent. Em *XXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2008)*, páginas 581–594.

[Singh et al., 2006] Singh, A., Ngan, T.-W., Druschel, P., e Wallach, D. S. (2006). Eclipse attacks on overlay networks: Threats and defenses. Em *25th Conference on Computer Communications (INFOCOM 2006)*. IEEE.

DoS *Denial of Service*

IP *Internet Protocol*

LRF *Local Rarest First*

NAT *Network Address Translation*

P2P *Peer-to-Peer*

RFC *Request for Comments*

SHA *Secure Hash Algorithm*

TCP *Transmission Control Protocol*

TFT *Tit-for-Tat*