PEDRO MANOEL FABIANO ALVES EVANGELISTA

EBITSIM: SIMULADOR DE BITTORRENT UTILIZANDO O
ARCABOUÇO OMNET++

São Paulo

2012

PEDRO MANOEL FABIANO ALVES EVANGELISTA

EBITSIM: SIMULADOR DE BITTORRENT UTILIZANDO O
ARCABOUÇO OMNET++

Dissertação apresentada à Escola
Politécnica da Universidade de São Paulo
para obtenção do título de Mestre em
Engenharia

São Paulo

2012

PEDRO MANOEL FABIANO ALVES EVANGELISTA

EBITSIM: SIMULADOR DE BITTORRENT UTILIZANDO O
ARCABOUÇO OMNET++

Dissertação apresentada à Escola
Politécnica da Universidade de São Paulo
para obtenção do título de Mestre em
Engenharia

Área de Concentração: Sistemas Digitais

Orientadora: Prof^ª Dr^a Tereza Cristina Melo
de Brito Carvalho

São Paulo

2012

FICHA CATALOGRÁFICA

Evangelista, Pedro Manoel Fabiano Alves
EbitSim: simulador de BitTorrent utilizando o arcabouço
OMNeT++ / P.M.F.A. Evangelista. -- São Paulo, 2012.
106 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade
de São Paulo. Departamento de Engenharia de Computação e
Sistemas Digitais.

1. Simulação 2. Redes de computadores 3. Internet 4. Proto-
colos de comunicação I. Universidade de São Paulo. Escola
Politécnica. Departamento de Engenharia de Computação e
Sistemas Digitais II. t.

AGRADECIMENTOS

À professora Tereza Cristina Melo de Brito Carvalho, pela orientação e pelo constante estímulo transmitido durante todo o trabalho.

Aos meus amigos e familiares, por todo apoio e suporte dados durante o meu mestrado.

Aos meus amigos Charles Miers e Rony Sakuragui, pelo apoio essencial e pelas idéias valiosas para a evolução desta dissertação.

Ao meu amigo Walter Goya, pelas idéias e discussões que colaboraram grandemente para a conclusão deste trabalho.

À Ericsson Telecomunicações do Brasil e Ericsson Research da Suécia, pelo suporte financeiro e cooperação conjunta nos projetos de pesquisa do LARC.

RESUMO

O protocolo *BitTorrent* é uma das aplicações P2P mais bem sucedidas da *Internet*, sendo largamente estudada pela comunidade de pesquisa. Contudo, o estudo da dinâmica de uma rede *BitTorrent* de larga escala apresenta diversos desafios, tais como a dificuldade em realizar capturas da rede ou montar experimentos para medição. O método utilizado para superar estes desafios é a simulação, porém não há uma ferramenta adequada disponível para a comunidade de pesquisa. Por conta disso, a maioria dos trabalhos que utilizam simulação desenvolvem os seus próprios simuladores, resultando em trabalhos que não podem ser repetidos ou verificados. Neste trabalho, é apresentado o simulador de *BitTorrent* *EbitSim*, que permite a alteração dos mecanismos utilizados, a configuração dos parâmetros do sistema e a definição da topologia utilizada. O simulador foi desenvolvido utilizando o arcabouço OMNeT++, que fornece um conjunto de ferramentas que facilitam a configuração de variados cenários e dos parâmetros do modelo. Além disso, o arcabouço INET foi utilizado para modelar as camadas inferiores de rede. O desenvolvimento do modelo do *BitTorrent* foi baseado na especificação oficial, e contou com o auxílio de trabalhos relacionados e discussões com desenvolvedores de clientes *BitTorrent*. O *EbitSim* foi validado por meio de comparações com resultados obtidos a partir de uma implementação real de uma rede *BitTorrent*, realizada em um ambiente de testes controlado. Foi demonstrado que o *EbitSim* apresenta resultados compatíveis com uma rede *BitTorrent* real.

Palavras-chave: *Peer-to-Peer*. *BitTorrent*. Simulação de Redes de Computadores.

ABSTRACT

The *BitTorrent* protocol is one of the most successful P2P applications, being largely studied by the research community. Nevertheless, studying the dynamics of a large *BitTorrent* network presents several challenges, such as difficulty in acquiring network traces or building measurement experiments. The simulation method is capable of overcoming these challenges, but there is not an adequate simulation tool available for the research community. This thesis presents the *EbitSim BitTorrent* Simulator, which is capable of modifying the utilized mechanisms, configuring the system's parameters and defining the topology used in the simulations. The simulator was developed using the OMNeT++ framework, which provides a set of tools that facilitates the configuration of diversified scenarios and the parameters of the model. In addition, the INET framework was utilized to accurately model the lower network levels. We developed the *BitTorrent* model based on the official specification, with the aid of related works and discussions with developers of *BitTorrent* client programs. The *EbitSim* Simulator was validated by performing comparisons with results obtained from a real implementation of a *BitTorrent* network, deployed in a controlled testbed. We show that the *EbitSim* Simulator generates results compatible with a real *BitTorrent* network.

Keywords: *Peer-to-Peer. BitTorrent. Computer Network Simulation.*

LISTA DE ABREVIATURAS E SIGLAS

API	<i>Application Programming Interface</i>
BEP	<i>BitTorrent Enhancement Proposal</i>
FTP	<i>File Transfer Protocol</i>
HTTP	<i>Hypertext Transfer Protocol</i>
IDE	<i>Integrated Development Environment</i>
IP	<i>Internet Protocol</i>
NED	<i>Network Description</i>
NIC	<i>Network Interface Card</i>
OO	Orientação a Objetos
P2P	<i>Peer-to-Peer</i>
RAM	<i>Random Access Memory</i>
RPC	<i>Remote Procedure Call</i>
TCP	<i>Transmission Control Protocol</i>

LISTA DE FIGURAS

Figura 1 – Exemplo dos modelos Cliente-servidor e P2P	14
Figura 2 – Volume de Dados Compartilhados utilizando <i>BitTorrent</i> e eDonkey em 2006	15
Figura 3 – Composição do Tráfego em Redes de Acesso Fixo nos EUA.....	16
Figura 4 – Arquitetura do <i>BitTorrent</i>	18
Figura 5 – Exemplo de divisão do conteúdo em pedaços e blocos	21
Figura 6 – Encadeamento de requisições.....	21
Figura 7 – Troca de mensagens entre dois <i>Peers</i> do <i>BitTorrent</i>	26
Figura 8 – Máquina de Estados de Conexão	27
Figura 9 – Máquina de Estados de Download	28
Figura 10 – Máquina de Estados de Upload	29
Figura 11 – Uso de simuladores para análise de sistemas P2P	38
Figura 12 – Elementos básicos do OMNeT++	58
Figura 13 – Hierarquia de módulos no OMNeT++	59
Figura 14 – Modelo de um switch ethernet do arcabouço INET	62
Figura 15 – Modelo de um roteador do arcabouço INET	62
Figura 16 – Modelo de um nó de rede do arcabouço INET	63
Figura 17 – Principais elementos do EbitSim.....	65
Figura 18 – Organização do módulo BitTorrentApp.....	66
Figura 19 – Processo de entrada do <i>Peer</i> no swarm.....	67
Figura 20 – Processamento de mensagens com atraso simples para uma conexão.....	70
Figura 21 – Processamento de mensagens com atraso simples.....	70
Figura 22 – Esquema do processamento de mensagens pelo Cliente <i>BitTorrent</i>	71
Figura 23 – Exemplo do funcionamento do módulo Choker	74
Figura 24 – Exemplo dos sinais de percentagem de aquisição capturados no nível mais alto da hierarquia.....	78
Figura 25 – Exemplo do sinal de emissão dos sinais de percentagem capturado no nível mais alto da hierarquia	78
Figura 26 – Exemplo do cruzamento dos sinais de percentagem com o sinal de emissão.....	79
Figura 27 – Cenário utilizado nos experimentos de Szydlowski	82

Figura 28 – Notação para especificação dos experimentos	84
Figura 29 – Comparação da quantidade de mensagens enviadas em todos os experimentos.....	86
Figura 30 – Função densidade de probabilidade do tempo de download para o cenário CL-35-5 e a simulação ES-35-5	87
Figura 31 – Função densidade de probabilidade do tempo de download para o cenário CL-92-5 e a simulação ES-92-5	88
Figura 32 – Variação do tempo de simulação de acordo com o tamanho do arquivo simulado	90
Figura 33 – Variação do tempo de simulação de acordo com o número de conexões	91
Figura 34 – Encadeamento do experimento de taxa de upload com 10 repetições .	92

LISTA DE TABELAS

Tabela 1 –	Informações no arquivo <i>.torrent</i> do sistema operacional Ubuntu 12.04 32 bits.	20
Tabela 2 –	Estado inicial de uma conexão PeerWire	23
Tabela 3 –	Agrupamento das mensagens <i>PeerWire</i> , do ponto de vista do Cliente	25
Tabela 4 –	Classificação dos estados da conexão	27
Tabela 5 –	Comparação sobre o nível de detalhamento da especificação original e a adotada neste trabalho	32
Tabela 6 –	Verificação dos requisitos para simulador de <i>BitTorrent</i> nos simuladores disponíveis.....	47
Tabela 7 –	Comparação entre as arquiteturas	51
Tabela 8 –	Comparação entre a usabilidade	53
Tabela 9 –	Comparação entre as capacidades estatísticas	54
Tabela 10 –	Comparação entre a modelagem da rede subjacente.....	55
Tabela 11 –	Comparação entre a estabilidade	55
Tabela 12 –	Parâmetros do experimento Aloha do OMNeT++.....	60
Tabela 13 –	Parâmetros do módulo “Gerente de Swarms”	68
Tabela 14 –	Parâmetros do módulo “Cliente <i>BitTorrent</i> ”	71
Tabela 15 –	Parâmetros do módulo Choker	75
Tabela 16 –	Parâmetros do módulo “Gerente de Conteúdo”.....	76
Tabela 17 –	Parâmetros do cenário de validação	82
Tabela 18 –	Experimentos realizados.....	85
Tabela 19 –	Resultados para o tempo médio de simulação do experimento da taxa de upload	89
Tabela 20 –	Resultados para o tempo médio de simulação do experimento do número de conexões	90
Tabela 21 –	Razão entre tempo simulado médio e o tempo de simulação médio para o experimento da taxa de upload	92
Tabela 22 –	Razão entre tempo simulado médio e o tempo de simulação médio para o experimento do número de conexões	93

SUMÁRIO

1 INTRODUÇÃO	11
1.1 MOTIVAÇÃO E DESCRIÇÃO DO PROBLEMA	12
1.2 OBJETIVO	13
1.3 ORGANIZAÇÃO DO TRABALHO.....	13
2 SISTEMAS P2P E O BITTORRENT.....	14
2.1 SISTEMAS P2P	15
2.2 O SISTEMA BITTORRENT	18
2.3 ARQUITETURA DO BITTORRENT.....	18
2.3.1 Atores do sistema BitTorrent.....	19
2.3.2 Mecanismos de Controle.....	22
2.3.3 O Protocolo PeerWire	24
2.3.4 O algoritmo de Choking.....	30
2.4 COMPARAÇÃO COM ESPECIFICAÇÃO ORIGINAL	32
2.5 CONSIDERAÇÕES PARCIAIS.....	33
3 TRABALHOS RELACIONADOS E REQUISITOS DO SIMULADOR	35
3.1 TRABALHOS RELACIONADOS.....	35
3.2 O USO DA SIMULAÇÃO NO ESTUDO DO BITTORRENT	38
3.3 REQUISITOS PARA UM SIMULADOR DE BITTORRENT	41
3.4 SIMULADORES DE BITTORRENT DISPONÍVEIS.....	43
3.4.1 Análise do simulador GPS	44
3.4.2 Análise do módulo BitTorrent para PeerSim.....	45
3.4.3 Análise do simulador OctoSim	46
3.4.4 Análise do simulador BIT-SIM.....	47
3.4.5 Considerações sobre a análise dos requisitos	48

3.5 CONSIDERAÇÕES PARCIAIS.....	49
4 ARCABOUÇOS DE SIMULAÇÃO E O OMNET++	50
4.1 ARCABOUÇOS DISPONÍVEIS	50
4.2 ESCOLHA DO ARCABOUÇO	52
4.2.1 Arquitetura	52
4.2.2 Usabilidade	54
4.2.3 Estatísticas	55
4.2.4 Modelagem da rede subjacente	56
4.2.5 Estabilidade	56
4.2.6 Resultado da comparação.....	57
4.3 O ARCABOUÇO DE SIMULAÇÃO OMNET++.....	58
4.3.1 Conceitos de modelagem.....	59
4.3.2 Execução de experimentos	61
4.4 ARCABOUÇO INET PARA OMNET++.....	62
4.5 CONSIDERAÇÕES PARCIAIS.....	65
5 O SIMULADOR DE <i>BITTORRENT EBITSIM</i>	66
5.1 ELEMENTOS DO SIMULADOR	66
5.2 MÓDULO GERENTE DE <i>SWARM</i>	68
5.2.1 Parâmetros.....	69
5.2.2 Sinais emitidos	70
5.3 MÓDULO CLIENTE <i>BITTORRENT</i>	70
5.3.1 Parâmetros.....	72
5.3.2 Sinais emitidos	73
5.4 MÓDULO <i>CHOKER</i>	74
5.4.1 Parâmetros.....	75
5.4.2 Sinais emitidos	76

5.5 MÓDULO GERENTE DE CONTEÚDO	76
5.5.1 Parâmetros.....	77
5.5.2 Sinais emitidos	77
5.6 CONSIDERAÇÕES PARCIAIS.....	80
6 VALIDAÇÃO DO EBITSIM.....	82
6.1 MÉTODO PARA VALIDAÇÃO.....	82
6.1.1 Cenários do experimento de Szydlowski.....	82
6.1.2 Métricas de comparação definidos por Szydlowski	85
6.2 REALIZAÇÃO DA VALIDAÇÃO DO <i>EBITSIM</i>	86
6.2.1 Comparação dos perfis de mensagens PeerWire.....	86
6.2.2 Comparação dos tempos de aquisição do conteúdo	88
6.3 DADOS SOBRE A EXECUÇÃO DAS SIMULAÇÕES	90
6.3.1 Relação entre o tempo de simulação e os parâmetros da medição	90
6.3.2 Relação entre o tempo simulado e os parâmetros da medição	93
6.4 CONSIDERAÇÕES PARCIAIS.....	94
7 CONSIDERAÇÕES FINAIS.....	96
7.1 CONTRIBUIÇÕES	96
7.2 TRABALHOS FUTUROS.....	98
7.3 PUBLICAÇÕES	99

1 INTRODUÇÃO

O surgimento dos sistemas *Peer-to-Peer* (P2P) no ano 2000 iniciou uma mudança do paradigma da *Internet*, onde usuários que eram apenas consumidores de conteúdo passaram a ser também produtores (não necessariamente de sua autoria) (ORAM, 2001). Neste sistema, o problema do uso de recursos do servidor é mitigado, uma vez que a capacidade de recursos é distribuída entre os usuários (*Peers*) da rede. Além disso, o sistema continua disponível enquanto houver pelo menos um *Peer* presente (XIAO, B.; CAO, J.; SHA, 2004).

O *BitTorrent* é um sistema P2P que foi criado com o intuito de diminuir os custos com largura de banda de empresas que disponibilizavam arquivos de tamanho grande (centenas de megabytes) para seus usuários¹. O sucesso do *BitTorrent* como aplicação para distribuição de conteúdo o tornou o protocolo P2P mais bem sucedido da atualidade, resultando no crescimento contínuo do tráfego *BitTorrent* na *Internet* (IPOQUE, 2006, 2007, 2009). Estudos atuais (IPOQUE, 2009, 2011; SANDVINE, 2011) mostram que o tráfego da Internet relacionado ao entretenimento em tempo real (gerados por serviços como *YouTube* e *PPLive*) ultrapassaram o tráfego *BitTorrent* em quantidade. Porém o tráfego P2P ainda é responsável por uma parte considerável do tráfego total da *Internet*.

Essa popularidade fez com que o *BitTorrent* ganhasse importância dentro das comunidades de pesquisa. Um dos primeiros trabalhos de análise de desempenho do *BitTorrent* apresentou uma análise abrangente do funcionamento dos mecanismos do *BitTorrent* (IZAL *et al.*, 2004). Além de trabalhos que verificam o funcionamento do *BitTorrent*, há também aqueles que demonstram a aplicação do *BitTorrent* em outras áreas além daquela para qual foi desenvolvido, como, por exemplo, *streaming* de vídeos (BAKKER *et al.*, 2010; PARVEZ *et al.*, 2012; SHAH; PARIS, 2007), distribuição de arquivos em redes locais (AGUILAR *et al.*, 2007; THOMPSON; BROOKS; GREER, 2008) e distribuição de dados em redes veiculares sem fio (SHRESTHA *et al.*, 2008).

¹ Website do *BitTorrent* em 2004:
<http://web.archive.org/web/20040919033827/BitTorrent.com/introduction.html>

1.1 MOTIVAÇÃO E DESCRIÇÃO DO PROBLEMA

Os diversos trabalhos desenvolvidos pela comunidade acadêmica utilizam diferentes métodos para efetuar a análise de desempenho do *BitTorrent*, como medição do comportamento de *Peers* em uma rede real, simulações e modelos matemáticos (XIA; MUPPALA, 2010). Essas abordagens podem ser abstraídas respectivamente nos métodos de: medição, simulação e modelagem analítica (JAIN, 1991). Dentre esses métodos, a simulação apresenta um conjunto de vantagens que justificam a sua escolha para a sua utilização no estudo do *BitTorrent*, tais como permitir a análise de versões modificadas dos mecanismos e efetuar o estudo do desempenho de acordo com a variação dos valores dos parâmetros do sistema.

Porém, os simuladores de *BitTorrent* disponíveis, a serem discutidos na seção 3.4, carecem de ferramentas apropriadas para a coleta de estatísticas e possuem difícil configuração dos parâmetros e dos cenários. Isso leva a grande maioria dos trabalhos a desenvolverem os seus próprios simuladores para obter resultados (MONTRESOR; JELASITY, 2009; YANG; ABU-GHAZALEH, 2005). Esta abordagem apresenta características não desejáveis (NAICKEN *et al.*, 2007):

- Baixa credibilidade, já que o funcionamento do simulador não é verificado pela comunidade de pesquisa.
- Baixa reprodutividade, pois nem todos os detalhes da simulação são disponibilizados.
- Maior tempo de desenvolvimento, tendo em vista que a tomada de resultados só pode ser iniciada depois que o simulador foi completado e testado.
- Baixa precisão, já que o simulador pode conter diversas falhas de *software* que não foram identificados pela equipe de desenvolvimento. Como o simulador não pode ser verificado por outros pesquisadores, os resultados não podem ser verificados.

O estado atual dos simuladores de *BitTorrent* mostra que é necessária a existência de uma ferramenta que preencha os requisitos de facilidade de configuração, personalização dos mecanismos e coleta de estatísticas. Esta ferramenta permitiria à comunidade de pesquisa produzir trabalhos que possam ser reproduzidos e

verificados pelos pares. A motivação deste trabalho é, portanto, o desenvolvimento de um simulador de *BitTorrent* capaz de atender estas características.

1.2 OBJETIVO

Este trabalho foi realizado com o objetivo de desenvolver um simulador de *BitTorrent* que pode ser utilizado como uma ferramenta de pesquisa capaz de estudar novas características além do comportamento normal do sistema *BitTorrent*. O simulador desenvolvido deve permitir a alteração dos mecanismos do *BitTorrent* e a configuração dos parâmetros do sistema, possibilitando o estudo de cenários diversos.

1.3 ORGANIZAÇÃO DO TRABALHO

Esta dissertação está organizada em 7 capítulos, sendo esta introdução o primeiro capítulo. O capítulo 2 apresenta o conceito de redes P2P e descreve o sistema *BitTorrent* em detalhes. O capítulo 3 segue com a apresentação dos trabalhos relacionados, cujos resultados são obtidos a partir de simulações, e define os requisitos para o desenvolvimento de um simulador de *BitTorrent*. A partir desta base, o capítulo 4 apresenta os arcabouços de simulação disponíveis e a justificativa para a escolha do arcabouço utilizado. No capítulo 5, o simulador de *BitTorrent* *EbitSim* é apresentado em detalhes. O capítulo 6, por sua vez, traz os resultados de validação do mesmo. Por fim, o capítulo 7 apresenta as considerações finais sobre este trabalho e discute possíveis trabalhos futuros.

2 SISTEMAS P2P E O BITTORRENT

O objetivo deste capítulo é descrever os elementos do sistema *BitTorrent* e como eles interagem para prover o serviço de distribuição de arquivos.

Primeiramente, serão apresentadas a definição de sistemas P2P e a sua significância no tráfego da Internet. A seguir, o sistema *BitTorrent*, que é um sistema P2P de segunda geração focado no compartilhamento de arquivos (ILIE *et al.*, 2004), será explicado em detalhes.

As descrições oferecidas pela especificação oficial não são suficientes para a compreensão do sistema *BitTorrent* pois não descreve os aspectos dinâmicos dos mecanismos e a interação entre os *Peers* de maneira adequada. Portanto, a descrição dos elementos e mecanismos do *BitTorrent* apresentada neste capítulo foi obtida após uma investigação aprofundada utilizando diversas fontes, além da sua especificação oficial (COHEN, 2008), relacionadas abaixo:

- Especificação não oficial elaborada por usuários e desenvolvedores de clientes *BitTorrent* (BITTORRENT SPECIFICATION, 2006),
- Trabalhos relacionados, como (INRIA, 2005; KONRATH; BARCELLOS; MANSILHA, 2007),
- Troca de informações com desenvolvedores de programas *BitTorrent* no canal IRC *#BitTorrent* encontrado em *irc.freenode.net*,
- Código fonte do Cliente de *BitTorrent* oficial versão 4.0.0 (BITTORRENT MAINLINE, 2006).

No contexto dessa dissertação, esta investigação foi necessária para que o autor tivesse a compreensão necessária do sistema *BitTorrent* para realizar o desenvolvimento do simulador de maneira detalhada, que será descrito no capítulo 5.

2.1 SISTEMAS P2P

No contexto de redes de computadores, um sistema P2P é definido como uma rede onde a maior parte da comunicação ocorre entre os usuários finais, denominados *Peers*, que contribuem com seus arquivos, capacidade de processamento ou outros recursos, para um projeto compartilhado entre eles (ORAM, 2001). Os *Peers* podem ser diferentes do ponto de vista de capacidade de recursos e desempenho, mas são iguais do ponto de vista funcional dentro do sistema, ou seja, uma rede P2P é descentralizada e não existem *Peers* privilegiados ou com papel de controladores do sistema. Na prática, muitas aplicações P2P valem-se de soluções híbridas para resolverem os seus problemas. Por exemplo, em um sistema de mensagens instantâneas, os usuários podem se comunicar diretamente uns com os outros, uma vez que os seus endereços são conhecidos. Porém, para obter estes endereços, um usuário precisa se comunicar com um servidor central que mantém o estado de todos os usuários do sistema (ORAM, 2001).

Li (LI, J., 2006) afirma que uma característica única dos sistemas P2P é que cada vez que um *Peer* entra no sistema, ele traz consigo capacidade de prover recursos, e, conseqüentemente, aumenta a capacidade do sistema como um todo. Isso é um contraste direto ao modelo cliente-servidor, onde toda a capacidade do sistema está localizada em um único ponto, e é mantida pelo provedor do serviço. No exemplo apresentado na Figura 1, o modelo cliente-servidor é comparado ao modelo P2P para distribuição de um conteúdo qualquer.

Figura 1 – Exemplo dos modelos Cliente-servidor e P2P

No modelo cliente-servidor à esquerda da Figura 1, o servidor provê conteúdo para diversos usuários através da *Internet*. Neste caso, o tráfego entre um usuário e o servidor deve atravessar diversos níveis da rede, passando pela *Internet* até as redes de borda. Este tráfego repete-se para cada usuário interessado em adquirir o mesmo conteúdo, causando uma sobrecarga da rede e limitando a escalabilidade do sistema, uma vez que o servidor possui restrições de banda e recursos.

Por outro lado, no modelo P2P, o conteúdo não reside em um local centralizado do sistema, mas sim é compartilhado entre todos os *Peers* através da *Internet*. Desta maneira, como o conteúdo não se encontra em um só local, a questão de múltiplos acessos simultâneos para a rede P2P é mitigada quando comparada ao servidor. Além do mais, a rede não é um gargalo do sistema, pois o tráfego é distribuído pela rede, com cada *Peer* funcionando tanto como servidor quanto como cliente.

Como no modelo P2P a disponibilidade de serviço não depende de um único servidor, esse sistema tornou-se muito atrativo para o compartilhamento de arquivos de tamanho grande, como vídeos, músicas e *software*. Esta popularidade pôde ser observada em um estudo realizado em (IPOQUE, 2006), que classificou o tipo de tráfego P2P na Europa, conforme apresentado na Figura 2. Esta figura mostra o volume de dados P2P dividido por categorias, considerando os protocolos *BitTorrent* e *eDonkey*, que juntos correspondem a 95% do total de tráfego P2P.

Figura 2 – Volume de Dados Compartilhados utilizando *BitTorrent* e *eDonkey* em 2006

Pode-se observar na Figura 2 que os vídeos são responsáveis por 70% do tráfego P2P, o que é uma parcela bastante considerável do tráfego total. Além do mais, o estudo mostra que os arquivos de vídeo e *software* são grandes, com o tamanho

variando de 422MB a 1500MB na rede *BitTorrent* e de 56MB a 690MB na rede *eDonkey*. Esses resultados corroboram a afirmativa de que o sucesso dos protocolos P2P se deve à demanda dos usuários por conteúdos multimídia, e que grande parte do conteúdo trafegado na *Internet* é do tipo vídeo (POUWELSE, J. *et al.*, 2008).

Apesar de estudos mais recentes mostrarem que o volume de tráfego P2P na *Internet* tem sofrido declínio, e que o tráfego de entretenimento em tempo real (*streaming* de áudio e vídeo) está aumentando, o tráfego P2P ainda tem presença significativa no tráfego gerado pelos usuários, tanto para compartilhamento de conteúdo quanto aplicações de vídeo, como *PPStream* e *PPLive* (SANDVINE, 2010, 2011). A Figura 3 mostra os resultados de um estudo de classificação do tráfego nos Estados Unidos para os anos de 2009, 2010 e 2011 (SANDVINE, 2011):

Figura 3 – Composição do Tráfego em Redes de Acesso Fixo nos EUA

Na Figura 3 é possível verificar que o tráfego P2P é menor que o entretenimento em tempo real (*streaming* de áudio e vídeo), e é possível observar que esse tipo de tráfego aumentou nos últimos 3 anos. A explicação para este fenômeno é o

surgimento de aplicações de *streaming* de vídeo, como o *YouTube* e o *NetFlix*, que oferecem vídeo na maioria das vezes gratuitamente, sem intervalos comerciais e sob demanda (POUWELSE, J. *et al.*, 2008).

2.2 O SISTEMA BITTORRENT

O sistema *BitTorrent*, criado por Bram Cohen (COHEN, 2003), foi desenvolvido como uma solução para o comportamento egoísta dos *Peers* da rede. Nos primeiros sistemas P2P, os *Peers* não tinham nenhum incentivo para compartilhar os seus recursos com a rede, como é o caso do *eDonkey*, *Kazaa* e *Gnutella*. Os *Peers* que somente adquirem conteúdo e retornam muito pouco para a rede são chamados de *free-riders*, uma vez que utilizam os recursos da rede P2P “de graça” (JIAN; MACKIE-MASON, 2008). Portanto, o desempenho dessas redes fica limitado ao comportamento de alguns poucos *Peers* altruístas, que são os responsáveis pela maior parte do compartilhamento e tráfego da rede (LEIBOWITZ; RIPEANU; WIERZBICKI, 2003; POUWELSE, J. *et al.*, 2008).

O *BitTorrent*, por outro lado, tenta alcançar o uso justo dos recursos da rede valendo-se de um mecanismo de “dar para receber” ou *tit-for-tat*, que consiste em um *Peer* fornecer mais conteúdo para aqueles que dão mais conteúdo de volta. Desta forma, os *Peers* altruístas atingem maiores taxas de *download*, enquanto o comportamento egoístico é punido com taxas menores. Esse mecanismo é a chave para a escalabilidade e robustez do sistema, uma vez que os *Peers* são incentivados a compartilhar os seus recursos com a rede, promovendo o aumento a capacidade do sistema como um todo.

2.3 ARQUITETURA DO BITTORRENT

A arquitetura do *BitTorrent* foi definida pela primeira vez em (COHEN, 2003), e depois oficializada por meio de um sistema mantido pela comunidade de desenvolvimento, chamado *BitTorrent Enhancement Proposals* (BEPs)². Neste

² Índice de BEPs http://www.BitTorrent.org/beps/bep_0000.html

sistema, as especificações do *BitTorrent* são mantidas e propostas de aprimoramento são discutidas e adicionadas à especificação original, quando aceitas pela comunidade.

Existem três BEPs principais:

- **BEP 0001:** Define o conceito de BEPs, os seus tipos e o fluxo de trabalho ao submeter uma nova proposta para o sistema.
- **BEP 0002:** Define um modelo para a criação de novas BEPs, com normas de escrita e diretrizes gerais sobre o que deve estar presente na proposta.
- **BEP 0003:** Apresenta a especificação oficial do sistema *BitTorrent*.

A BEP 0003 é a mais importante para o estudo do *BitTorrent*, pois define os **atores**, **protocolo de comunicação** e **mecanismos de controle**. Estes elementos serão discutidos em detalhes no restante desta seção.

2.3.1 Atores do sistema *BitTorrent*

O *BitTorrent* é composto por atores que interagem entre si para realizar o serviço de compartilhamento e distribuição de arquivos. Os atores básicos que compõem o sistema *BitTorrent* são: o *Peer*, o *Tracker* e o Repositório de *Torrents*. Esses elementos relacionam-se como mostrado na Figura 4, a seguir.

Figura 4 – Arquitetura do *BitTorrent*

Os *Peers* são os usuários que possuem uma aplicação *BitTorrent* instalada em suas máquinas e que desejam compartilhar um determinado arquivo. O arquivo sendo compartilhado é subdividido em pedaços, e um *Peer* pode ser um *Seeder*, quando possui 100% dos pedaços, ou um *Leecher*, quando ainda não possui o conteúdo completo. Para efetuar o compartilhamento do arquivo, um *Peer* se conecta a outros *Peers* interessados no mesmo conteúdo. Para evitar ambigüidades nas explicações que envolvam a interação entre dois *Peers*, fica definido que o *Peer* em foco na discussão será denominado Cliente, enquanto que o *Peer* conectado continuará a ser referenciado simplesmente por *Peer*, como mostrado na Figura 4.

O *Tracker* é a única entidade centralizada do sistema *BitTorrent*, e tem como função centralizar e gerenciar as informações sobre os conjuntos dos *Peers* que estão compartilhando um mesmo conteúdo, conjuntos estes denominados *swarms*, como visto na Figura 4. Desta maneira, para que um *Peer* comece a participar de um *swarm*, ele se conecta a um *Tracker* e obtém uma lista de *Peers* com quem pode se conectar para compartilhar o arquivo desejado. O *Tracker* também mantém informações sobre o estado e taxas de transferência para cada *Peer*, primariamente utilizados para fins estatísticos.

O Repositório de *Torrents* normalmente é um servidor *web* público que disponibiliza um arquivo de metadados com extensão “.*torrent*” chamado de Arquivo *Torrent*. Este arquivo é gerado pelo usuário que disponibiliza o conteúdo pela primeira vez, e contém informações necessárias para que os *Peers* possam iniciar o compartilhamento do conteúdo. A presença do Repositório de *Torrents* não é obrigatória no sistema *BitTorrent* desde que os usuários interessados possam adquirir os arquivos *Torrent* desejados de outras maneiras, tais como e-mail, servidores FTP, discos removíveis, etc.

A Tabela 1 apresenta alguns dos dados presentes em um arquivo *Torrent* adquirido do *website* de distribuição do *Linux Ubuntu*³:

³ Arquivos .torrent do Ubuntu: <http://www.ubuntu.com/download/desktop/alternative-downloads>

Tabela 1 – Informações no arquivo *.torrent* do sistema operacional Ubuntu 12.04 32 bits.

<i>Campo</i>	<i>Valor</i>
<i>Endereço do Tracker</i>	http://torrent.ubuntu.com:6969/announce
<i>Tamanho do arquivo (em bytes)</i>	735358976
<i>Tamanho do pedaço (em bytes)</i>	524288
<i>Hash dos pedaços</i>	0x35363734303a4693ee84...

Os dados necessários são o endereço do *Tracker*, onde foram publicados o conteúdo e as informações sobre o conteúdo, incluindo: o tamanho do arquivo, o tamanho do pedaço no qual o arquivo foi subdividido (a ser explicado em mais detalhes na seção 2.3.2) e uma lista dos *hashes* (resumos criptográficos) produzidos pela função de *hash* SHA-1 de todos pedaços do arquivo, utilizados para verificar a integridade do conteúdo.

Quando um Cliente deseja adquirir um determinado conteúdo, ele deve seguir três passos (conforme apresentado na Figura 4):

- 1. Adquirir o arquivo *Torrent* com informações sobre o conteúdo desejado.**
- 2. Anunciar ao *Tracker* o desejo de participar de um *swarm*.**

Esse anúncio é efetuado na forma de uma requisição HTTP, cuja resposta é uma lista contendo um subconjunto de *Peers* do *swarm*. Isto é feito periodicamente ou em decorrência de eventos significativos, como saída do *swarm*, quando o Cliente não deseja mais compartilhar conteúdo com outros *Peers*, ou aquisição do conteúdo terminada, quando o Cliente passa a ser um *Seeder* no *swarm*. Este anúncio tem as funções de manter o *Tracker* atualizado sobre o status do Cliente e renovar a lista de *Peers* com os quais o Cliente pode se conectar.

- 3. Estabelecer conexões com os *Peers* retornados no passo 2 e iniciar a aquisição do conteúdo.**

Essas conexões empregam o protocolo *PeerWire*, que utiliza o TCP como base. A partir deste momento, o Cliente começa a obter o conteúdo dos *Peers* conectados e a compartilhar com o *swarm* os dados já adquiridos.

A primeira e segunda interação são simples, e não necessitam de maiores detalhes. A conexão entre os *Peers*, por sua vez, representa grande parte da complexidade do sistema *BitTorrent*, e portanto será explicada em detalhes nas seções 2.3.2 e 2.3.3.

2.3.2 Mecanismos de Controle

O *BitTorrent* utiliza três mecanismos de controle para melhorar o uso dos recursos e o compartilhamento de conteúdos entre *Peers*. Eles baseiam-se nos conceitos de *como* (a), *o quê* (b) e *para quem* (c) transmitir dados:

a) Como transmitir

O conteúdo é dividido em pedaços, o que permite que o Cliente adquira simultaneamente pedaços diferentes dos diversos *Peers* com os quais está conectado. Os pedaços são subdivididos em blocos, que é a menor unidade de dados a ser transferida, como exemplificado na Figura 5.

Figura 5 – Exemplo de divisão do conteúdo em pedaços e blocos

Ainda assim, *Peers* só compartilham um pedaço quando possuem todos os blocos localmente. As requisições por blocos são encadeadas para melhor utilizar a conexão e diminuir o tempo entre as requisições, conforme apresentado na Figura 6.

Figura 6 – Encadeamento de requisições

A Figura 6 mostra que o encadeamento de mensagens diminui o tempo entre requisições e permite ao Cliente realizar mais requisições em um mesmo intervalo de tempo.

b) O quê transmitir

A seleção de quais pedaços adquirir primeiro influencia na eficiência do *BitTorrent*. Quatro estratégias são utilizadas no funcionamento do *BitTorrent*:

- *Rarest First*: O Cliente mantém uma lista dos pedaços de todos os *Peers* conectados. Desta maneira, é possível selecionar os pedaços mais raros primeiro, pois eles têm maior probabilidade de se tornarem indisponíveis com a saída de *Peers* do *swarm*, o que impediria qualquer outro *Peer* de obter todo o conteúdo. Como cada *Peer* utiliza este critério de seleção, os pedaços tornam-se disponíveis em mais *Peers* mais rapidamente, melhorando assim a distribuição do conteúdo.
- *Random First*: Como o Cliente necessita ter algum conteúdo para fazer uso do mecanismo de *tit-for-tat*, os quatro primeiros pedaços são escolhidos aleatoriamente. Um pedaço escolhido aleatoriamente tem maior chance de estar mais presente na rede *BitTorrent*.
- *Strict Priority*: Quando pelo menos um bloco é requisitado, todos os outros blocos do mesmo pedaço são requisitados com a maior prioridade. Isso garante que os pedaços irão terminar o mais rápido possível
- *End-game Mode*: Essa estratégia funciona em nível de blocos, e tem, como objetivo, diminuir o tempo de aquisição dos últimos pedaços ao requisitar os últimos blocos a todos os *Peers* conectados.

c) Para quem transmitir

O número de *Peers* que obtém dados simultaneamente de um cliente deve ser limitado. Se todos os *Peers* conectados pudessem obter o conteúdo simultaneamente, as taxas de transferência alcançadas seriam pequenas, pois seria dividida entre todas as conexões ativas. Além do mais, não haveria compartilhamento justo de dados, pois qualquer *Peer* poderia adquirir dados desse cliente. Esse controle é feito aplicando-se o algoritmo de *Choking*, que será explicado em detalhes na seção 2.3.4.

2.3.3 O Protocolo *PeerWire*

O protocolo *PeerWire*, definido em (COHEN, 2008), consiste em mensagens de tamanho fixo sendo trocadas entre dois *Peers* utilizando uma conexão TCP. Esta conexão fica aberta enquanto os *Peers* desejarem se comunicar.

O Cliente deve manter as informações sobre o estado de todas as conexões com outros *Peers*. Cada conexão tem o seu estado definido pelas seguintes variáveis:

- **Interessado** (no *Peer*): Quando o Cliente está interessado no conteúdo do *Peer*.
- **Bloqueando** (o *Peer*): Quando o Cliente não irá responder nenhuma requisição feita pelo *Peer*.
- **Interessante** (para o *Peer*): Quando o *Peer* está interessado no conteúdo do Cliente.
- **Bloqueado** (pelo *Peer*): Quando o *Peer* não irá responder nenhuma requisição feita pelo Cliente.

O estado inicial de uma conexão é apresentado na Tabela 2.

Tabela 2 – Estado inicial de uma conexão *PeerWire*

<i>Variável</i>	<i>Valor</i>
<i>Interessado (no Peer)</i>	Falso
<i>Bloqueando (o Peer)</i>	Verdadeiro
<i>Interessante (para o Peer)</i>	Falso
<i>Bloqueado (pelo Peer)</i>	Verdadeiro

Assim que a conexão é estabelecida, nem o Cliente nem o *Peer* conhecem a distribuição de pedaços um do outro. Portanto, ambos estão se bloqueando e não estão interessados no conteúdo do outro.

Após estabelecer a conexão TCP, o Cliente e o *Peer* primeiramente trocam *Handshakes* e em seguida iniciam a troca de mensagens utilizando o protocolo *PeerWire*. Estas mensagens podem tanto alterar o estado da conexão quanto efetuar o intercâmbio de dados entre os *Peers*.

Como os detalhes das mensagens (campos e possíveis valores descritos na especificação oficial (COHEN, 2008), não são relevantes para a compreensão da operação do protocolo, somente os significados dessas mensagens são explicados:

- **Handshake:** Essa é a primeira mensagem que os *Peers* trocam entre si, assim que a conexão TCP é estabelecida. Ela identifica o *Peer* e o conteúdo que será compartilhado. A conexão PeerWire é estabelecida somente quando dois *Peers* trocam *Handshakes* e concordam com o conteúdo que será compartilhado.
- **BitField:** É a primeira mensagem trocada depois do estabelecimento da conexão PeerWire, e consiste de um mapa de bits indicando quais pedaços o Cliente possui, que fica armazenado no *Peer*. Essa mensagem não é obrigatória no caso do Cliente não possuir nenhum pedaço.
- **Have:** Toda vez que um pedaço é adquirido pelo Cliente, essa mensagem é enviada para todos os *Peers* conectados para que assim eles mantenham a lista de pedaços do Cliente atualizada.
- **Interested e Not Interested:** Essa mensagem é enviada pelo Cliente quando este muda o seu interesse no conteúdo do *Peer*. O Cliente pode ficar interessado no *Peer* na hora em que recebe a mensagem de *BitField* ou quando recebe uma mensagem de *Have* de um pedaço que ainda não possui. Em contraste, o Cliente pode perder interesse em um ou mais *Peers* toda vez que adquire um pedaço, se esse pedaço for o único que os tornavam interessantes.
- **Choke e Unchoke:** Quando deseja bloquear o envio de conteúdo para um *Peer*, o Cliente envia a mensagem de *Choke*, colocando o *Peer* no estado *Choked*. Um *Peer* bloqueado não deve enviar requisições para o Cliente, e se enviar, será ignorado. A mensagem de *Unchoke* serve para desbloquear o envio de dados para o *Peer*.
- **Piece:** O conteúdo é dividido em pedaços, e cada pedaço é dividido em blocos. Apesar do nome, esse tipo mensagem não corresponde a um pedaço, mas sim a um bloco do conteúdo (explicado na seção 2.3.2).
- **Request:** O Cliente envia essa mensagem para requisitar um bloco de dados para o *Peer*. Essa mensagem só poderá ser enviada se antes o Cliente tiver recebido uma mensagem de *Unchoke*.
- **Cancel:** Essa mensagem só é utilizada durante a estratégia *End-game Mode* (explicado na seção 2.3.2), para que o Cliente possa cancelar requisições duplicadas assim que receber a resposta de uma delas.
- **Keep Alive:** Se uma conexão fica inativa por um tempo pré-determinado (normalmente 2 minutos), o *Peer* fecha a conexão. O Cliente envia essa mensagem periodicamente para manter a conexão ativa.

Apesar de detalhar o formato das mensagens e os tipos e faixas de valores dos seus campos, a especificação (COHEN, 2008) é vaga nos detalhes de como dois *Peers* interagem pela troca de mensagens. Parte do estudo realizado nesse trabalho de mestrado consistiu em determinar como o protocolo *PeerWire* é orquestrado.

Com o objetivo de melhor compreender as mensagens *PeerWire*, o primeiro passo realizado neste processo de investigação foi identificar quais mensagens são recebidas e quais são enviadas, e qual a função que elas exercem na conexão entre o Cliente e o *Peer*. As funções identificadas foram de conexão, *download* e *upload*. A Tabela 3 mostra como as mensagens foram agrupadas de acordo com esse conceito:

Tabela 3 – Agrupamento das mensagens *PeerWire*, do ponto de vista do Cliente

<i>Função</i>	<i>Recebidas</i>	<i>Enviadas</i>
<i>Conexão</i>	Handshake, Keep Alive	BitField, Handshake, Keep Alive
<i>Download</i>	BitField, Choke, Have, Piece, Unchoke	Cancel, Interested, Not Interested, Request
<i>Upload</i>	Cancel, Interested, Not Interested, Request	Choke, Have, Piece, Unchoke

O agrupamento apresentado na Tabela 3 expõe uma característica que não é clara somente com a descrição das mensagens: as funções de *download* e *upload* são complementares, tendo em vista que as mensagens enviadas pelo Cliente para realizar o *download* do conteúdo são as mesmas recebidas do *Peer* como mensagens de *upload*, e vice versa. Por exemplo, o Cliente envia mensagens de *Choke* e *Unchoke* para controlar o upload de dados para o *Peer*. Porém, as mensagens de *Choke* e *Unchoke* recebidas controlam o download do Cliente.

Além disso, há o estabelecimento e manutenção da conexão, funções efetuadas pelas mensagens de *Handshake* e *Keep Alive*, respectivamente, independente do sentido de envio das mensagens. A mensagem de *BitField* tem relação com a função de conexão pois é enviada somente após o *Handshake*, e também tem relação com a função de *download*, pois ao ser recebida pelo Cliente, ele pode definir o interesse no *Peer*.

A Tabela 3 apresenta as mensagens somente do ponto de vista do Cliente. Se for considerado tanto o Cliente quanto o *Peer*, a interação entre os dois se torna mais clara, como apresentado na Figura 7:

Figura 7 – Troca de mensagens entre dois Peers do *BitTorrent*

A Figura 7 pode ser utilizada para separar o funcionamento do protocolo *PeerWire* em três mecanismos distintos, cada um controlando uma das funções identificadas (de conexão, *upload* ou *download*).

A partir desta interpretação, foi possível definir as máquinas de estado que governam o protocolo *PeerWire*, que foi no segundo passo realizado no processo de investigação. A partir dos mecanismos previamente definidos, três máquinas de estados foram desenvolvidas: *Conexão*, *Download* e *Upload*, conforme apresentadas na Figura 8, Figura 9 e Figura 10, respectivamente.

A máquina de estados de *Conexão* apresentada na Figura 8 tem por finalidade o estabelecimento e manutenção da conexão. O Cliente pode iniciar uma conexão ativamente com um *Peer* ou pode receber um pedido de conexão. Além do mais, estados desta máquina são definidos pelo envio e recepção do *Handshake*. A conexão pode ser fechada em qualquer estado, como por exemplo se o *Handshake* for recusado.

Figura 8 – Máquina de Estados de Conexão

Considere que o Cliente iniciou a conexão: primeiramente, ele envia o *Handshake* e então espera a recepção do *Handshake* do *Peer*. Quando esta mensagem chega no Cliente, este responde com uma mensagem de *BitField*, mas somente se o *Peer* possuir o conteúdo. No estabelecimento da conexão (estado em destaque na Figura 8), a máquina ativa dois temporizadores, um para enviar a mensagem de *Keep Alive*, se nenhuma outra mensagem for enviada no intervalo, e outro para fechar a conexão se o *Peer* conectado não enviar nenhuma mensagem no intervalo determinado.

Para definir os estados das máquinas de *Download* e *Upload*, os estados da conexão definidos anteriormente foram agrupados de acordo com os mecanismos correspondentes, conforme apresentado na Tabela 4.

Tabela 4 – Classificação dos estados da conexão

<i>Mecanismo</i>	<i>Estado</i>
<i>Download</i>	Interessado (no <i>Peer</i>)
	Bloqueado (pelo <i>Peer</i>)
<i>Upload</i>	Interessante (para o <i>Peer</i>)
	Bloqueando (o <i>Peer</i>)

A máquina de estados de *Download* (Figura 9) do Cliente tem como funções manter as informações sobre o conteúdo do *Peer* atualizadas primeiro com o recebimento da mensagem *BitField* e depois as mensagens *Have*, e controlar quando fazer as requisições por blocos. Quando a análise do conteúdo do *Peer* mostra que este tem

pedaços que o Cliente não possui, este se torna interessado e envia uma mensagem *Interested* pela conexão.

Figura 9 – Máquina de Estados de Download

As requisições só começarão a ser enviadas quando a máquina de estados de *Download* for desbloqueada pelo *Peer* com a recepção de uma mensagem *Unchoke* (estado em destaque na Figura 9). Quando completada a aquisição de um pedaço, além de enviar uma mensagem *Have* para **todas** as suas conexões, o Cliente verifica se algum dos *Peers* deixou de ser interessante. Nesse caso, irá enviar uma mensagem *Not Interested* para estes *Peers*.

Quando recebe uma mensagem *Have*, o Cliente pode ganhar interesse no *Peer* que a enviou. Por outro lado, quando conclui um pedaço, o Cliente pode perder o interesse em um ou mais *Peers* conectados. De qualquer forma, ele deve avisar esses *Peers* assim que houver uma mudança no estado de interesse, para que eles possam controlar quem será desbloqueado. Quando o Cliente obtém todos os pedaços e converte-se em *Seeder*, ele se desconecta de todos os outros *Seeders*, pois não há sentido em manter uma conexão onde não ocorrerá trocar de conteúdo.

Por fim, a máquina de estados de *Upload*, apresentada na Figura 10, gerencia quais *Peers* irão ser desbloqueados e poderão obter o conteúdo do Cliente. O mecanismo de *Choke* atua diretamente nas máquinas de estado de todas as conexões. Quando o *Peer* está interessado no Cliente e este recebe a transição para desbloquear a

conexão, toda mensagem *Request* que chegar será respondida com uma mensagem *Piece* correspondente (estado em destaque na Figura 10).

Figura 10 – Máquina de Estados de Upload

A compreensão completa da máquina de estados de *Upload* depende em parte do entendimento do mecanismo de *Choking*, a ser explicado na seção 2.3.4.

2.3.4 O algoritmo de *Choking*

O algoritmo de *Choking* é utilizado para garantir a condição de *tit-for-tat*, que consiste em privilegiar os *Peers* que compartilham mais conteúdo em detrimento dos *Peers free-riders* (que não compartilham seus dados com os outros *Peers*). Este objetivo é atendido oferecendo maior tempo de *download* para *Peers* de quem o Cliente conseguiu obter mais dados em um certo intervalo de tempo. Além do mais, como o número máximo de *Peers* que obtém conteúdo simultaneamente do Cliente é limitado (o padrão é quatro *Peers*), o Cliente precisa bloquear o restante dos *Peers* conectados. Esses *Peers* desbloqueados são chamados de *Peers* ativos, e é dito que eles estão ocupando as vagas de *upload* do Cliente. Um *Peer* que está bloqueado é denominado *choked*, e *unchoked* quando está desbloqueado.

O comportamento do algoritmo muda dependendo se o *Peer* está no estado *Leecher* ou *Seeder*. No primeiro caso, os *Peers* dos quais o Cliente conseguiu maior taxa de *download* de blocos tem prioridade no uso das vagas de *upload*. No segundo caso, a especificação original informa para considerar a taxa de *upload* ao invés da taxa de

download ao classificar os *Peers*, já que o *Seeder* não vai adquirir nenhum conteúdo. Essa implementação apresenta o problema de que *Peers* com maior capacidade de *download* monopolizam as vagas de *upload* do Cliente. A partir da versão 4.0.0 do cliente oficial, o algoritmo quando o Cliente está no modo *Seeder* foi alterado para priorizar os *Peers* com menor tempo como *Peer* ativo.

O algoritmo de *Choking* é executado a cada 10 segundos (valor padrão, configurável), o que define uma rodada do algoritmo. Rodadas extras são chamadas toda vez que um *Peer* desbloqueado muda o seu interesse no Cliente ou quando um dos *Peers* ocupando uma vaga de *upload* desconecta.

Com o *Peer* no modo *Leecher*, a seqüência lógica é:

1. A cada rodada, os *Peers* conectados são ordenados de acordo com a taxa de *download* do Cliente. Os *Peers* são desbloqueados da maior para a menor taxa de *download*, até que três *Peers* interessados sejam desbloqueados. Todos os outros *Peers* da lista são bloqueados. Os *Peers* que não tiverem enviado nenhum bloco para o Cliente nos últimos 30 segundos não entram nessa lista.
2. A cada três rodadas do algoritmo (ou seja, a cada 30 segundos), um *Peer* que está bloqueado e interessado é selecionado aleatoriamente e desbloqueado. Este *Peer* é o **desbloqueio otimista**, e ficará desbloqueado até o próximo ser selecionado depois de mais três rodadas.
3. A lista de *Peers* desbloqueados é verificada para ver se o *Peer* do desbloqueio otimista já foi desbloqueado no primeiro passo. Se for o caso, outro *Peer* dentre todos os conectados é selecionado aleatoriamente e desbloqueado.
4. Se o *Peer* selecionado está interessado no Cliente, a rodada termina e caso contrário o passo 3 é repetido.

Quando o *Peer* está no modo *Seeder*, a seqüência lógica é:

1. A cada rodada, os *Peers* que foram desbloqueados há no máximo duas rodadas atrás (20 segundos) ou que tem requisições por blocos pendentes são escolhidos.

2. Os *Peers* do passo 1 são ordenados primeiro de acordo com o instante de desbloqueio, do mais recente para o mais antigo, e segundo pela taxa de *upload*, com os mais rápidos primeiro.
3. O restante dos *Peers* é ordenado pela taxa de *upload*, e essa lista é concatenada na lista do passo 2.
4. A cada três rodadas, os três primeiros *Peers* da lista são desbloqueados e um quarto *Peer* é escolhido aleatoriamente. Nas outras duas rodadas, os quatro primeiros *Peers* são desbloqueados.

É importante notar que *Peers* que não estão interessados podem ser desbloqueados, mas por não serem *Peers* ativos não irão ocupar vagas de *upload*. Porém, toda vez que um desses *Peers* se tornar interessado, uma rodada extra do algoritmo de *Choking* é executada para que os *Peers* ativos possam ser reavaliados imediatamente e o limite pré-estabelecido seja mantido.

Outro aspecto importante é a função do *Peer* otimista: ao escolher um *Peer* aleatoriamente e mantê-lo por três rodadas do algoritmo, o Cliente analisa novos *Peers* e verifica se eles podem oferecer melhores taxas de transferência. Caso não houvesse esse mecanismo, os *Peers* com altas taxas de *upload* para o Cliente teriam o monopólio das vagas de *upload*.

2.4 COMPARAÇÃO COM ESPECIFICAÇÃO ORIGINAL

Como mencionado no início do capítulo, a especificação original (oficial) do *BitTorrent* (COHEN, 2008) provê uma descrição dos atores e das mensagens do protocolo *PeerWire*, porém descreve o comportamento dinâmico do sistema de maneira breve. Alguns trabalhos correlatos (KRYCZKA *et al.*, 2011; LEGOUT *et al.*, 2007; PARVEZ *et al.*, 2012; RASTI; REJAIE, 2007; XIA; MUPPALA, 2010) descrevem os mecanismos do *BitTorrent*, mas também possuem descrições breves.

A investigação realizada por este trabalho, que resultou na descrição apresentada nas seções anteriores, possui um nível de detalhes maior que as documentações oficiais. Para ilustrar esta diferença no aprofundamento das explicações, a Tabela 5

apresenta um comparativo entre detalhamento da explicação do *BitTorrent* contida neste trabalho e a apresentada pela especificação original (oficial) (COHEN, 2008).

Tabela 5 – Comparação sobre o nível de detalhamento da especificação original e a adotada neste trabalho

<i>Aspectos de Comparação</i>	<i>Especificação Original</i>	<i>Especificação Adotada</i>
<i>Atores</i>	Detalhada	Detalhada
<i>Interação entre atores</i>	Detalhada	Detalhada
<i>Formato do arquivo Torrent</i>	Detalhada	Não explicado
<i>Comunicação com o Tracker</i>	Detalhada	Explicação funcional, sem detalhes do formato
<i>Formato das mensagens PeerWire</i>	Detalhada, porém confusa. Não utiliza recursos gráficos ou texto estruturado	Explicação funcional de cada mensagem, sem detalhes dos campos que as compõem
<i>Dinâmica do protocolo PeerWire</i>	Superficial, além de não utilizar recursos gráficos ou texto estruturado.	Detalhada, com mensagens agrupadas mecanismos e máquinas de estado
<i>Mecanismos de controle</i>		Detalhada
<i>Algoritmo de Choking</i>		Detalhada

A explicação oficial do *BitTorrent* é feita de maneira não estruturada, em um texto corrido e sem tópicos ou seções, e não há o uso de recursos gráficos como figuras, diagramas ou tabelas para auxiliar no entendimento do sistema. Ademais, os comportamentos dinâmicos do protocolo *PeerWire* e do algoritmo de *Choking* não são definidos em detalhes, e somente as linhas gerais são explicadas.

As explicações mais técnicas, como o formato do arquivo *Torrent*, o formato das mensagens *PeerWire* e da comunicação com o *Tracker* não foram abordadas nesse momento, pois uma explicação unicamente funcional é o suficiente dentro do escopo deste trabalho.

2.5 CONSIDERAÇÕES PARCIAIS

Este capítulo descreveu o conceito de sistemas P2P e a importância do *BitTorrent* para a comunidade de pesquisa, seguido de uma descrição detalhada do sistema *BitTorrent*.

Foi enfatizado que a especificação oficial (original) do *BitTorrent* não descreve todos os mecanismos de maneira adequada para a compreensão do sistema. Por conta

disso, foi realizada uma investigação sobre o sistema *BitTorrent* utilizando outras fontes além da oficial para desenvolver uma descrição detalhada que pudesse ser utilizada no desenvolvimento do simulador de *BitTorrent*.

A contribuição principal deste capítulo é a explicação detalhada dos mecanismos internos do *BitTorrent* e do comportamento dinâmico da conexão entre dois *Peers*, descritos pelo uso de máquinas de estado. A descrição apresentada neste capítulo não está presente em nenhum dos trabalhos estudados na investigação do *BitTorrent* com mesmo nível de detalhes.

O próximo capítulo irá apresentar os trabalhos relacionados com o estudo do *BitTorrent* e os requisitos que devem ser atendidos para o desenvolvimento do simulador.

3 TRABALHOS RELACIONADOS E REQUISITOS DO SIMULADOR

Este capítulo apresenta trabalhos relacionados sobre *BitTorrent* com o objetivo de contextualizar o estudo do sistema *BitTorrent* no ambiente de pesquisa. Os métodos de análise utilizados e os objetivos correspondentes serão discutidos, dando um foco maior no método de simulação por conta da flexibilidade que os simuladores oferecem na avaliação de desempenho do *BitTorrent*.

A partir desses estudos, serão levantados os requisitos que um simulador de *BitTorrent* deve atender para que seja utilizado como ferramenta para o estudo de sistemas *BitTorrent*.

Por fim, os simuladores de *BitTorrent* disponíveis são analisados de acordo com os requisitos definidos anteriormente com o intuito de expor a necessidade da criação de um simulador que atende todos os requisitos.

3.1 TRABALHOS RELACIONADOS

A popularidade do *BitTorrent* resultou em um grande número de trabalhos realizados pela comunidade científica e acadêmica para estudar o seu desempenho. Alguns pesquisadores realizam medições em implementações reais do sistema, outros utilizam simulações e outros modelos analíticos.

A medição pode ser realizada em uma implementação real do sistema ou em um ambiente de testes, como o *PlanetLab* ou uma rede privada, e tem como objetivo compreender o comportamento dos mecanismos que compõem o sistema *BitTorrent*. Segundo (KRYCZKA *et al.*, 2011), medições podem ser divididas em duas categorias: medições de desempenho macroscópicas, que coletam dados de *Trackers* e Repositórios de *Torrent* para compreender a demografia de um ecossistema *BitTorrent*, e medições de desempenho microscópicas, que utilizam clientes modificados para participar de um *swarm* e assim analisar o desempenho do ponto de vista de um único *Peer*. Segue abaixo alguns dos trabalhos mais relevantes:

- Izal *et al.* (IZAL *et al.*, 2004) utilizou *logs* do *Tracker* da distribuição *Red Hat Linux*, coletados por 5 meses, e verificou que muitos dos usuários permaneciam como *Seeders* após completarem a aquisição do conteúdo. Além disso, através do uso de um cliente modificado, foi verificado que o algoritmo de *tit-for-tat* é eficiente.
- Pouwelse *et al.* (POUWELSE, J. A. *et al.*, 2005) caracterizou o ciclo de vida de *swarms* verificando os arquivos disponibilizados em um Repositório de *Torrents* popular chamado *Suprnova*. Este trabalho também utilizou um *script* para coletar do *Tracker* os endereços IPs dos *Peers* do *swarm*. Foi concluído que o *BitTorrent* consegue lidar bem com *flash-crowds*, que é o período logo após a disponibilização de arquivos populares, caracterizado por um grande número de *Peers* entrando no *swarm*.
- Legout *et al.* (LEGOUT *et al.*, 2007) efetuou experimentos com *swarms* privados localizados em um ambiente de testes do *PlanetLab*. O comportamento individual de alguns *Peers* modificados foi estudado para verificar os efeitos do algoritmo de *Choking*. Os resultados mostram que o algoritmo desempenha bem, e além disso, os *Peers* podem se agrupar em grupos com taxas de *upload* semelhantes.

Além disso, muitos modelos analíticos foram propostos para tentar descrever o comportamento do sistema *BitTorrent*. Estudos que utilizam este método permitem uma análise eficiente e detalhada dos parâmetros de um sistema *BitTorrent*, verificando o efeito decorrente da variação do valor de não só um parâmetro por vez, mas também o efeito decorrente da variação em mais de um parâmetro. Os trabalhos abaixo são os mais relevantes dentro do escopo da análise de desempenho pelo uso de modelos analíticos:

- Guo *et al.* (GUO *et al.*, 2007) utilizou dados coletados de dois *Trackers* dedicados além de dados de um Repositório de *Torrents*, e a partir deles criaram um modelo matemático para descrever a taxa de entrada de *Peers* em um *swarm* no decorrer do seu ciclo de vida. A partir deste modelo, eles estenderam o modelo fluido descrito em (QIU; SRIKANT, 2004). Os resultados apresentados demonstram como a entrada e saída de *Peers* em um *swarm* pode afetar o desempenho geral do mesmo.
- Liao, Papadopoulos e Psounis (LIAO; PAPADOPOULOS; PSOUNIS, 2007) consideram a existência de dois tipos de *Peer* em um *swarm*: usuários com alta

capacidade de banda e altas taxas de *upload*, e usuários com baixa capacidade de banda e baixas taxas de *upload*. Eles desenvolveram um modelo matemático para prever o tempo médio de aquisição do conteúdo para as duas classes de usuários. Além disso, eles propuseram uma modificação no algoritmo de *Choking* que privilegia os usuários com baixa capacidade de banda quando estes enviam pedaços para *Peers* com alta capacidade. Os resultados foram validados com o uso de simulação.

Por fim, por serem programas de computador, as simulações apresentam a flexibilidade em modelar os mecanismos do *BitTorrent*, além de facilidades em alterar os parâmetros e configurações dos cenários de estudo. O modelo programado do sistema pode ser tão preciso quanto necessário, sem deixar de considerar o compromisso entre a complexidade do modelo e o uso de recursos computacionais (FORTIER; MICHEL, 2003). Seguem abaixo dois trabalhos relevantes que utilizam simulação para apresentar os resultados:

- Bharambe, Herley e Padmanabhan (BHARAMBE; HERLEY; PADMANABHAN, 2006) avaliam o impacto dos mecanismos do *BitTorrent* no desempenho geral do sistema sob uma variedade de cenários de *flash-crowd* utilizando um simulador personalizado. Este simulador modela a rede do ponto de vista da conexão entre os *Peers*, sem levar em consideração os efeitos das camadas de rede inferiores. Os resultados mostram que o *BitTorrent* é robusto e escala bem, que o *tit-for-tat* não previne o comportamento de usuários egoístas e que o mecanismo de *rarest first* é crítico para garantir que novos *Peers* tenham algum conteúdo para trocar com outros *Peers*.
- Chen Tian *et al.* (CHEN TIAN *et al.*, 2008) estuda os efeitos do algoritmo de *Choking* no tráfego entre provedores de acesso à *Internet*. Como o *BitTorrent* não utiliza nenhum mecanismo de escolha de *Peers* baseado em proximidade, o tráfego entre provedores é alto, resultando em custos elevados para o provedor e em tempos de resposta elevados para os clientes. Eles utilizam um simulador personalizado para avaliar o desempenho do *BitTorrent*, empregando-se algoritmos de *Choking* que consideram a localização dos *Peers* conectados. Os resultados mostram que a carga do tráfego entre provedores de acesso pode ser passada para enlaces locais, diminuindo o custo para o provedor e a latência percebida pelos usuários.

Cada um dos métodos apresentados possuem vantagens e desvantagens dependendo do estudo a ser realizado (XIA; MUPPALA, 2010). A medição é indicada quando o objetivo é obter um entendimento mais aprofundado do comportamento dos usuários do *BitTorrent* ou o desempenho dos algoritmos existentes. Porém, a medição não é a solução ideal para o estudo de modificações dos mecanismos do sistema, uma vez que o custo de se criar um ambiente de testes e o tempo necessário para a sua configuração são altos. Os modelos analíticos são indicados para o estudo dos efeitos que diferentes parâmetros têm no desempenho do *BitTorrent*, mas por conta das abstrações matemáticas, não consegue prover um entendimento individual dos mecanismos mais complexos. Por fim, as simulações apresentam grande flexibilidade em variar os diversos mecanismos utilizados pelo *BitTorrent* e observar o impacto dessas mudanças no desempenho do sistema. Contudo, as simplificações do modelo podem gerar resultados não tão exatos, e a simulação sempre necessita de validação por um dos outros métodos.

A validação consiste em repetir o experimento com os mesmos parâmetros e o mesmo cenário utilizando ou medição ou modelagem analítica, e obter conclusões não divergentes sobre o sistema (FORTIER; MICHEL, 2003). O uso da medição permite verificar se o simulador apresenta resultados compatíveis com um cenário real, fornecendo um indicativo de que o modelo simulado oferece uma boa representação do sistema real. Apesar da modelagem analítica não oferecer resultados exatos, as tendências apresentadas pelo modelo analítico podem ser utilizadas para verificar o comportamento da simulação.

3.2 O USO DA SIMULAÇÃO NO ESTUDO DO BITTORRENT

As simulações são muito populares no estudo do desempenho do *BitTorrent* por conta da sua flexibilidade em alterar os mecanismos utilizados e em configurar os parâmetros e topologia dos cenários estudados. Uma extensa pesquisa realizada em (NAICKEN *et al.*, 2007) mostrou o uso de simuladores em trabalhos acadêmicos sobre P2P. Naicken estudou 141 artigos, e identifica que 70 desses artigos não especificam o simulador utilizado e nem oferecem maiores detalhes que permitam a

reprodução dos experimentos, impossibilitando a verificação dos resultados. Os simuladores utilizados nos outros 71 artigos são apresentados na Figura 11.

Figura 11 – Uso de simuladores para análise de sistemas P2P

Como pode ser visto na Figura 11, mais de 60% dos artigos utilizam simuladores personalizados, que foram desenvolvidos com o propósito específico de verificar os algoritmos e sistemas propostos no artigo. Como muitos desses simuladores não são disponibilizados para o público, a verificação dos resultados e a reprodução dos experimentos se tornam difíceis para outros pesquisadores. Dos simuladores que estão publicamente disponíveis para uso acadêmico, a maioria não possui mecanismos para coleta de estatísticas e/ou apresenta documentação insuficiente, o que justifica o grande número de trabalhos que utilizam simuladores personalizados.

O autor realizou uma pesquisa utilizando o mesmo método aplicado por (NAICKEN *et al.*, 2007), onde trabalhos relevantes à pesquisa de *BitTorrent* foram encontrados e aqueles que utilizam a simulação como método de análise foram identificados e classificados. O propósito desta pesquisa foi compreender o uso da simulação na pesquisa de *BitTorrent* e identificar o estado dos simuladores disponíveis publicamente para a comunidade de pesquisa.

Essa pesquisa mostrou que a simulação é utilizada para estudar mudanças nos mecanismos de sistemas *BitTorrent*, como por exemplo o uso de um algoritmo de seleção de *Peers* que leve em consideração a localidade geográfica dos *Peers* (BINDAL *et al.*, 2006; CHEN TIAN *et al.*, 2008), ou um algoritmo de *tit-for-tat* que

considera a troca de blocos ao invés da taxa de aquisição do conteúdo (BHARAMBE; HERLEY; PADMANABHAN, 2006). Esse tipo de estudo é difícil de ser realizado por meio de medições em sistemas reais, que exigiriam a mudança e a adoção de um cliente *BitTorrent* com os mecanismos alterados, ou em ambientes de testes controlado, que exigiria um grande número de máquinas e o controle remoto de todos os *Peers*. Além disso, a simulação também é utilizada para a validação de modelos analíticos (CHEN, X.; JARVIS, 2009; GUO *et al.*, 2007; HOSSEINI *et al.*, 2008).

Esta pesquisa identificou 28 trabalhos sobre *BitTorrent* que utilizam simulação para validação dos resultados. Assim como em (NAICKEN *et al.*, 2007), foi identificado que a maioria dos trabalhos utilizam simuladores personalizados ou não especificam detalhes sobre o simulador, impossibilitando a reprodução ou verificação dos resultados. Abaixo são apresentados os artigos pesquisados, agrupados de acordo com a maneira como as simulações foram executadas:

- Doze trabalhos não descreveram o tipo de simulador utilizado para obter os resultados (AGUILAR *et al.*, 2007; BUSTOS *et al.*, 2008; GUO *et al.*, 2007; HOSSEINI *et al.*, 2008; JIA *et al.*, 2011; KONRATH; BARCELLOS; MANSILHA, 2007; LI, P.; JOHN; XU, 2008; PARVEZ *et al.*, 2012; QIU; SRIKANT, 2004; SHAH; PARIS, 2007; SHRESTHA *et al.*, 2008; URVOY-KELLER, G.; MICHIARDI, 2006).
- Sete trabalhos desenvolveram simuladores personalizados que não foram disponibilizados publicamente (BINDAL *et al.*, 2006; CHEN TIAN *et al.*, 2008; CHEN, X.; JARVIS, 2009; LUAN; TSANG, 2006; WEI HUANG; CHUAN WU; LAU, 2010; WU; CHIUEH, 2006; ZHENYU LI; GAOGANG XIE, 2010).
- Dois trabalhos desenvolveram o próprio simulador e o disponibilizaram publicamente. O simulador desenvolvido por (BHARAMBE; HERLEY; PADMANABHAN, 2006), chamado *OctoSim*, está disponível em (BITTORRENT SIMULATOR, 2005). Yang e Abu-Ghazaleh (YANG; ABU-GHAZALEH, 2005) disponibilizaram o seu simulador em (GPS, 2005).
- Quatro trabalhos utilizaram simuladores desenvolvidos por outros. Os trabalhos realizados por (KE XU; YAHUI YANG; TAO CHEN, 2008) e (LIAO; PAPADOPOULOS; PSOUNIS, 2007) modificaram o simulador desenvolvido por (BHARAMBE; HERLEY; PADMANABHAN, 2006) para atender às suas

necessidades, enquanto os trabalhos realizados por (ZHONG; WANG, X.; KIHIL, 2011) e (BAYOUMY; SALAMA; ZAYED, 2011) modificaram o módulo de *BitTorrent* XXX desenvolvido para o simulador *PeerSim* (MONTRESOR; JELASITY, 2009).

- Três trabalhos desenvolveram o simulador com base em um arcabouço de simulação. Os trabalhos realizados por (VOGELEER, DE; ERMAN, DAVID; POPESCU, ADRIAN, 2008) e (KATSAROS, K. *et al.*, 2009) utilizaram o arcabouço OMNeT++ no desenvolvimento de seus simuladores, enquanto (SAAD *et al.*, 2010) desenvolveu o seu simulador com o uso do arcabouço NS-2. O trabalho de Katsaros foi disponibilizado publicamente em (KATSAROS, KONSTANTINOS, 2009)

Essa pesquisa mostra que são poucos os trabalhos que utilizam arcabouços de simulação para desenvolver seus simuladores. Além disso, a maioria dos trabalhos utiliza simuladores personalizados, porém somente dois disponibilizaram o código do simulador publicamente. Em adição, os trabalhos que utilizam simuladores personalizados não disponibilizados para o público apresentam as seguintes desvantagens: baixa credibilidade, pois não é possível para a comunidade acadêmica verificar a veracidade dos resultados obtidos; impossibilidade de reproduzir os resultados, pois não é possível dizer como os mecanismos sendo estudados foram modelados; e maior tempo de desenvolvimento, pois além do cenário a ser experimentado, se torna necessário desenvolver todos os mecanismos do simulador.

3.3 REQUISITOS PARA UM SIMULADOR DE *BITTORRENT*

Com a análise dos trabalhos relacionados, é possível identificar os seguintes requisitos para a construção de um simulador:

1. Deve permitir a configuração dos mecanismos do *BitTorrent*

Todos os trabalhos relacionados que utilizam simulação apresentam uma característica em comum: eles testam novos algoritmos ou mudanças nos mecanismos do *BitTorrent* (conforme descrito na seção 2.3.2). Portanto, um

simulador de *BitTorrent* deve ser configurável a ponto de permitir a seleção do algoritmo e a mudança dos parâmetros dos mecanismos que regem o sistema.

2. Deve permitir a coleta de estatísticas sobre o modelo

A coleta de estatísticas é essencial para um simulador, e como é apontado por (NAICKEN *et al.*, 2007), os simuladores disponíveis pecam nessa área ou por não possibilitar a configuração de quais dados coletar ou por não permitir a coleta de dados diferentes sem extensa modificação do código fonte.

3. Deve permitir a configuração de diferentes topologias de rede

Outra característica comumente estudada com o auxílio de simuladores é o estudo dos efeitos da topologia da rede no desempenho do *BitTorrent*. Portanto, a configuração da topologia deve ser facilitada para permitir a criação de cenários diferentes.

4. Deve ser validado com outro método de análise de desempenho

Todo simulador deve ter o seu comportamento validado por outro método de análise de desempenho, como comparação com resultados medidos de uma implementação real do sistema *BitTorrent*.

5. Deve permitir cenários com mais de 500 nós

Este número de *Peers* é o suficiente para realizar um estudo significativo sobre o *BitTorrent*. Izal *et al.* (IZAL *et al.*, 2004) mostra que swarms populares possuem de 3 a 500 *Peers* fora do período de *flash-crowd*. Além disso, o trabalho apresentado por (SZYDLOWSKI *et al.*, 2011) analisa o *BitTorrent* por meio de medição em um experimento com 500 nós em ambiente controlado.

O simulador desenvolvido por este trabalho deve, portanto, cumprir esses cinco requisitos para poder ser utilizado de maneira eficiente pela comunidade acadêmica como ferramenta de pesquisa de sistemas *BitTorrent*.

3.4 SIMULADORES DE *BITTORRENT* DISPONÍVEIS

Para compreender o cenário atual de simulação de *BitTorrent*, os simuladores disponíveis ao público foram analisados de acordo com os requisitos definidos na seção 3.3. São eles:

- **O simulador GPS** (YANG; ABU-GHAZALEH, 2005)
É um simulador de P2P de propósito geral implementado em *Java*. De acordo com os autores, a principal vantagem do simulador GPS sobre outros trabalhos é o seu modelo da rede, que incorpora detalhes da topologia de rede porém não necessita modelar a transmissão de pacotes. Com isso, a escalabilidade do simulador é grande, e os resultados apresentados são mais exatos. A topologia da rede é definida pelo modelo *Georgia Tech Internet Topology Model* (GT-ITM) (ZEGURA; CALVERT; BHATTACHARJEE, 1996).
- **O módulo *BitTorrent* para *PeerSim*** (MONTRESOR; JELASITY, 2009)
É um módulo desenvolvido para o *PeerSim*, um simulador P2P de propósito geral, também desenvolvido em *Java*. O *PeerSim* foi desenvolvido com o propósito de ser um simulador extensível, permitindo a adição de novos protocolos com a implementação de algumas poucas classes. O protocolo *BitTorrent* foi desenvolvido por Frioli e Pedrolli (FRIOLI; PEDROLLI, 2008), e se encontra disponível no site do *PeerSim*. Além disso, o *PeerSim* é altamente escalável, pois utiliza dois mecanismos diferentes de simulação: execução em ciclos, onde os protocolos de todos os nós são executados de maneira seqüencial e cíclica, e outro baseado em eventos. Para o primeiro mecanismo, somente a rede sobrejacente é modelada. O segundo mecanismo modela a camada de transporte para apresentar maior realismo.
- **O simulador *OctoSim*** (BHARAMBE; HERLEY; PADMANABHAN, 2006)
Escrito em C#, foi desenvolvido com o propósito de estudar os mecanismos principais do *BitTorrent*. O simulador foi então disponibilizado publicamente pelos autores, e desde então já foi utilizado por outros trabalhos. Os detalhes das camadas de rede inferiores não são simulados, somente a rede sobrejacente.
- **O BIT-SIM** (KATSAROS, K. *et al.*, 2009)
Foi desenvolvido com o objetivo de ser uma ferramenta para a comunidade de pesquisa, e utiliza o arcabouço OMNeT++ versão 3 como base. Ele prove dois

modelos das camadas inferiores de rede: o primeiro é direcionado para exatidão do modelo e utiliza o modelo de rede do arcabouço INET, enquanto o segundo é focado em escalabilidade e utiliza o arcabouço *OverSim* para modelar a rede, onde os nós da rede trocam mensagem diretamente entre si. Além disso, a entrada e saída de *Peers* é controlada também pelo arcabouço *OverSim*, e também utilizam o GT-ITM para construir a topologia.

Estes simuladores foram analisados baseados na inspeção de código fonte, documentação disponível e artigos relacionados. Os resultados desta análise são apresentados nas seções seguintes.

3.4.1 Análise do simulador GPS

Os seguintes requisitos não foram atendidos pelo simulador GPS:

- **Configuração dos mecanismos do *BitTorrent***

Os mecanismos do *BitTorrent* são implementados em classes diferentes para facilitar a sua modificação ou substituição, porém só é possível selecionar os mecanismos ou alterar os seus parâmetros por meio de alterações no código fonte do simulador. Esta característica dificulta a comparação de diferentes algoritmos ou o estudo dos parâmetros dos mecanismos, uma vez que cada alteração no código fonte resulta na recompilação do simulador.

- **Coleta de estatísticas sobre o modelo**

Não há controle de quais estatísticas serão produzidas, sendo possível somente determinar em qual diretório o arquivo com as estatísticas será salvo. Além do mais, as únicas estatísticas presentes no modelo são os tempos de início e término da aquisição do conteúdo.

- **Validação do simulador**

O modelo de redes é validado com simulações do TCP realizadas no NS-2, e o modelo do *BitTorrent* é validado através de medições em um ambiente de testes de baixa escala (10 nós). Os autores do GPS afirmam que a validação do modelo do *BitTorrent* não é representativa por conta da baixa quantidade de nós, e expressaram o desejo de realizar validações com uma maior quantidade de nós. Contudo, nenhum trabalho subsequente foi realizado.

Os requisitos atendidos pelo simulador são:

- **Configuração das topologias de rede**

Utiliza arquivos de texto para configurar a topologia e as capacidades dos enlaces. A rede é dividida em redes de trânsito e redes de borda, de acordo com o modelo GT-ITM (ZEGURA; CALVERT; BHATTACHARJEE, 1996), e a camada de transporte é modelada utilizando três algoritmos diferentes. O mais simples, chamado *Peer Based Bandwidth* (PBB), atribui a largura de banda disponível para cada *Peer* ao invés de considerar os enlaces. Essa escolha é feita pois o cálculo é mais simples, uma vez que não é necessário calcular o uso de todos os enlaces da rede. O segundo protocolo estende o protocolo PBB para levar em consideração a topologia da rede, e é chamado *Dynamic Link-level Network Bandwidth* (BLNB). O terceiro é chamado *TCP Based Flow* (TBF), e consiste em um modelo matemático que estima a largura de banda disponível considerando que o TCP é limitado pelos erros e congestionamentos, e não pela capacidade dos enlaces.

- **Quantidade de nós**

O simulador GPS foi capaz de executar um cenário de simulação com 1054 *Peers* em 128 minutos de simulação.

3.4.2 Análise do módulo *BitTorrent* para PeerSim

Os seguintes requisitos não foram atendidos pelo módulo *BitTorrent* para PeerSim:

- **Configuração dos mecanismos do *BitTorrent***

Somente parte deste requisito é atendido pelo módulo *BitTorrent*. Os parâmetros podem ser configurados a partir de um arquivo de configuração, pois todo módulo desenvolvido para o *PeerSim* precisa implementar uma classe de inicialização que efetua a leitura deste arquivo. Porém, os mecanismos do *BitTorrent* estão implementados em uma única classe, tornando modificações ou substituições complicadas de serem realizadas.

- **Configuração das topologias de rede**

A rede é modelada como conexões diretamente efetuadas entre os *Peers*, com a latência calculada pela ferramenta *King* (GUMMADI; SAROIU; GRIBBLE, 2002),

que modela a latência entre dois nós quaisquer na Internet. Portanto, não há nenhum controle sobre a topologia da rede.

- **Validação do simulador**

Nem o *PeerSim* nem o módulo *BitTorrent* foram validado pelos autores. Além disso, os trabalhos de (ZHONG; WANG, X.; KIHIL, 2011) e (BAYOUMY; SALAMA; ZAYED, 2011), que utilizam o módulo *BitTorrent*, assumem que o simulador já está validado.

Os requisitos atendidos pelo simulador são:

- **Coleta de estatísticas sobre o modelo**

Assim como a classe de inicialização dos protocolos, módulos programados para o *PeerSim* precisam implementar uma classe de captura de estatísticas do protocolo. Esta classe possui acesso ao protocolo e pode obter qualquer informação durante a execução da simulação em intervalos determinados.

- **Quantidade de nós**

O autor do módulo de *BitTorrent* para *PeerSim* não apresenta nenhum dado sobre a quantidade de nós simulados. Porém os autores do *PeerSim* afirmam que o simulador é capaz de simular centenas de milhares de nós, mesmo para os algoritmos mais complexos.

3.4.3 Análise do simulador *OctoSim*

Os seguintes requisitos não foram atendidos pelo simulador *OctoSim*:

- **Configuração dos mecanismos do *BitTorrent***

Apesar de possibilitar a escolha de algoritmos diferentes, todos os algoritmos implementados estão descritos em uma única classe. Isso faz com que qualquer alteração resulte em extensa modificação do código. Além disso, as configurações são definidas em outra classe, portanto alterações nos valores dos parâmetros resultam em recompilação do simulador.

- **Coleta de estatísticas sobre o modelo**

A coleta de estatísticas também está espalhada pelo código, e não há configuração de quais são os dados coletados pelo simulador

- **Validação do simulador**

O autor do *OctoSim* não realiza nenhuma validação dos resultados apresentados no seu trabalho, e os trabalhos que modificaram e utilizaram o *OctoSim* (KE XU; YAHUI YANG; TAO CHEN, 2008; LIAO; PAPADOPOULOS; PSOUNIS, 2007) também não realizaram nenhuma validação.

- **Configuração das topologias de rede**

A rede modelada não considera a topologia da rede, somente a conexão entre os *Peers*.

O único requisito que o *OctoSim* atende é o seguinte:

- **Quantidade de nós**

Os autores apresentam resultados de uma simulação com 8 mil *Peers*.

3.4.4 Análise do simulador BIT-SIM

Os seguintes requisitos não foram atendidos pelo simulador BIT-SIM:

- **Configuração dos mecanismos do *BitTorrent***

Todos os mecanismos do *BitTorrent* estão modelados em uma única classe, o que dificulta a modificação do seu funcionamento. Contudo, a configuração dos parâmetros é suportada pelo arcabouço OMNeT++, que efetua a leitura dos arquivos de configuração automaticamente.

- **Validação do simulador**

Os autores só fornecem dados sobre o desempenho do simulador quanto ao uso de memória e o tempo de simulação de acordo com o número de *Peers* simulados, e não efetuam a validação dos resultados.

- **Quantidade de nós**

Os autores apresentam, no seu trabalho, testes com no máximo 120 *Peers* na rede, e consideram que o compromisso entre o realismo do modelo e a escalabilidade é aceitável.

Os seguintes requisitos são cumpridos pelo BIT-SIM

- **Configuração das topologias de rede**

A topologia da rede é totalmente configurável por conta das capacidades do arcabouço OMNeT++. Além disso, a rede é modelada por outro arcabouço chamado INET, que considera desde a camada de enlace até a camada de transporte, fornecendo um maior realismo para o simulador.

- **Coleta de estatísticas sobre o modelo**

O núcleo do arcabouço OMNeT++ fornece as ferramentas para a coleta de estatísticas e permite a configuração individual de quais estatísticas serão produzidas na execução da simulação. Além disso, o OMNeT++ fornece também as ferramentas para efetuar a análise dos resultados coletados.

3.4.5 Considerações sobre a análise dos requisitos

Conforme apresentado nesta análise, nenhum dos simuladores de *BitTorrent* disponíveis para a comunidade acadêmica consegue atender a todos os requisitos de simulação. A Tabela 6 apresenta um resumo dos resultados obtidos:

Tabela 6 – Verificação dos requisitos para simulador de BitTorrent nos simuladores disponíveis

<i>Requisitos</i>	<i>GPS</i>	<i>PeerSim</i>	<i>OctoSim</i>	<i>BIT-SIM</i>
<i>Permitir a configuração dos mecanismos</i>	✗	✗	✗	✗
<i>Permitir a coleta de estatísticas</i>	✗	✓	✗	✓
<i>Permitir a configuração da topologia</i>	✓	✗	✗	✓
<i>Tem resultados validados por outros métodos</i>	✗	✗	✗	✗
<i>Permitir cenários com mais de 500 Peers</i>	✗	✗	✓	✗

A Tabela 6 mostra que nenhum dos simuladores analisados consegue preencher todos os requisitos, sendo o mais preocupante a validação dos resultados do simulador.

3.5 CONSIDERAÇÕES PARCIAIS

Neste capítulo foram apresentados os trabalhos relacionados à análise de desempenho do *BitTorrent*, com o intuito de compreender quais aspectos do sistema são estudados pela comunidade de pesquisa, com foco nos trabalhos que utilizaram a simulação para a avaliação.

Foi enfatizado que os trabalhos do *BitTorrent* visam o estudo dos mecanismos que controlam a interação entre os *Peers*, e que a simulação é a melhor ferramenta para este tipo de análise por conta de sua flexibilidade. Utilizando a simulação é possível modelar diferentes algoritmos e alterar os parâmetros dos mecanismos, e verificar os efeitos no desempenho do sistema, o que não é possível utilizando a medição de sistemas reais ou a modelagem analítica. Ademais, foi mostrado que a maioria dos trabalhos que utilizam simulação desenvolve os seus próprios simuladores, o que dificulta a verificação e replicação dos resultados apresentados.

O levantamento dos trabalhos relacionados possibilitou a definição dos requisitos necessários para um simulador de *BitTorrent*. Os simuladores de *BitTorrent* publicamente disponíveis foram analisados, e foi mostrado que nenhum deles consegue atender a todos os requisitos previamente definidos. Considerando a popularidade do *BitTorrent* e o número estudos que utilizam simulação como método de análise, o desenvolvimento de um simulador de *BitTorrent* capaz de atender a todos os requisitos de simulação definidos é importante para a comunidade de pesquisa.

O próximo capítulo irá discutir a necessidade de se utilizar um arcabouço de simulação no desenvolvimento do simulador, a fim de diminuir o trabalho de desenvolvimento e produzir um simulador que possa atender aos requisitos definidos neste capítulo. Os arcabouços disponíveis serão apresentados e comparados, a fim de escolher o que melhor se adéqua às necessidades do simulador.

4 ARCABOUÇOS DE SIMULAÇÃO E O OMNET++

O uso de um arcabouço para o desenvolvimento do simulador de *BitTorrent* permite o desenvolvimento seja focado somente no modelo, sem a preocupação de desenvolver também os mecanismos do simulador, tais como a fila de eventos, a geração de números aleatórios e distribuições estatísticas, as ferramentas para coleta de estatísticas, os arquivos de configuração dos experimentos, etc.

Este capítulo realiza uma análise comparativa dos arcabouços NS-3 e OMNeT++, e será mostrado que o OMNeT++ é o mais apropriado no desenvolvimento do simulador do *BitTorrent*.

A seguir, serão apresentados em detalhes o arcabouço de simulação OMNeT++, escolhido para servir de base para o desenvolvimento do simulador de *BitTorrent*, e o arcabouço INET, que oferece modelos das camadas de rede para serem usadas por simuladores desenvolvidos com OMNeT++.

4.1 ARCABOUÇOS DISPONÍVEIS

Dois arcabouços foram considerados por conta de sua popularidade para o desenvolvimento de simuladores de redes de computadores: o NS-3⁴ e o OMNeT++⁵. O arcabouço NS-2, apesar da sua grande popularidade, não foi considerado por não ser apropriado para o desenvolvimento de modelos na camada de aplicação (NAICKEN *et al.*, 2007).

O NS-3 é um arcabouço que foi desenvolvido para ser um substituto do NS-2, e tem como objetivo superar diversas de suas limitações, tais como o processamento paralelo, melhor computação estatística, melhor coleção de dados e assim por diante⁶. O OMNeT++ é um arcabouço de simulação modular, e que apesar de oferecer facilidades para desenvolver simulações de redes de computadores, não

⁴ Página do arcabouço NS-3: <http://www.nsnam.org/>

⁵ Página do arcabouço OMNeT++: <http://omnetpp.org/>

⁶ Objetivos do Projeto do NS-3: www.nsnam.org/docs/meetings/wns2/ns-3-project-goals.ppt

está limitado a esta área de pesquisa. Qualquer modelo que pode ser descrito por módulos trocando mensagens entre si podem ser modelados no OMNeT++.

Os critérios apresentados por (NAICKEN *et al.*, 2007) foram utilizados para comparar os arcabouços a fim de escolher o melhor para o desenvolvimento do simulador de *BitTorrent*. Estes critérios são:

- **Arquitetura**

Está relacionada a como os modelos desenvolvidos com o arcabouço são organizados. O nível de extensibilidade (adição de novas funcionalidades) e reuso do modelo dependem da arquitetura utilizada, que pode ser dividido em:

- **Elementos básicos:** Qual são os conceitos básicos e abstrações utilizadas no desenvolvimento de um modelo.
- **Organização:** Como os elementos modelados são organizados e como eles interagem entre si.

- **Usabilidade**

Refere-se à curva de aprendizagem e facilidades para o desenvolvimento presentes no arcabouço. Pode ser dividido nos itens:

- **Linguagens de programação:** Quais são as linguagens utilizadas para desenvolver os modelos.
- **API disponível:** Quais são as funções contidas na API do arcabouço. As funcionalidades mais importantes que devem estar presentes são: geradores de números aleatórios, funções geradoras de distribuição e funções relacionadas ao tráfego de dados, como cálculo de rota e atraso de transmissão.

- **Ambiente de desenvolvimento**

Qual é o ambiente no qual a simulação é desenvolvida. Uma IDE (*Integrated Development Environment*) voltada para o desenvolvimento com o arcabouço é opcional, porém de grande utilidade se disponível.

- **Depuração:** Qual o nível de depuração oferecido pelo arcabouço.
- **Estatísticas:** Diz respeito à coleção de dados do modelo e quais ferramentas de captura estão disponíveis.
- **Modelagem da rede subjacente:** Refere-se aos modelos disponíveis de redes de computadores e o seu nível de detalhes, utilizados como base para as simulações desenvolvidas.

- **Estabilidade**

Significa como o arcabouço é visto pela comunidade acadêmica. Este critério é dividido em:

- **Uso acadêmico:** Qual a popularidade do arcabouço no meio acadêmico, e quantos trabalhos o utilizam como ferramenta.
- **Versão atual:** Qual o estágio do desenvolvimento do arcabouço.

Com estes critérios, os arcabouços NS-3 e OMNeT++ foram comparados de maneira sistemática. Ambos os arcabouços comparados são usados para desenvolver simulações de eventos discretos.

4.2 ESCOLHA DO ARCABOUÇO

O resultado da comparação e a escolha do arcabouço são apresentados nas seções abaixo.

4.2.1 Arquitetura

A comparação das características arquiteturais é apresentada na Tabela 7.

Tabela 7 – Comparação entre as arquiteturas

	<i>NS-3</i>	<i>OMNeT++</i>
<i>Elementos Básicos</i>	Aplicação, Nó, Dispositivo de Rede, Pacote e Canal	Módulo, Mensagem, Porta e Enlace
<i>Organização</i>	Nós de rede conectados entre si	Módulos genéricos organizados de maneira hierárquica

Os arcabouços apresentam conceitos fundamentalmente diferentes no desenvolvimento de seus modelos. O NS-3 oferece abstrações para elementos reais de redes de computadores, e é voltado exclusivamente para o desenvolvimento de simuladores de redes. O OMNeT++, por outro lado, apresenta o conceito de módulos genéricos que são combinados como blocos de montar em arquiteturas

mais complexas, o que permite o desenvolvimento de modelos fora do domínio de redes de computadores (por exemplo, um modelo de um sistema de arquivos⁷).

Os elementos básicos utilizados pelo NS-3 no desenvolvimento de modelos são os seguintes:

- **Aplicação:** É onde os dados são gerados ou consumidos, como uma aplicação de redes em um computador real.
- **Nó:** É a representação de um nó da rede, composto por Dispositivos de Rede, Aplicações e uma pilha de protocolos.
- **Dispositivo de Rede:** É a representação de uma placa de interface de rede (NIC – *Network Interface Card*), e define a interface de comunicação do Nó. Somente dois Dispositivos de Rede iguais podem ser conectados, e devem utilizar um Canal adequado. Por exemplo, uma placa Ethernet só consegue se comunicar com outra placa Ethernet utilizando um enlace físico comum com a mesma taxa de transferência de dados.
- **Pacote:** É a unidade de informação da simulação trocada entre Aplicações, e é responsável pela geração de eventos no simulador.
- **Canal:** Representa o meio físico pelo qual Pacotes trafegam. É possível simular desde um canal simples como um fio com atraso de transmissão até uma área com obstáculos para comunicações sem fio.

Os elementos básicos que compõem uma simulação desenvolvida no OMNeT++ são os seguintes:

- **Módulo:** Modelam comportamentos simples, e podem ser agregados para modelar comportamentos mais complexos.
- **Mensagem:** Os módulos se comunicam através da passagem de mensagens, que são responsáveis pelos eventos do simulador. Estas mensagens podem ser utilizadas para modelar qualquer entidade móvel do modelo, tais como pacotes de dados transmitidos entre nós de uma rede ou trabalhos ou usuários em um sistema de filas.

⁷ Simulação de um sistema de arquivos feita com o OMNeT++:
<http://www.omnetpp.org/filemgmt/singlefile.php?lid=104>

- **Porta:** É a entrada ou saída dos módulos, e o ponto de conexão entre um módulo e outro.
- **Enlace:** É a conexão entre duas portas, e pode possuir os parâmetros de atraso de propagação, taxa de erro de bits e taxa de transmissão de dados.

No OMNeT++, os modelos são formados pela organização hierárquica dos módulos, onde módulos mais complexos são formados pela composição de módulos mais simples. O NS-3, por outro lado, não apresenta essa funcionalidade, a não ser por meio de técnicas de programação do modelo.

4.2.2 Usabilidade

A comparação dos arcabouços pelo ponto de vista de usabilidade é disposta na Tabela 8.

Tabela 8 – Comparação entre a usabilidade

	<i>OMNeT++</i>	<i>NS-3</i>
<i>Linguagens de Programação</i>	C++, NED, arquivos de configuração	C++
<i>API disponível</i>	Distribuições estatísticas, auxiliares de topologia, extensões do arcabouço	Distribuições estatísticas, auxiliares de topologia, protocolos de rede
<i>IDE disponível</i>	Edição visual dos módulos, IDE de C++	Não específica
<i>Depuração</i>	Execução passo a passo na IDE	Mensagens de log na linha de comando

O NS-3 utiliza C++ para desenvolver os modelos e criar os cenários de experimentação. Por outro lado, o OMNeT++ utiliza linguagens diferentes para descrever o modelo, a topologia e a configuração dos parâmetros da simulação. Os modelos são programados em C++, a hierarquia dos módulos e topologia são descritos com arquivos NED (NEtwork Description) e a configuração é feita com arquivos de texto. Esta separação provê maior flexibilidade, uma vez que topologias diferentes podem ser desenvolvidas com os mesmos modelos simples, e parâmetros diferentes podem ser testados mudando-se os arquivos de configuração.

Quanto à API disponível, ambos os arcabouços provêm funcionalidades semelhantes, tais como geradores de números aleatórios a partir de diferentes

distribuições estatísticas, funções para calcular rotas e configurar a topologia. O NS-3 provê modelos das camadas de transporte (TCP e UDP), redes (IPv4 e IPv6), e enlace (ARP). O OMNeT++ não suporta diretamente esses modelos, porém é possível utilizar modelos disponíveis na página do arcabouço⁸ para acrescentar funcionalidades à simulação, como por exemplo modelos relacionados à Internet e modelos de Redes Sem Fio IEEE 802.11.

O OMNeT++ disponibiliza uma IDE personalizada para trabalhar com a edição de arquivos NED, arquivos de configuração e análise dos dados estatísticos. O NS-3 não provê uma IDE, porém os modelos podem utilizar IDEs voltadas para o desenvolvimento em C++. Além disso, ferramentas como o Gnuplot⁹ podem ser utilizadas para análise dos dados.

Por fim, simuladores desenvolvidos tanto com o NS-3 quanto com o OMNeT++ podem ser depurados com técnicas de depuração de programas em C++, tais como impressão de mensagens e o uso de depuradores. Porém, o OMNeT++ oferece um modo de execução visual da simulação, que possibilita a execução passo a passo de eventos e recursos visuais de inspeção do modelo.

4.2.3 Estatísticas

As funções estatísticas e a capacidade de coletar dados para cada arcabouço são listadas na Tabela 9.

Tabela 9 – Comparação entre as capacidades estatísticas

	OMNeT++	NS-3
Coleção de dados	Geração de estatísticas e objetos ouvintes, formato próprio	Geração de estatísticas e objetos ouvintes, formato livre, coleção de dados de pacote

Tanto o NS-3 quanto o OMNeT++ utilizam o mesmo conceito para efetuar captura de estatísticas: valores são emitidos pelo modelo e são utilizados por objetos que estão capturando estas emissões. No NS-3, a saída de dados é formatada da

⁸ Catálogo de modelos do OMNeT++: <http://www.omnetpp.org/models/catalog/>

⁹ <http://www.gnuplot.info/>

maneira que o desenvolvedor desejar, enquanto no OMNeT++, os dados são gerados em formatos específicos, que podem ser histogramas, vetores de instantes de tempo e valores numéricos. O NS-3 pode gerar arquivos pcap (captura de pacotes), utilizados por ferramentas como *Dumpcap*, *TCPdump* e *Wireshark*. O OMNeT++ não possui esta capacidade nativamente.

4.2.4 Modelagem da rede subjacente

O tipo de simulação da rede subjacente utilizada nos modelos é identificado na Tabela 10 para ambos os arcabouços.

Tabela 10 – Comparação entre a modelagem da rede subjacente

	OMNeT++	NS-3
<i>Nível de detalhes</i>	Não possui suporte nativo, mas existem arcabouços que podem estender o modelo	Suporta nativamente a simulação desde a camada física até a camada de aplicação

As simulações feitas em NS-3 utilizam o modelo de redes desde a camada física, em nível de pacotes, até a camada de aplicação. O OMNeT++ não possui suporte nativo à simulação de redes, porém existem arcabouços que provém essa capacidade, tais como o INET.

4.2.5 Estabilidade

Esta pesquisa foi realizada em meados de 2009, portanto a estabilidade dos dois arcabouços na época da comparação é apresentada na Tabela 11.

Tabela 11 – Comparação entre a estabilidade

	OMNeT++	NS-3
<i>Uso em publicações</i>	Maior número de referências	Menor número de referências
<i>Versão</i>	Estável 4.0, 12 de Março de 2009	Estável 3.3, 18 de Dezembro de 2008

O NS-3 era um projeto relativamente novo na época da comparação, portanto não haviam muitos trabalhos desenvolvidos a partir dele. A primeira versão estável foi lançada em Dezembro de 2008. Ainda assim, modelos feitos em NS-2 foram refeitos para NS-3 e os resultados apresentados foram próximos o suficiente para validar o funcionamento do NS-3. O OMNeT++ é desenvolvido desde 1998, e já teve diversas versões lançadas. Por conta disso, o número de trabalhos desenvolvidos a partir dele é maior. Ambos os simuladores são mantidos por comunidades ativas, que garantem a estabilidade e a correção dos arcabouços.

4.2.6 Resultado da comparação

A comparação entre os arcabouços NS-3 e OMNeT++ precisam levar em consideração as métricas definidas no capítulo 3, a saber:

- Facilidade de configuração do modelo – o arcabouço deve facilitar a configuração do modelo, permitindo a troca dos algoritmos e dos parâmetros do *BitTorrent*, além de permitir a configuração da topologia de rede.
- Ferramentas para análise estatística – o arcabouço deve ter ferramentas que facilitem a coleta de estatísticas para análise posterior.
- Maturidade do arcabouço – o uso do arcabouço pela comunidade acadêmica é um indicativo da sua estabilidade e nível de aceitação.

O arcabouço escolhido para o desenvolvimento do simulador de *BitTorrent* é o OMNeT++ pelos seguintes motivos:

- O primeiro requisito de configurabilidade do modelo é melhor atendido pelo arcabouço OMNeT++ por conta da separação entre o modelo, a topologia e a configuração das simulações. As configurações são lidas e validadas pelo ambiente provido pelo OMNeT++, e é possível realizar estudos de parâmetros com poucas linhas de configuração.
- O segundo requisito de ferramentas de análise estatística é atendido pelos dois arcabouços.
- O arcabouço OMNeT++ é mais maduro que o NS-3, e na época em que esta comparação foi realizada, em meados de 2009, o NS-3 era relativamente novo.

4.3 O ARCABOUÇO DE SIMULAÇÃO OMNET++

O OMNeT++ é um arcabouço de simulação de eventos discretos escrito em C++, modular, e voltado para o desenvolvimento de simulações de redes de computadores. O arcabouço OMNeT++ em si não é um simulador, mas sim uma infraestrutura e um conjunto de ferramentas que permitem o desenvolvimento de simuladores. Além disso, por possuir uma arquitetura genérica, não está limitado ao desenvolvimento de modelos de redes de computadores, podendo ser utilizado para criar modelos nos mais diferentes domínios.

O arcabouço OMNeT++ apresenta um conjunto de características interessantes que auxiliam no desenvolvimento de simuladores:

- A arquitetura modular proporciona maior reuso de código, e o desenvolvimento de modelos mais complexos pode ser feito pela composição de módulos mais simples. Este conceito é utilizado por outros arcabouços, que são conjuntos de módulos modelados em OMNeT++ com o propósito de serem utilizados por outros desenvolvedores.
- As topologias são descritas por uma linguagem própria chamada NED (Network Description Language), com o intuito de separar os conceitos de topologia e de comportamento do modelo. É possível criar topologias diferentes sem alterar o código dos modelos.
- Os parâmetros do simulador e do modelo são lidos de arquivos de configuração pelo núcleo do OMNeT++, que verifica a validade dos valores passados, facilitando a criação de diferentes cenários e o estudo do efeito dos parâmetros no desempenho do modelo.
- O modo gráfico de execução permite a inspeção do modelo durante a execução de uma simulação, e o controle dos eventos passo a passo, facilitando a depuração do modelo.
- A IDE oferece suporte para o desenvolvimento, com facilidades tanto para o desenvolvimento de códigos em C++ quanto para a criação e modificação de arquivos NED e arquivos de configuração.

- O OMNeT++ é portátil, e pode ser instalado no Linux, Windows ou Mac OS. Além disso, a execução de um modelo precisa de modificações mínimas para serem portadas de um sistema para outro.

4.3.1 Conceitos de modelagem

Um modelo do OMNeT++ consiste de módulos que se comunicam com a troca de mensagens. Em adição, existem cinco conceitos utilizados para criar um modelo no OMNeT++: módulo, porta, canal, mensagem e sinal, a serem explicados abaixo.

Os módulos onde as mensagens são geradas e consumidas são denominados *módulos simples*, sendo programados em C++. Módulos mais complexos podem ser criados pela composição de módulos mais simples, e são denominados *módulos compostos*. Módulos compostos podem ser utilizados como submódulos de outros módulos compostos, sem limite para o nível de hierarquias. A Figura 12 exemplifica esta arquitetura:

Figura 12 – Elementos básicos do OMNeT++

Na Figura 12, estão representados em cinza os módulos simples, programados em C++, e os módulos compostos. O módulo composto no topo da hierarquia é chamado de *módulo rede*. Em adição, os módulos podem possuir parâmetros, cujos valores podem ser booleanos, constantes numéricas, distribuições estatísticas, textos ou arquivos XML. Os parâmetros são utilizados para passar dados para o módulo simples, ou para auxiliar na definição da topologia.

Além disso, conforme apresentado na Figura 12, módulos podem ter portas, que são os pontos de entrada e saída do módulo, com exceção do módulo rede, que não

possui portas. Estas portas podem ser conectadas por canais, desde que dentro do mesmo nível hierárquico, garantindo assim que os módulos são auto-contidos. Os canais podem possuir parâmetros de taxa de transmissão, atraso e taxa de erros.

A comunicação entre os módulos simples é efetuada pela troca de mensagens, que podem ser usadas para modelar pacotes de um protocolo de comunicação, trabalhos ou usuários de um sistema de filas, ou qualquer outro tipo de entidade móvel. Além disso, elas podem conter informações arbitrariamente complexas, dependendo do seu propósito, e são geradas e consumidas por módulos simples. As mensagens podem ser enviadas diretamente para um módulo ou podem seguir um caminho pré-determinado pelas portas e canais do modelo. Quando uma mensagem é enviada, o núcleo do OMNeT++ calcula o seu instante de chegada de acordo com o caminho percorrido, ou de acordo com o instante de agendamento. Cada mensagem recebida corresponde a um evento, e o relógio da simulação avança de acordo com os tempos de chegada dessas mensagens.

Por fim, o OMNeT++ possui um mecanismo de emissão e captura de sinais, onde um módulo ou canal emite um sinal com um valor arbitrário que irá percorrer a hierarquia de módulos até chegar ao nível mais alto, no módulo rede. Esses sinais são capturados por objetos “ouvintes”, que podem ser afixados a algum módulo da hierarquia. A Figura 13 é utilizada para exemplificar o conceito de sinais:

Figura 13 – Hierarquia de módulos no OMNeT++

As caixas da Figura 13 representam módulos do modelo, sendo que os módulos de cor cinza representam módulos simples. As setas demonstram a relação hierárquica dos módulos, ou seja, o módulo A é composto pelos módulos C e D, enquanto o módulo rede é composto pelos módulos A e B. Se o módulo C emite um sinal e o módulo D deseja capturá-lo, ele pode registrar um objeto ouvinte no módulo A.

Ainda, se o módulo D deseja capturar os eventos de toda a rede, ele pode registrar um objeto ouvinte no módulo rede.

Não é somente os módulos simples que criam objetos ouvinte; o núcleo do OMNeT++ pode adicionar um objeto ouvinte no lugar especificado pelos arquivos de configuração para capturar determinados sinais e gerar estatísticas a partir dele. As possibilidades de estatísticas são variadas, tais como gerar um histograma com os valores capturados, somar todos os valores ou criar um vetor de valores ordenado pelos instantes de emissão.

4.3.2 Execução de experimentos

O arcabouço OMNeT++ introduz três conceitos que são úteis para organizar os resultados de uma simulação: o experimento, a medição e a réplica. Um **experimento** é desenvolvido com o propósito de responder uma pergunta sobre um cenário específico, como por exemplo “Como o número de pacotes que chegam em um roteador influenciam na sua vazão?”. Para que esta pergunta seja respondida, diversas **medições** devem ser efetuadas, cada uma com um parâmetro diferente. E para eliminar qualquer viés por conta de uma seqüência específica de números aleatórios, uma medição pode ser **replicada** diversas vezes, cada vez com uma seqüência de números aleatórios diferentes.

O OMNeT++ disponibiliza um exemplo em sua instalação que modela o protocolo Aloha. Este protocolo é utilizado para compartilhar um canal de transmissão de rádio entre diversos usuários com demandas de utilização de banda relativamente baixas (ABRAMSON, 1970). Neste exemplo, o experimento *PureAlohaExperiment* tem como objetivo verificar a utilização do canal de transmissão em função do número de usuários e da taxa de geração de pacotes. A configuração do experimento é apresentada na Tabela 12.

Tabela 12 – Parâmetros do experimento Aloha do OMNeT++

Parâmetros	Valores
Número de usuários	10, 20, 25
Tempo entre pacotes, em segundos	Exponencial, com media igual a 1, 2, 3, 4, 5, 7 ou 9 segundos
Repetições	2

As medições são geradas a partir da combinação dos parâmetros, resultando em 21 medições diferentes (3 valores para o número de usuário vezes 7 valores para o tempo entre pacotes). Além disso, cada medição é repetida duas vezes. Portanto, todo o experimento necessita de 42 simulações para ser completado.

O arcabouço facilita a execução das medições em lotes, sem a necessidade de usos de ferramentas externas, além de permitir a execução paralela de diferentes medições.

4.4 ARCABOUÇO INET PARA OMNET++

O arcabouço INET é um conjunto de módulos, desenvolvido utilizando o arcabouço OMNeT++, que modelam diversos protocolos de redes cabeadas e sem fio, incluindo UDP, TCP, SCTP, IP, IPv6, Ethernet, PPP, 802.11, MPLS, OSPF, dentre muitos outros. Estes módulos são utilizados para desenvolver modelos mais complexos, dentro da arquitetura modular e hierárquica do OMNeT++, como descrito anteriormente.

Em seguida, será apresentada uma visão geral dos módulos do arcabouço INET que necessitam de um maior destaque dentro do contexto de simulação de redes *BitTorrent* são:

- **EtherSwitch** – modela um *switch ethernet*, onde o número de portas *ethernet* é configurável.
- **Router** – modela um roteador com interfaces *ethernet* ou *ppp* em quantidade configurável.
- **StandardHost** – modela um nó de rede onde as aplicações para diferentes camadas de transporte são configuráveis. Além disso, o nó possui interfaces

ethernet, ppp ou uma interface externa para conexões com sistemas reais. O número de portas é configurável.

Os módulos *EtherSwitch* e *Router* são usados para criar a topologia da rede e conectar os nós modelados pelo módulo *StandardHost*. As Figura 14, Figura 15 e Figura 16 mostram os modelos dos módulos citados, tirados diretamente do ambiente de desenvolvimento do OMNeT++.

O módulo *EtherSwitch* modela o comportamento de um switch ethernet conforme apresentado na Figura 14.

Figura 14 – Modelo de um *switch ethernet* do arcabouço INET

O módulo *mac* modela os algoritmos do enlace *ethernet* enquanto a retransmissão dos pacotes é feita pelo módulo *relayUnit*. O número de módulos *mac* é configurável, e corresponde ao número de portas do *EtherSwitch*.

O módulo *Router*, apresentado na Figura 15, possui um comportamento mais complexo, e portanto é composto por mais módulos.

Figura 15 – Modelo de um roteador do arcabouço INET

Os módulos *eth* e *ppp* modelam interfaces *ethernet* e *PPP* (*Point-to-point protocol*) respectivamente, sendo o número de interfaces configurável. O módulo *networkLayer*, por sua vez, implementa o algoritmo IP. Já o módulo *routingTable*

permite a configuração das tabelas de roteamento por meio de arquivos de texto, trabalhando em conjunto com o módulo *interfaceTables*, responsável por registrar dinamicamente as interfaces criadas. Por fim, o módulo *namTrace* gera dados sobre o funcionamento do roteador em um formato que pode ser lido por ferramentas externas, enquanto o módulo *notificationBoard* é utilizado para capturar eventos que ocorrem no roteador.

A Figura 16 mostra o módulo *StandardHost*, que modela os protocolos TCP, UDP, SCTP e IP com os módulos *tcp*, *udp*, *sctp* e *networkLayer*, respectivamente.

Figura 16 – Modelo de um nó de rede do arcabouço INET

O módulo *StandardHost* possui módulos que modelam as interfaces *ethernet*, PPP e interfaces externas que interagem com interfaces reais, sendo o número de cada interface configurável. Além disso, os módulos *tcpApp*, *udpApp* e *sctpApp* são interfaces para módulos que implementem uma determinada aplicação para TCP, UDP ou SCTP, respectivamente. Para o caso de uma simulação do *BitTorrent*, os elementos do sistema são modelados em módulos que utilizam a interface *tcpApp*, uma vez que o *BitTorrent* utiliza o TCP para efetuar a comunicação entre os elementos. Por fim o módulo *tcpdump* pode ser utilizado para capturar os pacotes que passam pelas interfaces do modelo para serem analisados por ferramentas externas.

4.5 CONSIDERAÇÕES PARCIAIS

Nesta seção, os arcabouços de simulação de redes de computadores NS-3 e OMNeT++ foram comparados a fim de escolher o mais apropriado para o desenvolvimento do simulador de *BitTorrent*. A comparação resultou na escolha do arcabouço OMNeT++ como base de desenvolvimento por conta da facilidade de configuração dos modelos desenvolvidos provida pelo ambiente, e pela sua maturidade. Foi enfatizado que a arquitetura modular do arcabouço facilita o reuso dos modelos por outros simuladores e dá grande flexibilidade para os modelos desenvolvidos.

O arcabouço OMNeT++ foi descrito em detalhes, e foi apresentado o arcabouço INET, que adiciona ao OMNeT++ modelos de um grande número de protocolos de redes. Em adição, os módulos utilizados no desenvolvimento do simulador foram explicado.

O capítulo a seguir irá apresentar o simulador de *BitTorrent* desenvolvido nesta dissertação, chamado EbitSim, apresentando em detalhes o seu funcionamento.

5 O SIMULADOR DE *BITTORRENT* *EBITSIM*

Este capítulo apresenta o simulador de *BitTorrent* desenvolvido para este trabalho, chamado *EbitSim*, acrônimo para “*Enhanced BitTorrent Simulator*”, com o propósito de ser uma ferramenta para simulação a ser utilizada pela comunidade de pesquisa.

O simulador *EbitSim* é um modelo do sistema *BitTorrent* desenvolvido utilizando o arcabouço OMNeT++, com o auxílio do arcabouço INET para modelar os detalhes de rede. Os detalhes do modelo desenvolvido serão apresentados, assim como os parâmetros e estatísticas que podem ser coletadas.

5.1 ELEMENTOS DO SIMULADOR

O modelo do simulador *EbitSim* é composto por três elementos principais: o *Peer*, o *Tracker* e o Controlador de Clientes, mostrados na Figura 17.

Figura 17 – Principais elementos do *EbitSim*

O *Peer* e o *Tracker* são nós do arcabouço INET com aplicações TCP que modelam o comportamento do cliente *BitTorrent* e da aplicação do Tracker, respectivamente. Por serem um nó INET, diversos módulos *Peer* e *Tracker* podem ser interconectados por uma rede composta por elementos de rede, como roteadores e switches, também provenientes do arcabouço INET.

O “Controlador de Clientes” foi desenvolvido como um módulo simples no OMNeT++, e sua função é determinar os instantes de entrada do *Peer* no *swarm* e os conteúdos a serem adquiridos pelos *Peers*. O seu objetivo é modelar os passos

de adquirir os arquivos *Torrent* e iniciar a aplicação *BitTorrent*. Desta maneira, é possível modelar diferentes comportamentos dos usuários, tais como diferentes processos de chegada ou entrada simultânea de um *Peer* em múltiplos *swarms*.

A aplicação do nó Tracker, denominada *TrackerApp*, é modelada como um servidor HTTP simples e que efetua as funções do *Tracker*, a relembrar, manter listas de *Peers* para diferentes *swarms* e responder a requisição de um Cliente (que é também um *Peer*) com uma lista de *Peers* aleatoriamente escolhidos. Por conta da sua simplicidade, o seu comportamento não necessita de maiores explicações.

Por sua vez, a aplicação do nó *Peer*, denominada *BitTorrentApp*, possui a maior parte da complexidade do modelo do *BitTorrent*. Este módulo é composto por módulos mais simples, responsáveis por implementar separadamente os mecanismos do *BitTorrent*. A decisão de separar a implementação em módulos foi tomada para que os mecanismos sejam mais facilmente configuráveis. Além do mais, o modelo se torna mais claro com a separação dos papéis e os módulos se tornam mais simples. A Figura 18 mostra a composição da aplicação *BitTorrentApp*:

Figura 18 – Organização do módulo BitTorrentApp

Como pode ser visto na Figura 18, a aplicação *BitTorrentApp* é um módulo composto do OMNeT++ que contém dois módulos simples, chamados “Gerente de *Swarms*” e “Cliente *BitTorrent*”. O primeiro se comunica com o *Tracker*, e o segundo

se comunica com outros *Peers*. Os objetos *Swarm* e *Thread* que residem dentro dos módulos “Gerente de *Swarms*” e do “Cliente *BitTorrent*”, respectivamente, não possuem relação com módulos do OMNeT++, e por isso são chamados de objetos C++. Por fim, existem mais dois módulos simples dentro da aplicação, chamados *Choker* e “Gerente de Conteúdo”, que são criados e destruídos dinamicamente pelo objeto C++ *Swarm*.

Nas seções a seguir, todos os componentes apresentados na Figura 18 serão discutidos com mais detalhes.

5.2 MÓDULO GERENTE DE SWARM

O objetivo do “Gerente de *Swarms*” é efetuar a comunicação entre o Cliente e um ou mais *Trackers*. Portanto, ele é responsável pela entrada e saída do Cliente em um *swarm* e pela requisição de uma nova lista de *Peers*.

A Figura 19 apresenta os passos realizados pelo Gerente de *Swarms* para entrar em um *swarm*.

Figura 19 – Processo de entrada do *Peer* no *swarm*

Como pode ser observado na Figura 19, o processo de entrada em um *swarm* é iniciado pelo “Controlador de Clientes” no passo 1, quando este manda um comando de entrada para o “Gerente de *Swarms*” contento a identificação do conteúdo e o endereço do *Tracker* com o qual se conectar. No passo 2, é enviado um anúncio de entrada ao *Tracker* especificado no passo anterior. No passo 3, a resposta do

Tracker é recebida. Finalmente, no passo 4 o objeto C++ *Swarm*, que é uma abstração do *swarm* no qual o Cliente acabou de entrar, e os módulos OMNeT++ *Choker* e “Gerente de Conteúdo” são criados no passo 5.

Múltiplos objetos *Swarm* podem ser criados simultaneamente, e como cada um possui seus próprios mecanismos de controle, o Cliente possui a capacidade de participar de múltiplos *swarms* simultaneamente. É interessante enfatizar que esta característica não foi encontrada em outros simuladores.

Depois da criação do *Swarm*, o “Gerente de *Swarms*” passa a enviar um anúncio ao *Tracker* periodicamente, para que o Cliente possa renovar a sua lista de *Peers* conectados. Além do mais, esses anúncios podem ser adiantados toda vez que o Cliente necessitar de mais *Peers*, o que ocorre quando a maior parte dos *Peers* recebidos anteriormente fechou a conexão com o Cliente. Ainda assim, a frequência de anúncios que podem ser enviadas ao *Tracker* é limitada, conforme especificado na documentação oficial.

Por fim, o Cliente possui uma probabilidade de continuar como *Seeder* até o fim da simulação, que é testada assim que termina a aquisição do conteúdo. Este papel é desempenhado pelo “Gerente de *Swarms*”, que neste caso irá enviar um anúncio de saída para o *Tracker* e apagar o objeto C++ *Swarm* correspondente. Isso faz com que todas as conexões com *Peers* do mesmo *swarm* sejam fechadas e que os módulos *Choker* e “Gerente de Conteúdo” sejam destruídos.

5.2.1 Parâmetros

Os parâmetros disponíveis para configuração do módulo “Gerente de *Swarms*” são apresentados na Tabela 13. Os valores padrão são definidos pela especificação oficial.

Tabela 13 – Parâmetros do módulo “Gerente de *Swarms*”

	<i>Parâmetro</i>	<i>Valor padrão</i>
1	Número de <i>Peers</i> na lista	50
2	Intervalo entre anúncios normais	180 s
3	Tempo mínimo entre anúncios	20 s
4	Porcentagem de permanência	

O primeiro parâmetro determina o número máximo de *Peers* que o *Tracker* pode colocar na lista retornada como resposta para um anúncio, com o valor padrão de 50 *Peers*. O segundo parâmetro define o tempo para o envio de anúncios normais de maneira automática, onde o valor padrão é um anúncio a cada 2 minutos. O terceiro parâmetro limita a quantidade de anúncios que podem ser enviados em seqüência para o *Tracker*. O valor padrão para este caso é de um anúncio a cada 20 segundos. Por fim, o quarto parâmetro define a porcentagem de permanência do Cliente no *swarm* ao término da aquisição do conteúdo, e não possui valor padrão.

5.2.2 Sinais emitidos

Os seguintes sinais são emitidos pelo módulo “Gerente de *Swarms*”:

- *SwarmManager_EnterSwarm* – Emite a identificação do *swarm* no instante de entrada do Cliente.
- *SwarmManager_LeaveSwarm* – Emite a identificação do *swarm* no instante de saída do Cliente.

Como foi explicado anteriormente na seção 4.3.1, esses sinais podem ser capturados pelos mecanismos do arcabouço OMNeT++ para gerar diferentes estatísticas. É possível capturar os sinais de entrada no nível mais alto da hierarquia do cenário, e apresentar os dados em forma de histograma, que deve representar uma distribuição de Poisson.

5.3 MÓDULO CLIENTE *BITTORRENT*

O módulo Cliente *BitTorrent* estabelece e gerencia as conexões entre o Cliente e os *Peers*. Cada conexão é gerenciada por um objeto C++ *Thread*, que é uma abstração de um subprocesso criado no instante da conexão. Cada objeto *Thread* possui as três máquinas de estado definidas na seção 2.3.3 para estabelecer e gerenciar o protocolo *PeerWire* entre o Cliente e o *Peer*. Os eventos que geram transições nas máquinas de estado do objeto *Thread* são: chegada de mensagem *PeerWire*, ação

dos módulos *Choker* e “Gerenciador de Conteúdo” do *swarm* correspondente e estouro dos temporizadores.

O modelo mais simples de processamento, utilizado por todos os simuladores que mencionaram o modelo de processamento, é quando as mensagens que chegam são somente atrasadas antes de serem processadas. Quando há somente um *Peer* conectado, este modelo pode ser representado pela Figura 20.

Figura 20 – Processamento de mensagens com atraso simples para uma conexão

As mensagens chegam pela conexão com o *Peer* e esperam em uma fila para serem processadas. Porém, quando há mais de uma conexão, este modelo funciona como múltiplas filas com múltiplos processadores, conforme apresentado na Figura 21.

Figura 21 – Processamento de mensagens com atraso simples

Como o atraso no processamento das mensagens é independente do número de conexões, o modelo funciona como se cada conexão possuísse um processador dedicado.

No *EbitSim*, por outro lado, as *Threads* compartilham o uso de um único processador, conforme apresentado pela Figura 22, modelando um sistema multiprogramado.

Figura 22 – Esquema do processamento de mensagens pelo Cliente *BitTorrent*

Observa-se na Figura 22 que os objetos *Threads* formam uma fila circular, onde cada objeto *Thread* possui uma fila de mensagens *PeerWire*. Neste modelo, o processador irá retirar uma mensagem do topo da fila de uma das *Threads*, processar esta mensagem e passar para o objeto *Thread* seguinte. Desta maneira, a cada volta na fila circular de objetos *Thread*, o processador irá processar uma mensagem de cada um dos objetos, e depois repetir o ciclo.

O tempo utilizado no atraso é gerado a partir de um histograma de tempos de processamento observados de um sistema real. Estes valores foram determinados medindo o tempo entre a chegada de uma mensagem e o envio de sua resposta em um sistema *BitTorrent* real.

5.3.1 Parâmetros

Os parâmetros que configuram o módulo “Cliente *BitTorrent*” são apresentados na Tabela 14. Os valores padrão são definidos pela especificação oficial.

Tabela 14 – Parâmetros do módulo “Cliente *BitTorrent*”

	Parâmetro	Valor padrão
1	Histograma do tempo de processamento	
2	Intervalo do cálculo da taxa de download	1 s
3	Intervalo do cálculo da taxa de upload	1 s
4	Intervalo entre Keep Alives	60 s
5	Tempo de timeout	120 s
6	Número de conexões ativas	4
7	Número de conexões passivas	6

O primeiro parâmetro possibilita a mudança do histograma utilizado para gerar os tempos de processamento. Este histograma pode ser criado a partir de ferramentas do OMNeT++. Os segundo e terceiro parâmetros definem o intervalo para o cálculo das taxas de *download* e *upload*, respectivamente, que são utilizadas na ordenação dos *Peers* pelo algoritmo de *Choking*, conforme explicado na seção 2.3.4. O quarto parâmetro define o intervalo do temporizador de *Keep Alive*. Este temporizador é reiniciado sempre que uma mensagem é enviada, e quando ocorre o seu estouro, uma mensagem de *KeepAlive* é enviada. Desta maneira, o Cliente não ficará mais do que o tempo do intervalo sem mandar uma mensagem para o *Peer*. O quinto parâmetro estabelece o máximo tempo sem receber mensagens do *Peer* que o Cliente irá tolerar. Se nenhuma mensagem chegar dentro deste tempo, a conexão é fechada. O sexto parâmetro define o número de conexões que o Cliente irá iniciar, enquanto o sétimo parâmetro define o máximo número de conexões que irá aceitar de outros *Peers*. Logo, a soma do sexto e sétimo parâmetro define o número máximo de conexões que o “Cliente *BitTorrent*” irá estabelecer.

5.3.2 Sinais emitidos

Os sinais emitidos pelo “Cliente *BitTorrent*” são apresentados abaixo.

- *BitTorrentClient_NumUnconnected* – Cada vez que o Cliente se conecta com um *Peer*, esse sinal é emitido com o número de *Peers* restantes para realizar futuras conexões.
- *BitTorrentClient_NumConnected* – Sinal emitido cada vez que ocorre uma conexão ou desconexão, indicando o número de conexões estabelecidas no momento da emissão do sinal.

Os sinais acima podem ser capturados para verificar como o número de conexões varia no *Peer* durante a simulação.

- *BitTorrentClient_ProcessingTime* – Emite o tempo de processamento de cada mensagem.

O sinal acima, quando capturado em um histograma, deve retornar o mesmo histograma utilizado para gerar os tempos de processamento.

- *BitTorrentClient_PeerWireBytesSent* – Emite o número total de *bytes* enviados com mensagens *PeerWire*.
- *BitTorrentClient_PeerWireBytesReceived* – O mesmo que o anterior, porém contabilizando as mensagens recebidas.
- *BitTorrentClient_ContentBytesSent* – Emite o número total de *bytes* de conteúdo enviados.
- *BitTorrentClient_ContentBytesReceived* – O mesmo que o anterior, porém contabilizando a recepção de conteúdo.

Os sinais acima computam a progressão do número de *bytes* enviados e recebidos pelo *Peer*, de acordo com o tipo de mensagem. Desta maneira, é possível calcular o *overhead* do protocolo *PeerWire* capturando o último valor emitido de cada sinal.

- *BitTorrentClient_*Sent* – Há um sinal para cada mensagem *PeerWire* (a saber, *BitField*, *Cancel*, *Choke*, *Unchoke*, *Handshake*, *Have*, *Interested*, *NotInterested*, *KeepAlive*, *Piece*, e *Request*) e indica o instante em que cada uma é enviada.
- *BitTorrentClient_*Received* – O mesmo que o anterior, porém para a recepção de mensagens *PeerWire*.

Esses sinais podem ser utilizados para contabilizar cada uma das mensagens do protocolo *PeerWire* enviadas ou recebidas por um *Peer* ou por todos os *Peers* do cenário, caso a captura seja efetuada no nível mais alto da hierarquia.

5.4 MÓDULO CHOKER

O módulo *Choker* implementa o algoritmo de *Choking* conforme descrito na seção 2.3.4. Como foi apresentado na Figura 17, há um *Choker* para cada swarm do qual o Cliente participa. Além disso, quando um objeto *Thread* é criado no módulo “Cliente *BitTorrent*”, ele recebe uma referência para o seu módulo *Choker* correspondente, para que a máquina de estados de upload possa efetuar chamadas do algoritmo de *Choking*. Vale lembrar que toda vez que um *Peer* que está desbloqueado muda o seu interesse no Cliente, ocorre uma rodada extra do algoritmo de *Choking*, para que a lista de *Peers* ativos possa ser atualizada.

Toda vez que ocorre uma rodada do algoritmo de *Choking*, o módulo de *Choker* acessa o “Cliente *BitTorrent*” e obtém uma lista dos *Peers* ordenados de acordo com os critérios estabelecidos anteriormente. Então ele efetua os bloqueios e desbloqueios das conexões, gerando transições nas máquinas de estados de *upload* dos *Peers* escolhidos. A Figura 23 apresenta um exemplo do funcionamento do módulo *Choker*.

Figura 23 – Exemplo do funcionamento do módulo *Choker*

No lado esquerdo da Figura 23, os *Peers* A, B, C e D estão desbloqueados e interessados no Cliente, portanto estão ocupando as vagas de *upload* do Cliente. O *Peer E* também está interessado, mas por estar bloqueado não pode enviar requisições para o Cliente. No passo 1, o *Peer B* perde o interesse no Cliente e avisa o módulo *Choker*. Como este *Peer* está ocupando uma das vagas de *upload*, o *Choker* chama uma roda do algoritmo de *Choking* no passo 2. No passo 3, o *Peer B* é bloqueado e finalmente o *Peer E*, que está interessado, é desbloqueado e passa a ocupar a vaga que antes era do *Peer B*. O lado direito da Figura 23 mostra o estado dos *Peers* ao término da rodada.

5.4.1 Parâmetros

Os parâmetros que permitem configurar o módulo *Choker* são apresentados na Tabela 15. Os valores padrão são definidos pela especificação oficial.

Tabela 15 – Parâmetros do módulo Choker

	<i>Parâmetro</i>	<i>Valor padrão</i>
1	Taxa de rodadas otimistas quando Seeder	3
2	Taxa de rodadas otimistas quando Leecher	3
3	Número de vagas otimistas	1
4	Número de vagas regulares	3
5	Intervalo entre rodadas normais	10s

O primeiro e segundo parâmetros da Tabela 15 definem quantas rodadas normais ocorrem entre cada rodada otimista quando o Cliente está no modo *Seeder* e *Leecher*, respectivamente. O terceiro e quarto parâmetros definem o número de vagas de upload otimistas e normais que o Choker irá liberar. Por fim, o quinto parâmetro define o intervalo entre rodadas normais.

5.4.2 Sinais emitidos

O módulo *Choker* não emite sinais, porém sinais podem ser adicionados para coletar o número de rodadas extras de *Choking* que são chamadas, ou alguma outra estatística desejada.

5.5 MÓDULO GERENTE DE CONTEÚDO

O “Gerente de Conteúdo” é responsável por manter uma visão atualizada da distribuição do conteúdo pelos *Peers* conectados e por verificar quais são os pedaços que o Cliente necessita para concluir a aquisição do conteúdo. Além disso, ele limita a taxa com que os pedaços são enviados para o *Peer* utilizando o algoritmo de *Token Bucket*. Essa limitação é global, portanto, se todas as vagas de upload estiverem sendo ocupadas, a taxa máxima para cada *Peer* será o limite da taxa de upload dividido pelo número de *Peers* desbloqueados.

Quando o Cliente inicia uma conexão com um *Peer*, o *BitField* inicial é armazenado pelo “Gerente de Conteúdo”, e as mensagens *Have* que o Cliente recebe deste *Peer* servem para atualizar o *BitField* do mesmo. Desta maneira, sempre que o *BitField* de um *Peer* se tornar interessante, o “Gerente de Conteúdo” efetua uma transição na máquina de estados de download deste *Peer*.

Por outro lado, toda vez que o Cliente completa a aquisição de um pedaço, o Cliente pode perder o interesse em mais de um dos *Peers* conectados, se o único pedaço que tornava esses *Peers* interessantes for aquele recém adquirido. Neste caso, o “Gerente de Conteúdo” poderá efetuar uma transição na máquina de estados de download de mais de um dos *Peers* conectados.

Em adição, o “Gerente de Conteúdo” implementa os mecanismos de *Rarest First*, *Random First* e *Strict Priority* definidos na seção 2.3.2, mas não implementa o *End-game mode* por simplicidade. Outros trabalhos também não modelam este mecanismo (LUAN; TSANG, 2006; WU; CHIUEH, 2006), pois conforme afirma (LEGOUT; URVOY-KELLER, G; MICHIARDI, 2006), este modo de seleção de pedaços não tem impacto grande no tempo de aquisição do conteúdo.

5.5.1 Parâmetros

Os parâmetros que permitem configurar o módulo “Gerente de Conteúdo” são apresentados na Tabela 16.

Tabela 16 – Parâmetros do módulo “Gerente de Conteúdo”

	<i>Parâmetro</i>	<i>Valor padrão</i>
1	Taxa de upload	
2	Número de pedaços pendentes	

O primeiro parâmetro limita a taxa de upload do Cliente e o segundo parâmetro define o número máximo de requisições esperando resposta. Este parâmetro define o encadeamento de requisições utilizado para aumentar o uso da conexão.

5.5.2 Sinais emitidos

Por fim, os sinais emitidos pelo módulo “Gerente de Conteúdo” são:

- *ContentManager_PieceDownloadTime* – Emite o tempo gasto para adquirir o último pedaço. Emitido quando um pedaço é completado.
- *ContentManager_PieceDownloaded* – Emite a identificação do pedaço no instante em que ele é completado.

Capturando o primeiro sinal em um histograma, a distribuição do tempo que cada pedaço demora para ser adquirido é adquirida. Se capturado no nível mais alto do cenário, todos os *Peers* do cenário são considerados. A captura do segundo sinal no nível mais alto da hierarquia permite determinar a distribuição de cada pedaço no decorrer da simulação.

- *ContentManager_TotalBytesDownloaded* – A cada pedaço adquirido, emite o total de bytes adquiridos até o momento.
- *ContentManager_TotalBytesUploaded* – O mesmo que o anterior, porém com a taxa de upload.

O último valor emitido dos sinais acima representa o total de bytes adquiridos e compartilhados, respectivamente. A captura deste valor pode ser utilizada para determinar a razão entre dados enviados e dados recebidos para cada *Peer*, representando o altruísmo dos *Peers* na rede.

- *ContentManager_25_percentDownloadMark* – Emite o tempo que o Cliente demorou para adquirir 25% do conteúdo, no instante que isso ocorre.
- *ContentManager_50_percentDownloadMark* – O mesmo que o anterior, porém para quando 50% do conteúdo foi completado.
- *ContentManager_75_percentDownloadMark* – O mesmo que o anterior, porém para quando 75% do conteúdo foi completado.
- *ContentManager_100_percentDownloadMark* – O mesmo que o anterior, porém para quando 100% do conteúdo foi completado.
- *ContentManager_EmittedPeerId* – Emite a identificação do Cliente no instante em que um dos 4 sinais de porcentagem de aquisição são emitidos.

Estes sinais representam a evolução da aquisição do conteúdo no *Peer*. Quando os sinais *ContentManager_<N>_percentDownloadMark* são capturados em um nível acima do *Peer*, a informação de qual *Peer* emitiu o sinal é perdida. Para recuperá-la, o último sinal permite cruzar os tempos de emissão dos sinais de porcentagem com a identificação do *Peer*. Este processo é exemplificado pelas Figura 24, Figura 25 e Figura 26.

A Figura 24 apresenta os sinais que indicam a porcentagem de conclusão da aquisição do conteúdo (*ContentManager_<N>_percentDownloadMark*) capturados no nível mais alto na hierarquia como uma série de valores temporizados.

Figura 24 – Exemplo dos sinais de porcentagem de aquisição capturados no nível mais alto da hierarquia

Pode-se notar na Figura 24 que não há nenhuma informação sobre quais foram os *Peers* que emitiram cada sinal, somente quanto tempo o *Peer* demorou para alcançar porcentagem correspondente e os instantes de emissão de cada sinal.

A Figura 25 apresenta o sinal *ContentManager_EmittedPeerId*, que é emitido ao mesmo tempo que os sinais *ContentManager_<N>_percentDownloadMark*, que possui como valor a identificação do *Peer*, também capturado no nível mais alto da hierarquia.

Figura 25 – Exemplo do sinal de emissão dos sinais de porcentagem capturado no nível mais alto da hierarquia

O cruzamento das informações dos sinais de porcentagem com o sinal *ContentManager_EmittedPeerId* é feito pelo tempo de emissão. A chance de *Peers* distintos emitirem o mesmo sinal ao mesmo tempo é baixa o suficiente para considerar que dois sinais que apresentam o mesmo tempo de emissão foram emitidos por um único *Peer*. A partir desta suposição, é possível construir a Figura 26.

Figura 26 – Exemplo do cruzamento dos sinais de porcentagem com o sinal de emissão

Na Figura 26, é apresentado o progresso de cada *Peer* na aquisição do conteúdo. O eixo das abscissas apresenta a identificação do *Peer*, enquanto o eixo das ordenadas apresenta o tempo necessário para alcançar a porcentagem do conteúdo especificada.

É importante ressaltar que esta computação pode ser realizada automaticamente pela IDE do OMNeT++, facilitando a análise das estatísticas coletadas.

5.6 CONSIDERAÇÕES PARCIAIS

Neste capítulo os elementos que compõem o simulador de *BitTorrent EbitSim* são descritos em detalhes. O uso do arcabouço INET provê um maior nível de detalhes ao *EbitSim* com modelos detalhados das camadas de rede inferiores. Além disso, o modelo de processamento de mensagens leva em consideração o compartilhamento de um único processador pelas diversas conexões, característica não encontrada em nenhum dos simuladores estudados.

Foram ressaltados os parâmetros de configuração de cada módulo, assim como os sinais emitidos para a coleção de estatísticas sobre o modelo. Novos sinais podem ser adicionados para prover maior coleção de estatísticas. Em adição, as ferramentas de análise de estatísticas fornecidas pelo OMNeT++ permite análises mais elaboradas pela composição de mais de um sinal.

A modularidade do modelo permite a troca dos mecanismos do *BitTorrent*, bastando trocar os módulos que os implementam. Essa configuração é realizada pelo OMNeT++, não sendo necessárias recompilações ou alterações no código fonte.

O próximo capítulo irá apresentar a validação do simulador EbitSim, realizada pela comparação com resultados obtidos por medição de um ambiente de testes real.

6 VALIDAÇÃO DO EBITSIM

O objetivo deste capítulo é apresentar a validação do simulador EbitSim. Para isso, os resultados da medição realizada por (SZYDLOWSKI *et al.*, 2011) serão comparados com resultados obtidos pela execução de simulações utilizando o mesmo cenário e os mesmos parâmetros.

A comparação deve mostrar que o modelo desenvolvido para o *EbitSim* é uma boa representação do sistema *BitTorrent*, capaz de fornecer resultados compatíveis com o comportamento real dos elementos do sistema. Ademais, a escalabilidade do *EbitSim* será discutida com base nos dados de execução das simulações.

6.1 MÉTODO PARA VALIDAÇÃO

Para verificar a validade do simulador EbitSim, o trabalho de medição realizado em (SZYDLOWSKI *et al.*, 2011) foi utilizado como base para comparação. Neste trabalho, o desempenho de quatro clientes populares de *BitTorrent*, a saber, *Azureus*, *μ Torrent*, *Transmission* e *Mainline* (cliente oficial), é analisado e comparado. Os experimentos são realizados em um ambiente de testes privado, e as medições são efetuadas por meio de captura de pacotes em todos os *Peers*. O método de validação consistiu em simular os mesmos cenários realizados por Szydlowski e comparar os resultados obtidos da simulação com os resultados apresentados em seu trabalho.

6.1.1 Cenários do experimento de Szydlowski

O experimento de Szydlowski foi montado em dois ambientes diferentes: um *cluster* privado de 28 máquinas conectadas por enlaces *Ethernet* de 1Gbps e um conjunto de nós no *PlanetLab*¹⁰. O experimento no *cluster* foi realizado com até 580 *Peers* conectados (diversas instâncias do cliente *BitTorrent* podem ser executadas numa

¹⁰ Rede mundial de pesquisa em redes de computadores, composta por mais de mil nós distribuídos em diversos países. Informações em <https://www.planet-lab.org/>

mesma máquina física), e por sua vez, forneceu um ambiente homogêneo para os testes. O experimento no *PlanetLab* contou com 512 nós, com um *Peer* presente por nó, e teve como objetivo testar o comportamento do *BitTorrent* em uma rede heterogênea e globalmente distribuída.

Como o *EbitSim* não possui um modelo para o comportamento da *Internet*, o experimento realizado no *PlanetLab* não é adequado para a validação do mesmo. Portanto, somente os resultados do *cluster* são utilizados.

O cenário do experimento realizado no *cluster* é apresentado na Figura 27.

Figura 27 – Cenário utilizado nos experimentos de Szydlowski

Neste cenário, há um *Tracker* e diversos *Peers* conectados a um roteador numa topologia estrela, estabelecendo a conectividade entre todos os elementos. Todos os *Peers* participam de um único *swarm*, ou seja, somente um arquivo está sendo compartilhado.

A Tabela 17 apresenta os parâmetros utilizados por Szydlowski em seu trabalho:

Tabela 17 – Parâmetros do cenário de validação

Parâmetro	Valor		
Quantidade de Leechers	Até 580		
Quantidade de Seeders	1		
Processo de chegada	Poisson com $\lambda = 10$ Peers/minuto		
Probabilidade de permanência	30% ao completar		
Limite de taxa de upload	35 kB/s	92 kB/s	512 kB/s
Tamanho do arquivo	64 MiB	128 MiB	512 MiB
Número de conexões	5, 10, 50, 100		

As duas primeiras linhas da Tabela 17 apresentam os parâmetros do experimento que determinam o tamanho do *swarm* e o comportamento dos *Peers*. Foram

utilizados até 580 *Peers* nos experimentos realizados, sendo que somente um desses foi utilizado como *Seeder* inicial.

As terceira e quarta linhas da Tabela 17 descrevem o comportamento dos *Peers* no *swarm*. Os *Peers* entram no *swarm* de acordo com um processo de *Poisson*, com o parâmetro de taxa de chegada (λ) de 10 *Peers* por minuto. O processo de *Poisson* consiste em um processo estocástico onde os eventos, neste caso a entrada de novos *Peers* no *swarm*, são independentes uns dos outros, e o intervalo entre eles é descrito pela distribuição exponencial $F(t) = 1 - e^{-\lambda t}$. Este processo modela um período no ciclo de vida de um *swarm* conhecido como *flash-crowd*, que é caracterizado por uma alta taxa de entrada de *Peers* no *swarm* recém-criado de um conteúdo popular (IZAL *et al.*, 2004). A saída dos *Peers* do *swarm* ocorre somente quando a aquisição do conteúdo é finalizada, sendo que cada *Peer* tem uma probabilidade de 30% de permanecer no *swarm*, passando a contribuir como *Seeder* para os demais *Peers*.

Por fim, as duas últimas linhas da Tabela 17 apresentam os parâmetros do *swarm*. Os *Peers* podem limitar a taxa com que fazem o *upload* de conteúdo para os *Peers* conectados, e o experimento considera três taxas diferentes escolhidas para se aproximar das taxas de upload disponíveis para usuários domésticos de internet a cabo/DSL (Digital Subscriber Line): 35 kB/s, 92 kB/s e 512 kB/s. Em relação ao tamanho dos arquivos compartilhados no *swarm*, Szydlowski afirma que a sua variação enquanto os outros parâmetros permanecem constantes não tem grande influência no desempenho do sistema, somente no tempo de aquisição do conteúdo é estendido. Portanto, o tamanho dos arquivos no experimento é utilizado como um limitador do tempo de medição e varia conforme a taxa de *upload* escolhida. Três tamanhos de arquivo são utilizados, de acordo com o valor da taxa de upload da medição: 64 MiB para a taxa de 35 kB/s, 128 MiB para a taxa de 92 kB/s e 512 MiB para a taxa de 521 kB/s. Por fim, o número de conexões realizadas por cada *Peer* é escolhida a partir de 5, que é o suficiente para fazer o algoritmo de *Choking* explicado na 2.3.4 funcionar, até um valor acima do padrão de 20 conexões.

Szydlowski denomina suas medições de acordo com a notação apresentada na Figura 28:

Figura 28 – Notação para especificação dos experimentos

Nesta notação, o *Testbed* pode ser PL (*PlanetLab*) ou CL (*cluster*), a taxa de *upload* pode ser 35, 92 e 512, e o número de conexões pode ser 5, 10, 50 e 100. Por exemplo, o experimento executado no **cluster**, com taxa de upload de **35 kB/s** e **10 conexões** é denominado **CL-35-10**.

Cada experimento é realizado 4 vezes, cada vez com um dos clientes estudados. Os clientes *Azureus*, *µTorrent* e *Transmission* foram escolhidos por Szydlowski devido à sua popularidade, sendo os programas mais utilizados por usuários do *BitTorrent*. Já o cliente *Mainline*, que foi desenvolvido diretamente pelo criador do *BitTorrent*, é utilizado como base de comparação para o desempenho dos outros clientes.

6.1.2 Métricas de comparação definidos por Szydlowski

Os diferentes clientes estudados por Szydlowski são comparados utilizando duas métricas:

1. **Perfil de mensagens PeerWire do experimento** – Esta métrica é utilizada para comparar indiretamente os algoritmos do *BitTorrent* de cada cliente por meio do perfil de mensagens enviadas. As mensagens *PeerWire* enviadas por todos os *Peers*, em todas as medições, são contabilizadas e os clientes são comparados de acordo com a quantidade enviada de cada mensagem.
2. **Tempo de aquisição do conteúdo** – Considerando que o principal objetivo do cliente *BitTorrent* é adquirir o conteúdo, esta métrica é utilizada para avaliar o quão rápido o conteúdo é adquirido. Quanto menor o tempo de aquisição, melhor é o desempenho.

6.2 REALIZAÇÃO DA VALIDAÇÃO DO *EBITSIM*

Nesta seção serão realizadas as comparações das medições realizadas pelo EbitSim com as medições realizadas por Szydlowski. Para fins de comparação, as simulações foram executadas com os mesmos parâmetros de Szydlowski. As simulações serão identificadas pela mesma notação de Szydlowski, porém utilizando a sigla ES (*EbitSim*). Por exemplo, a simulação comparável com o experimento CL-35-10 seria ES-35-10.

Os experimentos apresentados na Tabela 18 foram realizados na validação do EbitSim, e são os mesmos realizados por Szydlowski.

Tabela 18 – Experimentos realizados

<i>Experimento</i>	<i>Medida</i>	<i>Taxa de upload (kB/s)</i>	<i>Arquivo (MiB)</i>	<i>Conexões</i>
<i>Taxa de upload</i>	ES-35-5	35	64	5
	ES-92-5	92	128	5
	ES-512-5	512	512	5
<i>Número de conexões</i>	ES-35-5	35	64	5
	ES-35-10	35	64	10
	ES-35-50	35	64	50

O primeiro grupo de medições destinou-se a verificar o efeito da taxa de upload no tempo de aquisição do conteúdo, variando-se a taxa de upload em 35, 92 e 512 kB/s enquanto o número de conexões ficou fixo em 5. O segundo grupo de medições verificou o efeito do número de conexões no tempo de aquisição do conteúdo, variando este valor em 5, 10 e 50. Szydlowski repetiu cada medição 5 vezes para verificar se experimentos apresentavam resultados consistentes. As simulações também foram executadas 5 vezes cada, pelo mesmo motivo.

6.2.1 Comparação dos perfis de mensagens PeerWire

A primeira comparação realizada utiliza a métrica do perfil de mensagens PeerWire. O número de mensagens PeerWire enviadas por todos os *Peers* em todas as simulações foi somado, e os resultados são comparados aos resultados obtidos por Szydlowski na Figura 29.

Quantidade de mensagens enviadas por Cliente

Figura 29 – Comparação da quantidade de mensagens enviadas em todos os experimentos

A análise de Szydlowski exhibe as diferenças entre os clientes estudados a partir do perfil de mensagens *PeerWire* enviadas. Por exemplo, pode-se notar pela Figura 29 que a quantidade de mensagens *Cancel* enviadas pelo cliente *Transmission* é de 15%, enquanto que para os outros clientes, este valor está abaixo de 0,3%. Esta diferença ocorre pois o *Transmission* requisita o mesmo pedaço para diversos clientes diferentes simultaneamente, não respeitando o mecanismo de *strict priority* definido na 2.3.2, e portanto, precisa cancelar as requisições pendentes quando o bloco é recebido.

Para a validação do *EbitSim*, o perfil de mensagens *PeerWire* do simulador foi comparado com o perfil de mensagens de todos os clientes. Pela Figura 29, nota-se que os perfis de mensagens do *EbitSim* e do cliente *Mainline* são semelhantes, o que é um resultado esperado, uma vez que ambos implementam a especificação oficial, enquanto os outros clientes possuem modificações feitas no intuito de otimizar o *BitTorrent*.

6.2.2 Comparação dos tempos de aquisição do conteúdo

A segunda comparação foi feita utilizando a métrica do tempo de aquisição do conteúdo. A Figura 30 mostra os resultados apresentados por Szydlowski para o cenário CL-35-5 ao lado dos resultados para a simulação ES-35-5:

Figura 30 – Função densidade de probabilidade do tempo de download para o cenário CL-35-5 e a simulação ES-35-5

As curvas apresentadas na Figura 30 mostram a probabilidade acumulada empírica do tempo de aquisição do conteúdo para os quatro clientes estudados e para o *EbitSim*. Uma curva neste gráfico apresenta a probabilidade de um *Peer* completar a aquisição do conteúdo antes de um instante qualquer no eixo das abscissas.

Espera-se, novamente, que o desempenho do *EbitSim* seja similar ao desempenho do cliente *Mainline*. Nota-se, na Figura 30, que o *EbitSim* e o cliente *Mainline* apresentam curvas próximas uma das outras, demonstrando que o modelo de simulação segue a mesma tendência do sistema real.

A Figura 31 apresenta os resultados obtidos da simulação da medição ES-92-5 juntamente com os resultados da medição CL-92-5.

Figura 31 – Função densidade de probabilidade do tempo de download para o cenário CL-92-5 e a simulação ES-92-5

Como pode ser observado na Figura 31, o *EbitSim* apresenta um desempenho melhor quando comparado com o cliente *Mainline*, pois a sua curva está localizada mais à esquerda da curva do *Mainline*. Contudo, a curva de densidade de probabilidade do tempo de aquisição do conteúdo do *EbitSim* apresenta a mesma tendência das curvas obtidas pelos experimentos, o que é uma boa indicação de que o modelo do *EbitSim* gera resultados compatíveis com o sistema real.

As diferenças podem ser justificadas devido à especificação deixar pontos não definidos, que ficam a cargo do desenvolvedor do cliente *BitTorrent* decidir, como por exemplo o número de conexões ativamente iniciadas pelo Cliente. Apesar disso, os resultados apresentados possibilitam chegar às mesmas conclusões que as medições, validando o simulador *EbitSim*.

A validação realizada mostra que o simulador *EbitSim* apresenta resultados compatíveis com o cenário apresentado, o que fornece indícios do seu funcionamento para cenários com parâmetros diferentes (FORTIER; MICHEL, 2003; JAIN, 1991).

6.3 DADOS SOBRE A EXECUÇÃO DAS SIMULAÇÕES

O equipamento utilizado para executar as simulações do EbitSim foi um computador com processador Intel Xeon E5520 @ 2.27GHz de 8 núcleos, 8GB de memória RAM, utilizando o sistema operacional Ubuntu Maverick 10.10. A versão do OMNeT++ utilizada foi a 4.2 e a versão do INET foi 20111128. Os diversos cenários foram executados em paralelo e de maneira automatizada com o auxílio do arcabouço OMNeT++. Desta maneira, o tempo total de simulação, que é o tempo necessário para executar todas as medições do experimento, foi diminuído.

Serão apresentados os dados sobre o tempo de simulação, que é o tempo necessário para completar a execução de uma medição, e o tempo simulado, que é o tempo do modelo, para cada uma das simulações executadas. Essas informações são apresentadas com o intuito de demonstrar a escalabilidade do simulador de acordo com a complexidade dos cenários simulados.

6.3.1 Relação entre o tempo de simulação e os parâmetros da medição

Os tempos médios das medições do experimento da taxa de upload, calculados como a média dos tempos das repetições, são mostrados na Tabela 20.

Tabela 19 – Resultados para o tempo médio de simulação do experimento da taxa de upload

<i>Medição</i>	<i>Tempo de simulação médio (s)</i>	<i>Tempo de simulação médio (h)</i>	<i>Desvio Padrão (s)</i>
<i>ES-35-5</i>	4858,795	1h 20min 58,795s	62,1806
<i>ES-92-5</i>	8643,554	2h 24min 03,554s	76,7522
<i>ES-512-5</i>	33466,115	9h 17min 46,115s	347,8078

Pode-se ver na Tabela 20 que quanto maior a taxa de *upload*, e por consequência o tamanho do arquivo, maior é o tempo de simulação da medição. Este resultado é esperado, pois quanto maior o arquivo simulado, maior for o número de pedaços sendo compartilhados entre os diversos *Peers*, e portanto, maior a quantidade de eventos simulados. A Figura 32 apresenta a relação entre os tempos de simulação e o tamanho dos arquivos simulados.

Figura 32 – Variação do tempo de simulação de acordo com o tamanho do arquivo

Na Figura 32, estão representados os tempos de simulação de todas as medições e repetições executadas. A reta representa a regressão linear do tempo de simulação pelo tamanho do arquivo, descrita pela fórmula $y = 614,99 + 64,04 \cdot x$ e calculada pelo método dos mínimos quadrados. O valor de R^2 próximo de 1 mostra que o modelo linear apresenta uma boa estimativa da relação entre o tempo de simulação e o tamanho do arquivo simulado.

Para cada 1 MiB adicional simulado, o tempo de simulação aumenta em 64 segundos. Com este modelo linear, é possível estimar o tempo de simulação do cenário para outros tamanhos de arquivo. Por exemplo, se o tamanho do arquivo simulado for de 1024 MiB (ou seja, 1 GiB), o tempo de simulação será próximo de $y = 614,99 + 64,04 \cdot 1024$, que é igual a 66191,95 segundos ou aproximadamente 18 horas e 23 minutos.

Tabela 20 – Resultados para o tempo médio de simulação do experimento do número de conexões

<i>Medição</i>	<i>Tempo de simulação médio (s)</i>	<i>Tempo de simulação médio (h)</i>	<i>Desvio Padrão (s)</i>
<i>ES-35-5</i>	4051,426	1h 07min 31,426s	18,1785
<i>ES-35-10</i>	4163,408	1h 09min 23,408s	28,8064
<i>ES-35-50</i>	5550,045	1h 32min 30,045s	68,9255

Na Tabela 20, nota-se que quanto maior o número de conexões estabelecidas, maior o tempo para conclusão da simulação. Esta relação é apresentada na Figura 33.

Figura 33 – Variação do tempo de simulação de acordo com o número de conexões

Utilizando o mesmo método de regressão aplicado nos pontos da Figura 32, o modelo linear dos pontos da Figura 33 é calculado como $y = 3855,41 + 33,83 \cdot x$. O valor de R^2 próximo de 1 mostra que o modelo se adequa bem aos dados representados. Para cada conexão adicional simulada, o tempo de simulação aumenta em aproximadamente 34 segundos. Se fossem simuladas 100 conexões, o tempo de simulação seria de 7238,41 segundos, ou aproximadamente 2 horas.

Mesmo com tempos de simulação aumentando com a complexidade do cenário, a coleta de resultados utilizando a simulação no hardware especificado é muito mais rápida do que a medição em *testbed*. Por exemplo, no trabalho de Szydlowski, os experimentos foram executados em um único cluster, portanto, as medições precisam ser efetuadas seqüencialmente. Considere que cada medição realizada demora cerca de 3 horas, no melhor dos casos. Desconsiderando o tempo necessário para a configuração do ambiente de testes, um experimento com 3 medições replicada 10 vezes, necessitaria de 90 horas para ser finalizado.

Por outro lado, as medições do EbitSim podem ser executadas paralelamente em um único computador multicore, de maneira automatizada pelo arcabouço OMNeT++. A Figura 34 apresenta um exemplo onde as medições do experimento

de taxa de upload, replicadas 10 vezes, são executadas em 6 processadores simultaneamente.

Figura 34 – Encadeamento do experimento de taxa de upload com 10 repetições

A Figura 34 mostra como as simulações são encadeadas se os cenários mais complexos forem executados primeiro. Os tempos de execução são aproximados dos valores da Tabela 20 e aproximados para 20 minutos, por simplificação. Os números dentro das caixas indicam a ordem de execução das simulações. Nota-se que para executar as 30 medições são necessários 23 horas e 20 minutos, contra 90 horas de execução das medições no ambiente real. Portanto, neste exemplo, o tempo de simulação é quase 4 vezes menor que o tempo de experimentação. Importante ressaltar que as medições são configuradas manualmente, enquanto as simulações são executadas de maneira automatizada.

6.3.2 Relação entre o tempo simulado e os parâmetros da medição

A Tabela 21 apresenta a relação entre o tempo simulado médio e o tempo de simulação médio do experimento da taxa de upload, calculados como a média das repetições de cada medição:

Tabela 21 – Razão entre tempo simulado médio e o tempo de simulação médio para o experimento da taxa de upload

<i>Medição</i>	<i>Tempo simulado médio (s)</i>	<i>Tempo simulado médio (h)</i>	<i>Desvio Padrão (s)</i>	<i>Tempo de simulação médio (s)</i>	<i>simulado/simulação</i>
<i>ES-35-5</i>	4694,073	1h 18min 14,073s	80,4068	1h 20min 58,795s	0,966
<i>ES-92-5</i>	4094,116	1h 08min 14,116s	49,6913	2h 24min 03,554s	0,474
<i>ES-512-5</i>	3556,680	59min 16,680s	80,7175	9h 17min 46,115s	0,106

O tamanho do arquivo tem um efeito bem acentuado sobre a razão entre o tempo simulado e o tempo de simulação. Isso ocorre pois quanto maior for o número de pedaços, maior é o número de eventos simulados. Como o número de eventos simulados por segundo não muda, então o tempo de simulação cresce juntamente com o número de eventos.

A Tabela 22 apresenta a relação entre o tempo simulado médio e o tempo de simulação médio do experimento da taxa de upload, calculados como a média das repetições de cada medição:

Tabela 22 – Razão entre tempo simulado médio e o tempo de simulação médio para o experimento do número de conexões

<i>Medição</i>	<i>Tempo simulado médio (s)</i>	<i>Tempo simulado médio (h)</i>	<i>Desvio Padrão (s)</i>	<i>Tempo de simulação médio (s)</i>	<i>simulado/simulação</i>
<i>ES-35-5</i>	5113,940	1h 25min 13,940s	75,6734	1h 07min 31,426s	1,262
<i>ES-35-10</i>	4948,768	1h 22min 28,768s	82,7944	1h 09min 23,408s	1,189
<i>ES-35-50</i>	4144,087	1h 08min 34,087s	77,3361	1h 32min 30,045s	0,747

O mesmo comportamento apresentado com o tamanho do arquivo se repete com o número de conexões, porém de forma mais branda.

6.4 CONSIDERAÇÕES PARCIAIS

Neste capítulo foi apresentada a validação do simulador *EbitSim*, realizada com base no trabalho realizado por Szydlowski *et al.* (2011). Foi demonstrado que o simulador *EbitSim* apresenta resultados que são compatíveis com o desempenho cliente de *BitTorrent* oficial e possibilitam chegar às mesmas conclusões que a medição. Diferenças nos valores da simulação e da medição são resultado de

diferenças no modelo do *BitTorrent* por conta da especificação oficial, que deixa aspectos do funcionamento do sistema a cargo do desenvolvedor, e por conta de simplificações no modelo desenvolvido.

Em adição a estes resultados, foram coletados dados sobre a execução das simulações, considerando a relação entre o tempo de simulação (tempo gasto para concluir uma medição) com o tamanho do arquivo e com o número de conexões. Foi inferido que tanto o tamanho do arquivo quanto o número de conexões possuem uma relação linear com o tempo de simulação. Com essa informação, é possível prever o tempo de simulação necessário para executar uma simulação, utilizando a mesma configuração de hardware.

A execução das simulações não exigiu o uso de um equipamento especializado, sendo realizada em um único computador de configurações convencionais¹¹. As diversas simulações foram realizadas em paralelo, de maneira automatizada pelo arcabouço OMNeT++, para diminuir o tempo de experimentação. Por outro lado, o experimento realizado por Szydlowski necessitou de 32 computadores conectados em uma rede privada, e as medições precisaram ser configuradas manualmente e realizadas em série.

Ainda que no *EbitSim* os cenários mais complexos tenham necessitado de por volta de 9 horas para serem executados, o tempo total de experimentação foi cerca de 4 vezes menor que medições realizadas por Szydlowski, sem considerar o tempo necessário para a configuração do ambiente de testes. Portanto, o uso do *EbitSim* necessita de menos recursos para executar os mesmos experimentos que um ambiente de testes dedicado, sendo uma ferramenta acessível para a comunidade de pesquisa.

¹¹ Processador com 8 núcleos e 8GB de memória RAM

7 CONSIDERAÇÕES FINAIS

Este capítulo apresenta as considerações finais sobre esta dissertação, que resultou no desenvolvimento do simulador de *BitTorrent EbitSim*. O *EbitSim* foi desenvolvido para suprir a necessidade do ambiente de pesquisas sobre *BitTorrent* por uma ferramenta padronizada de simulação. Inicialmente, será apresentada uma análise geral deste trabalho, ressaltando as contribuições desta dissertação. A seguir, serão discutidas as possibilidades de trabalhos futuros que podem ser realizados a partir deste trabalho. Por fim, serão apresentadas as publicações no qual o autor participou direta ou indiretamente no decorrer do programa de mestrado.

7.1 CONTRIBUIÇÕES

Conforme discutido na introdução desta dissertação, o sistema *BitTorrent* é estudado ativamente pela comunidade de pesquisa por conta de sua popularidade como ferramenta de distribuição de conteúdo. O *BitTorrent* foi desenvolvido como um mecanismo para distribuição de arquivos de grande tamanho (centenas de MB) para um grupo de usuários, porém pesquisas demonstraram a sua aplicabilidade em áreas fora deste escopo, como por exemplo na distribuição de vídeos ao vivo.

A pesquisa dos trabalhos relacionados revelou a utilidade do emprego do método de simulação na avaliação do impacto de modificações do sistema *BitTorrent*, assim como em pesquisas que envolvam uma grande quantidade de *Peers*. Porém, também ficou evidente que não há um simulador adequado que possui capacidade de configuração dos mecanismos e ferramentas de coleta de estatísticas. Este problema levou a um grande número de trabalhos realizados desenvolverem os seus próprios simuladores, o que dificulta ou impossibilita a reprodução dos experimentos e a verificação dos resultados.

A principal contribuição deste trabalho foi o simulador de *BitTorrent EbitSim*, que cumpre o objetivo desta dissertação, que é desenvolver um simulador de *BitTorrent* que permite a alteração dos mecanismos e a configuração dos parâmetros do sistema.

O *EbitSim*, desenvolvido sobre o arcabouço OMNeT++, apresenta as características abaixo:

- Modelo detalhado dos mecanismos do *BitTorrent*, desenvolvidos a partir de uma investigação do funcionamento do sistema a partir da especificação oficial, trabalhos relacionados e discussões com desenvolvedores de clientes *BitTorrent*.
- Facilidade em adicionar novas funcionalidades aos modelos devido à natureza modular do simulador.
- Facilidade em configurar os parâmetros da simulação e definir as topologias dos experimentos por meio das ferramentas fornecidas pelo arcabouço INET.
- Ferramentas de coleta e análise de estatística que permite a configuração e o processamento dos dados coletados.
- Modelo de rede detalhado por conta do uso do dos modelos das camadas de rede inferiores do arcabouço INET.

Essas características permitem o uso do *EbitSim* em variados cenários, além de facilitar a configuração dos parâmetros e a mudança dos mecanismos do *BitTorrent*. Além destas características, as seguintes características não foram encontradas em trabalhos relacionados, mas foram desenvolvidas no *EbitSim*:

- O modelo de processamento dos Clientes contempla o efeito de múltiplas conexões. Outros modelos supõem que o número de conexões simultâneas não afeta o tempo de processamento.
- É possível simular o acesso de um *Peer* a múltiplos *swarms* de maneira simultânea.

O simulador foi validado por meio da comparação com um trabalho de medição realizado em um ambiente de testes privado com mais de 500 *Peers*. Os resultados mostraram que o *EbitSim* apresenta resultados compatíveis com o sistema real e, portanto, pode ser utilizado para auxiliar a compreensão do funcionamento do *BitTorrent* em cenários simulados.

O *EbitSim* foi disponibilizado no *GitHub*¹², um repositório de software livre, sob licença *Creative Commons*¹³, permitindo que outros possam compartilhar e alterar o

¹² Em <https://github.com/pedromanoel/EbitSim>

simulador desde o autor seja citado e que seja utilizado para fins não comerciais. Desta maneira, espera-se que outros pesquisadores possam contribuir para o código da simulação, e que esta ferramenta seja utilizada para realizar outros trabalhos na área de redes P2P *BitTorrent*. É importante ressaltar que logo após a disponibilização do *EbitSim*, o autor já foi contatado por outros pesquisadores interessados em utilizar o simulador em suas pesquisas.

A segunda contribuição deste trabalho foi uma descrição detalhada dos componentes dinâmicos e da interação entre os *Peers* do sistema *BitTorrent*, algo que não foi encontrado nem na especificação oficial nem em trabalhos correlatos. O correto entendimento dos mecanismos do *BitTorrent* contribuiu para um modelo mais detalhado, requisito necessário para o correto desenvolvimento do simulador e a geração de resultados corretos pela ferramenta.

7.2 TRABALHOS FUTUROS

Como o simulador *EbitSim* tem o seu código disponibilizado publicamente, modificações no modelo feitas por outros pesquisadores podem ser adicionadas ao simulador, incrementando o leque de configurações que podem ser realizadas. Uma lista de possíveis trabalhos futuros que adicionem funcionalidade ao *EbitSim* é apresentada abaixo:

- **Implementar modelos diferentes do comportamento de entrada e saída dos *Peers* no *swarm***
Ao alterar os processos de entrada e saída dos *Peers* do *swarm*, é possível estudar como diferentes comportamentos dos usuários afetam o desempenho do *BitTorrent*. Pode-se modelar, por exemplo, *Peers* que deixam o *swarm* antes de completar a aquisição do conteúdo, ou *Peers* que participam de mais de um *swarm* simultaneamente.
- **Implementar diferentes versões dos mecanismos do *BitTorrent***

¹³ Texto da licença disponível em http://creativecommons.org/licenses/by-nc-sa/3.0/deed.pt_BR

Permite estudar como os mecanismos influenciam o desempenho do *BitTorrent*. Por exemplo, ao invés de selecionar os pedaços mais raros primeiro, pode-se selecionar primeiro os pedaços do começo do arquivo.

- **Implementar um modelo do *Tracker* com maior inteligência**

O *Tracker* pode controlar o formato da rede sobreposta de *Peers*, retornando para o Cliente uma lista com *Peers* selecionados de acordo com algum critério diferente do aleatório. Por exemplo, os *Peers* retornados podem ser os mais próximos do Cliente, ou os menos retornados.

- **Implementar um modelo de rede simplificado para aumentar a escalabilidade do sistema**

A rede pode ser modelada somente no nível de aplicação, e o seu desempenho em comparação com o modelo de rede detalhado pode ser caracterizado a fim de compreender o compromisso entre nível de detalhamento e exatidão dos resultados.

- **Implementar um modelo da *Internet***

O modelo da rede pode levar em consideração a topologia e a latência da *Internet*, como por exemplo o trabalho realizado em (ZHANG, B. *et al.*, 2010), que apresenta um modelo estatístico do atraso entre duas redes de borda na *Internet*.

7.3 PUBLICAÇÕES

Durante o programa de mestrado, o autor participou da escrita de dois artigos publicados em conferências internacionais. Vale ressaltar que os resultados apresentados nesta dissertação serão compilados em um artigo a ser publicado em conferencia internacional.

Os seguintes artigos publicados tiveram a participação do autor, sendo o primeiro diretamente relacionado ao *EbitSim* e a esta dissertação:

EVANGELISTA, P.; AMARAL, M.; MIERS, C.; GOYA, W.; SIMPLICIO, M.; CARVALHO, T.; SOUZA, V. **EbitSim: An Enhanced *BitTorrent* Simulation Using OMNeT++ 4**. Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium, pags.437-440. Cingapura/Cingapura 25-27 de Julho de 2011.

MIERS, CHARLES; BARROS, MARCEL; SIMPLÍCIO, MARCOS; GONZALEZ, NELSON; EVANGELISTA, PEDRO; GOYA, WALTER; CARVALHO, TEREZA et al. **Using Trade Wind to sail in the clouds**. Proceedings of Parallel and Distributed Computing and Systems (PDCS), 8 pags. Dallas/EUA: 14-16 de Dezembro de 2011.

REFERÊNCIAS BIBLIOGRÁFICAS

- ABRAMSON, N. **THE ALOHA SYSTEM: another alternative for computer communications**. Proceedings of the November 17-19, 1970, fall joint computer conference. **Anais...**, AFIPS '70 (Fall). New York, NY, USA: ACM. , 1970
- AGUILAR, A.; EGE, R. K.; MAKKI, K.; BUSTOS, R. **Enabling Peer Cooperation in Private Local Area Networks using BitTorrent**. 12th IEEE Symposium on Computers and Communications, 2007. ISCC 2007. **Anais...** [S.l.]: IEEE. , 1 jul 2007
- BAKKER, A.; PETROCCO, R.; DALE, M.; GERBER, J. *et al.* **Online Video Using BitTorrent and HTML5 Applied to Wikipedia**. 2010 IEEE Tenth International Conference on Peer-to-Peer Computing (P2P). **Anais...** [S.l.]: IEEE. , 25 ago 2010
- BAYOUMY, A. Z.; SALAMA, M. A.; ZAYED, H. H. Adaptive Sliding Piece Selection Window for BitTorrent Systems. **Advances in Multimedia-An International Journal (AMIJ)**, v. 2, n. 1, p. 18, 2011.
- BHARAMBE, A. R.; HERLEY, C.; PADMANABHAN, V. N. **Analyzing and Improving a BitTorrent Networks Performance Mechanisms**. INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings. **Anais...** [S.l.: s.n.]. , abr 2006
- BINDAL, R.; CAO, P.; CHAN, W.; MEDVED, J. *et al.* **Improving Traffic Locality in BitTorrent via Biased Neighbor Selection**. 26th IEEE International Conference on Distributed Computing Systems, 2006. ICDCS 2006. **Anais...** [S.l.: s.n.]. , 2006
- BitTorrent Mainline Client**. Disponível em:
<<https://launchpad.net/bittorrent/main/4.0.0/+download/BitTorrent-4.0.0.tar.gz>>.
Acesso em: 30 jul. 2012.
- BitTorrent Simulator - Microsoft Research**. Disponível em:
<<http://research.microsoft.com/en-us/downloads/20d68689-9a8d-44c0-80cd-66dfa4b0504b/>>. Acesso em: 13 ago. 2012.
- BitTorrent Specification - TheoryOrg**. Disponível em:
<<http://wiki.theory.org/BitTorrentSpecification>>. Acesso em: 30 jul. 2012.
- BUSTOS, R.; AGUILAR, A.; MAKKI, K.; EGE, R. K. **Multicast-P2P content distribution in large-scale enterprise networks**. IEEE Symposium on Computers and Communications, 2008. ISCC 2008. **Anais...** [S.l.: s.n.]. , jul 2008
- CHEN TIAN; XUE LIU; HONGBO JIANG; WENYU LIU; YI WANG. **Improving BitTorrent Traffic Performance by Exploiting Geographic Locality**. IEEE Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. **Anais...** [S.l.]: IEEE. , 30 dez 2008
- CHEN, X.; JARVIS, S. A. **Analysing BitTorrent's Seeding Strategies**. International Conference on Computational Science and Engineering, 2009. CSE '09. **Anais...** [S.l.: s.n.]. , ago 2009

COHEN, B. **Incentives Build Robustness in BitTorrent**. Workshop on Economics of Peer-to-Peer Systems. **Anais...** Berkeley, CA, USA: [s.n.]. Disponível em: <<http://www.bittorrent.org/bittorrentecon.pdf>>. Acesso em: 10 jun. 2009. , 5 jun 2003

COHEN, B. **BitTorrent Official Specification**. Disponível em: <http://www.bittorrent.org/beps/bep_0003.html>. Acesso em: 5 nov. 2011.

FORTIER, P. L.; MICHEL, H. E. **Computer Systems Performance Evaluation and Prediction**. 1st. ed. [S.l.]: Digital Press, 2003.

FRIOLI, F.; PEDROLLI, M. **A BitTorrent module for Peersim**. Disponível em: <<http://peersim.sourceforge.net/code/bittorrent.tar.gz>>. Acesso em: 24 ago. 2012.

GPS - General Purpose Simulator for P2P network. Disponível em: <<http://www.cs.binghamton.edu/~wyang/gps/>>. Acesso em: 13 ago. 2012.

GUMMADI, K. P.; SAROIU, S.; GRIBBLE, S. D. **King: estimating latency between arbitrary internet end hosts**. Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment. **Anais...**, IMW '02. New York, NY, USA: ACM. . Acesso em: 15 ago. 2012. , 2002

GUO, L.; CHEN, S.; XIAO, Z.; TAN, E. *et al.* A performance study of BitTorrent-like peer-to-peer systems. **Selected Areas in Communications, IEEE Journal on**, v. 25, n. 1, p. 169, 155, 2007.

HOSSEINI, F.; KHONSARI, A.; TALEBI, M. S.; MORAVEJIZ, A. **Optimizing shared upload bandwidth in BitTorrent-like peer-to-peer networks**. 14th Asia-Pacific Conference on Communications, 2008. APCC 2008. **Anais...** [S.l.]: IEEE. , 14 out 2008

ILIE, D.; ERMAN, D.; POPESCU, A.; NILSSON, A. A. **Traffic measurements of P2P systems**. Swedish National on Computer Networking Workshop (SNCNW04). **Anais...** [S.l.: s.n.]. . Acesso em: 16 ago. 2012. , 2004

INRIA. **Understanding BitTorrent: An Experimental Perspective**. . [S.l.: s.n.]. Disponível em: <<http://hal.inria.fr/inria-00000156/en>>. Acesso em: 9 nov. 2011. , 2005

IPOQUE. **Ipoque Internet Study 2006**. Disponível em: <www.ipoque.com/sites/default/files/mediafiles/documents/p2p-survey-2006.pdf>. Acesso em: 4 nov. 2011.

IPOQUE. **Ipoque Internet Study 2007**. Disponível em: <www.ipoque.com/sites/default/files/mediafiles/documents/internet-study-2007.pdf>. Acesso em: 4 nov. 2011.

IPOQUE. **Ipoque Internet Study 2008/2009**. Disponível em: <www.ipoque.com/sites/default/files/mediafiles/documents/internet-study-2008-2009.pdf>. Acesso em: 4 nov. 2011.

IPOQUE. **Internet Observatory 2011**. Disponível em: <<http://www.internetobservatory.net/>>. Acesso em: 6 nov. 2011.

IPOQUE. **Ipoque Internet Studies**. Disponível em:
<<http://www.ipoque.com/en/resources/internet-studies>>. Acesso em: 4 nov. 2011.

IZAL, M.; URVOY-KELLER, GUILLAUME; BIRSACK, E.; FELBER, P. *et al.*
Dissecting BitTorrent: Five Months in a Torrent's Lifetime. In: BARAKAT, C.; PRATT,
I. (Eds.). **Passive and Active Network Measurement**. Lecture Notes in Computer
Science. [S.l.]: Springer Berlin / Heidelberg, 2004. v. 3015p. 1-11.

JAIN, R. **The Art of Computer Systems Performance Analysis: Techniques for
Experimental Design, Measurement, Simulation, and Modeling**. [S.l.]: Wiley,
1991.

JIA, A. L.; D'ACUNTO, L.; MEULPOLDER, M.; POUWELSE, J. A.; EPEMA, D. H. J.
BitTorrent's dilemma: Enhancing reciprocity or reducing inequity. 2011 IEEE
Consumer Communications and Networking Conference (CCNC). **Anais...** [S.l.: s.n.].
, jan 2011

JIAN, L.; MACKIE-MASON, J. K. **Why share in peer-to-peer networks?**
Proceedings of the 10th international conference on Electronic commerce. **Anais...**,
ICEC '08. New York, NY, USA: ACM. . Acesso em: 9 nov. 2011. , 2008

KATSAROS, K.; KEMERLIS, V. P.; STAIS, C.; XYLOMENOS, G. **A BitTorrent
module for the OMNeT++ simulator**. IEEE International Symposium on Modeling,
Analysis Simulation of Computer and Telecommunication Systems, 2009. MASCOTS
'09. **Anais...** [S.l.: s.n.]. , set 2009

KATSAROS, KONSTANTINOS. **BitTorrent for the OMNeT++ Simulation Platform**.
Disponível em:
<http://mm.aueb.gr/index.php?option=com_content&view=article&id=53:bittorrent&catid=49:software&Itemid=94>. Acesso em: 13 ago. 2012.

KE XU; YAHUI YANG; TAO CHEN. **Improving BitTorrent Network's Performance
via Deploying Helpers**. IEEE/IFIP International Conference on Embedded and
Ubiquitous Computing, 2008. EUC '08. **Anais...** [S.l.]: IEEE. , 17 dez 2008

KONRATH, M. A.; BARCELLOS, M. P.; MANSILHA, R. B. **Attacking a Swarm with
a Band of Liars: evaluating the impact of attacks on BitTorrent**. Seventh IEEE
International Conference on Peer-to-Peer Computing, 2007. P2P 2007. **Anais...**
[S.l.]: IEEE. , 2 set 2007

KRYCZKA, M.; CUEVAS, R.; GUERRERO, C.; AZCORRA, A.; CUEVAS, A.
Measuring the bittorrent ecosystem: Techniques, tips, and tricks. **IEEE
Communications Magazine**, v. 49, n. 9, p. 144-152, set 2011.

LEGOUT, A.; LIOGKAS, N.; KOHLER, E.; ZHANG, L. **Clustering and sharing
incentives in BitTorrent systems**. Proceedings of the 2007 ACM SIGMETRICS
international conference on Measurement and modeling of computer systems.
Anais..., SIGMETRICS '07. New York, NY, USA: ACM. . Acesso em: 10 nov. 2011. ,
2007

LEGOUT, A.; URVOY-KELLER, G; MICHIARDI, P. **Rarest first and choke algorithms are enough**. Proceedings of the 6th ACM SIGCOMM conference on Internet measurement. **Anais...** [S.l.]: ACM. , 2006

LEIBOWITZ, N.; RIPEANU, M.; WIERZBICKI, A. **Deconstructing the Kazaa Network**. Proceedings of the The Third IEEE Workshop on Internet Applications. **Anais...**, WIAPP '03. Washington, DC, USA: IEEE Computer Society. . Acesso em: 8 nov. 2011. , 2003

LI, J. **Peer-to-peer multimedia applications**. Proceedings of the 14th annual ACM international conference on Multimedia. **Anais...**, MULTIMEDIA '06. New York, NY, USA: ACM. . Acesso em: 8 nov. 2011. , 2006

LI, P.; JOHN, C. S. L.; XU, Y. A stochastic model for BitTorrent-like systems. **ACM SIGMETRICS Performance Evaluation Review**, v. 36, n. 2, p. 118, 116, 2008.

LIAO, W.; PAPADOPOULOS, F.; PSOUNIS, K. Performance analysis of bittorrent-like systems in heterogeneous networks. **Performance Evaluation**, v. 64, n. 9, p. 876-891, 2007.

LUAN, H.; TSANG, D. **A simulation study of block management in BitTorrent**. InfoScale '06: Proceedings of the 1st international conference on Scalable information systems. **Anais...** Hong Kong: ACM. , 2006

MONTRESOR, A.; JELASITY, M. **PeerSim: A Scalable P2P Simulator**. Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09). **Anais...** Seattle, WA: [s.n.]. , set 2009

NAICKEN, S.; LIVINGSTON, B.; BASU, A.; RODHETBHAI, S. *et al.* The state of peer-to-peer simulators and simulations. **SIGCOMM Comput. Commun. Rev.**, v. 37, n. 2, p. 98, 95, 2007.

ORAM, A. **Peer-to-Peer : Harnessing the Power of Disruptive Technologies**. 1st. ed. [S.l.]: O'Reilly Media, 2001.

PARVEZ, K. N.; WILLIAMSON, C.; MAHANTI, A.; CARLSSON, N. Insights on Media Streaming Progress Using BitTorrent-Like Protocols for On-Demand Streaming. **IEEE/ACM Transactions on Networking**, v. 20, n. 3, p. 637 -650, jun 2012.

POUWELSE, J. A.; GARBACKI, P.; EPEMA, D. H. J.; SIPS, H. J. **The Bittorrent P2P File-Sharing System: Measurements and Analysis**. 4TH INTERNATIONAL WORKSHOP ON PEER-TO-PEER SYSTEMS (IPTPS). **Anais...** [S.l.: s.n.]. , 2005

POUWELSE, J.; GARBACKI, P.; EPEMA, D.; SIPS, H. Pirates and Samaritans: A decade of measurements on peer production and their implications for net neutrality and copyright. **Telecommunications Policy**, v. 32, n. 11, p. 701-712, dez 2008.

QIU, D.; SRIKANT, R. **Modeling and performance analysis of BitTorrent-like peer-to-peer networks**. SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications. **Anais...** Portland, Oregon, USA: ACM. , 2004

RASTI, A. H.; REJAIE, R. **Understanding Peer-level Performance in BitTorrent: A Measurement Study**. Proceedings of 16th International Conference on Computer Communications and Networks, 2007. ICCCN 2007. **Anais...** [S.l.]: IEEE. , 13 ago 2007

SAAD, R.; SERHROUCHNI, A.; BEGLICHE, Y.; CHEN, K. **Evaluating Forward Error Correction performance in BitTorrent protocol**. 2010 IEEE 35th Conference on Local Computer Networks (LCN). **Anais...** [S.l.]: IEEE. , 10 out 2010

SANDVINE. **Global Interent Phenomena Report: 2010**. Disponível em: <<http://www.sandvine.com/downloads/documents/2010%20Global%20Internet%20Phenomena%20Report.pdf>>. Acesso em: 6 nov. 2011.

SANDVINE. **Global Interent Phenomena Report: Fall 2011**. Disponível em: <http://www.sandvine.com/downloads/documents/10-26-2011_phenomena/Sandvine%20Global%20Internet%20Phenomena%20Report%20-%20Fall%202011.pdf>. Acesso em: 6 nov. 2011.

SHAH, P.; PARIS, J.-F. **Peer-to-Peer Multimedia Streaming Using BitTorrent**. Performance, Computing, and Communications Conference, 2007. IPCCC 2007. IEEE International. **Anais...** [S.l.]: IEEE. , 11 abr 2007

SHRESTHA, B.; NIYATO, D.; ZHU HAN; HOSSAIN, E. **Wireless Access in Vehicular Environments Using BitTorrent and Bargaining**. IEEE Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. **Anais...** [S.l.]: IEEE. , 30 dez 2008

SZYDLOWSKI, M.; ZHAO, B. Y.; KIRDA, E.; KRUEGEL, C. **BTLab: A System-Centric, Data-Driven Analysis and Measurement Platform for BitTorrent Clients**. Computer Communications and Networks (ICCCN), 2011 Proceedings of 20th International Conference on. **Anais...** [S.l.: s.n.]. , 31 ago 2011

THOMPSON, C.; BROOKS, C.; GREER, J. **Distributed Image Processing for Automated Lecture Capture Post-Production**. Tenth IEEE International Symposium on Multimedia, 2008. ISM 2008. **Anais...** [S.l.]: IEEE. , 15 dez 2008

URVOY-KELLER, G.; MICHIARDI, P. **Impact of Inner Parameters and Overlay Structure on the Performance of BitTorrent**. INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings. **Anais...** [S.l.]: IEEE. , 23 abr 2006

VOGELEER, K. DE; ERMAN, DAVID; POPESCU, ADRIAN. **Simulating BitTorrent**. Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops. **Anais...** Marseille, France: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). , 2008

WEI HUANG; CHUAN WU; LAU, F. C. . **The Performance and Locality Tradeoff in BitTorrent-Like P2P File-Sharing Systems**. 2010 IEEE International Conference on Communications (ICC). **Anais...** [S.l.]: IEEE. , 23 maio 2010

WU, G.; CHIUUEH, T. How efficient is BitTorrent. **SPIE Multimedia Computing and Networking Conf**, v. 6071, 2006.

XIA, R. L.; MUPPALA, J. K. A Survey of BitTorrent Performance. **IEEE Communications Surveys & Tutorials**, v. 12, n. 2, p. 140-158, Second Quarter 2010.

XIAO, B.; CAO, J.; SHA, E. H.-M. **Maintaining Comprehensive Resource Availability in P2P Networks**. IN P2P NETWORKS, GCC 2004, LNCS 3251. **Anais...** [S.l: s.n.]. , 2004

YANG, W.; ABU-GHAZALEH, N. **GPS: a general peer-to-peer simulator and its use for modeling BitTorrent**. Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2005. 13th IEEE International Symposium on. **Anais...** [S.l: s.n.]. , 2005

ZEGURA, E. W.; CALVERT, K. L.; BHATTACHARJEE, S. **How to model an internetwork**. Proceedings IEEE INFOCOM '96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation. **Anais...** [S.l: s.n.]. , mar 1996

ZHANG, B.; NG, T. S. E.; NANDI, A.; RIEDI, R. H. *et al.* Measurement-Based Analysis, Modeling, and Synthesis of the Internet Delay Space. **IEEE/ACM Transactions on Networking**, v. 18, n. 1, p. 229 -242, fev 2010.

ZHENYU LI; GAOGANG XIE. **Enhancing Content Distribution Performance of Locality-Aware BitTorrent Systems**. GLOBECOM 2010, 2010 IEEE Global Telecommunications Conference. **Anais...** [S.l.]: IEEE. , 6 dez 2010

ZHONG, L.; WANG, X.; KIHIL, M. **Topological model and analysis of the P2P BitTorrent protocol**. 2011 9th World Congress on Intelligent Control and Automation (WCICA). **Anais...** [S.l: s.n.]. , jun 2011