UNIVERSIDADE FEDERAL FLUMINENSE

DANIEL VASCONCELOS CORRÊA DA SILVA

DETECÇÃO E ANÁLISE DO IMPACTO DE STREAMERS EM REDES BITTORRENT

NITERÓI

2014

UNIVERSIDADE FEDERAL FLUMINENSE

DANIEL VASCONCELOS CORRÊA DA SILVA

DETECÇÃO E ANÁLISE DO IMPACTO DE STREAMERS EM REDES BITTORRENT

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Redes e Sistemas Distribudos e Paralelos.

Orientador:
Prof. Antonio Augusto de Aragão Rocha

NITERÓI

2014

Ficha Catalográfica elaborada pela Biblioteca da Escola de Engenharia e Instituto de Computação da UFF

S586 Silva, Daniel Vasconcelos Corrêa da
Detecção e análise do impacto de *streamers* em redes B*itTorrent* / Daniel Vasconcelos Corrêa d Silva. – Niterói, RJ : [s.n.], 2014.
97 f.

Dissertação (Mestrado em Computação) - Universidade Federal Fluminense, 2014.
Orientador: Antonio Augusto de Arago Rocha.

1. Rede de comunicação de computadores. 2. Análise do modelo *Peer-to-Peer*. 3. Entropia. 4. *BitTorrent*. I. Título.

CDD 004.6

DANIEL VASCONCELOS CORRÊA DA SILVA

DETECÇÃO E ANÁLISE DO IMPACTO DE STREAMERS EM REDES BITTORRENT

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Redes e Sistemas Distribudos e Paralelos.

Aprovada em agosto de 2014.

BANCA EXAMINADORA

Prof. Antonio Augusto de Aragão Rocha - Orientador, UFF

Prof. Célio Vinícius Neves de Albuquerque, UFF

Prof. José Ferreira de Resende, UFRJ

Niterói

2014

Dedicatória(s): Dedico esse trabalho à minha avó Celina Vasconcelos, que além do seu papel de matrona, oferece diarimente exemplos de dedicação, disciplina, trabalho e estudo.

Agradecimentos

Agradeço todo o apoio, as orientações, e até as broncas do prof Antonio Augusto, sem os quais não teria jamais terminado esse trabalho.

À minha família, em especial Vilma, Laura, Nina e Fábio que me acolheram e hospedaram sempre que preciso ao longo do curso, minha mãe pelas orações e pela substituição do roteador usado nos experimentos, Pedro, Lua e Sookie pelo apoio incondicional, aos meus cumpadres Vanderson e Luiza e afilhado Arthur pelas portas sempre abertas.

Aos meus amigos Tiago, Renato e Rogério do Instituto Federal Fluminense, pelo apoio, equipamentos e revisões.

À Kelly Bentes pelo código para ler as informações do PirateBay e ao Leonardo pela ajuda com os trackers domésticos.

Às tomadas 110v dos ônibus da 1001, que me garantiram boas horas de leitura e programação.

Agradeço também meus alunos que me ajudaram nos experimentos de laboratório, sem os quais todas as semanas de baterias dos experimentos teriam se multiplicado em várias outras. Bianca, Thayná, Meloid, Maria Alice, Jhonny, Fernando, Lucas, Ayres, Mariana, Brenda e Garfinho.

Resumo

Transferência de arquivos usando tecnologia *peer-to-peer* hoje é uma das formas de compartilhamento de conteúdo mais utilizadas no mundo, e muitas ferramentas desenvolvidas como clientes BitTorrent passaram a implementar *features* que permitem aos usuários transformar essas redes em um serviço de mídia sob demanda, em que é possível reproduzir o conteúdo em áudio ou vídeo quase que imediatamente, fazendo uso da popularidade e da enormidade do acervo de mídias disponíveis nesses ambientes. Explorar essa capacidade do BitTorrent se apresenta ainda mais interessante como estratégia alternativa à arquitetura cliente-servidor usada em grande parte dos serviços de *streaming*. No entanto, transformar os enxames em serviços de mídia sob demanda pode trazer sérios prejuízos ao desempenho geral da rede, todos os tipos de usuários podem experimentar queda na qualidade de serviço, isso porque para adaptar essas redes, os clientes modificam a forma de *download* determinada no protocolo. Nessa dissertação, é apresentado um cliente BitTorrent espião, desenvolvido para monitorar trocas de mensagens na rede. Então, a partir do formato dessas mensagens, usando os conceitos de entropia, é definida uma forma de cálculo para determinar e classificar os *peers* dentro de um enxame. A partir disso, foram feitos experimentos para detectar a presença desses usuários *streamers*. Além de criar mecanismos que detectam a presença desses usuários, também foi avaliado o impacto desse tipo de usuário nos enxames.

Palavras-chave: BitTorrent, Streaming, Entropia, Peer-to-peer, mídia sob demanda

Abstract

Now a days transferring files using peer-to-peer technology is one of the most popular forms of content sharing used in the world, and many tools developed as BitTorrent clients now implement features that allow users to transform these networks in a video on demand service, being possible to play the audio or video content almost immediately, using of the popularity and the enormity of the collection of media available in these environments. Exploit this BitTorrent's capability appears more interesting as an alternative approach to client-server architecture, used in most streaming services. However, transforming the swarms into an on-demand media services can cause serious damage to the overall network performance, all types of users may experience degradation in quality of service, because those streamers clients modify the way to download determined in the protocol. In this dissertation, a spy Bittorrent client, developed to monitor exchanges of messages in the network is presented. Then, from these messages, using the concepts of entropy, is defined a calculation form to determine and classify the peers within a swarm. From this, experiments were made to detect the presence of these users in public swarms. In addition to creating mechanisms that detect the presence of these users, was also evaluated the impact of this type of user in swarms.

Keywords: BitTorrent, Streaming, Entropy, Peer-to-peer, media on demand.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

BT	:	Rede BitTorrent;
P2P	:	Peer-to-Peer;
IPTV	:	Transmissão de TV pela Internet;
DHT	:	Distributed hash table;
PEX	:	Peer Exchange;
QoS	:	Quality of Service;
GNU	:	General Public License;
MOS	:	Mean Opinion Score;
TTL	:	Time to Live;

Sumário

Capítulo 1

Introdução

Nos últimos anos, a grande melhora da infraestrutura da Internet ofereceu aos usuários ordinários uma melhora gigante na velocidade de acesso, permitindo a materialização de novos tipos de aplicação, bem como novas formas dos usuários interagirem com aplicações antigas.

O grande destaque dessas mudanças é a introdução das mídias (seja vídeo ou música) no cotidiano de grande parte dos usuários da rede. O uso de mídia na Internet, que sempre existiu, teve seu uso completamente modificado quando os usuários passaram a ter uma largura de banda (também compreendida como velocidade de acesso) suficiente para transferir o arquivo de mídia em questão durante a reprodução do mesmo, sem que houvesse interrupção ou que as interrupções fossem consideradas toleráveis pelos usuários. A esse tipo de transmissão em que a mídia é reproduzida durante o *download*, foi dado o nome de *streaming*.

Streaming pode ser tanto de mídia ao vivo, em que o conteúdo é divulgado ao mesmo tempo em que é produzido, em formato parecido com TV e rádio ao vivo; quanto é possível distribuir conteúdo produzido antes da transmissão, geralmente chamado de mídia armazenada.

É possível transmitir por *streaming* fundamentalmente em duas formas:

1. Transmissão de mídia sob demanda, em que o usuário pode escolher a qualquer tempo o que deseja assistir. Para oferecer esse poder de escolha aos usuários, a distribuição de mídia sob demanda oferece quase que exclusivamente transmissão de mídias armazenadas.

2. Transmissão de oferta única. Esse formato é análogo a TV ou rádio, pois o usuário não tem opção de escolha. E como o conteúdo compartilhado depende unicamente do canal ou estação transmissora, é possível difundir não só mídia armazenada, mas também conteúdo ao vivo.

Tratando-se de transmissão de mídia por *streaming*, quer seja ao vivo ou armazenada, um problema que deve ser avaliado é a escalabilidade. Numa estrutura cliente-servidor com uma implementação tradicional, cada usuário interessado em uma determinada mídia conecta-se ao servidor responsável pela distribuição desse conteúdo. Entretanto, devido à limitação de sua própria infraestrutura, existe um número máximo de usuários que o servidor de *streaming* consegue atender. Uma das soluções para mitigar esta limitação da distribuição desse conteúdo é usar as redes *peer-to-peer*.

O termo *peer-to-peer*, ou P2P, se refere a um sistema distribuído em que cada computador da rede se comporta, de forma simultânea, como cliente e/ou servidor em relação às outras máquinas na rede. Dessa forma, é viável compartilhar conteúdo sem a obrigatoriedade de um servidor central. Isso garante à rede a capacidade de acolher um número muito maior de usuários simultâneos e a cooperação entre eles pode, como apresentado em [26], inclusive ser usada para responder de forma apropriada cenários de *flash crowd*, quando a rede recebe num curto espaço de tempo um número muito grande de usuários, feito dificilmente realizável por um servidor central.

Existem hoje muitas redes de compartilhamento de conteúdo por *peer-to-peer*, e podemos classificá-las em dois tipos diferentes:

1 As de transmissão em P2PTV, seja rádio *on-line*, ou transmissão de TV pela Internet, também conhecida como IPTV, como apresentado em [20]. Esse tipo de rede já nasce com a conceito de *streaming* implementado, pois a reprodução é iniciada enquanto a mídia é carregada. Exemplo: PPLive [30] e SopCast [39];

2 As de conteúdo armazenado, são as redes nas quais os usuários precisam carregar os arquivos em sua totalidade antes de iniciar a reprodução. O melhor e mais popular exemplo é o BitTorrent (BT).

A diferença entre redes P2PTV para as redes de conteúdo armazenado é que, na primeira, existe a necessidade de que o conteúdo seja distribuído sequencialmente, enquanto que na rede tradicional de conteúdo armazenado, a disseminação da informação é feita obedecendo uma política de prioridade (como por exemplo, com prioridade para

o bloco mais raro), uma política de compartilhamento (por exemplo, "toma-lá-dá-cá") e uma política de compartilhamento otimista (para que novos usuários ganhem um poder de barganha, eles recebem conteúdo de usuários altruístas, mesmo sem ter como, inicialmente, oferecer uma contrapartida).

Na Internet atual, BitTorrent é sem dúvida uma das aplicações de rede mais populares do mundo, e a maior dentre as redes *peer-to-peer*. De acordo com a matéria [5] do site Bittorrent.com em 2012, a soma de usuários ativos dos *softwares* clientes μTorrent e BitTorrent já ultrapassava 150 milhões de usuários mensais, com grandes perspectivas de crescimento. Em fevereiro de 2014, o portal TorrentFreak estimou no artigo [46] que o número de usuários simultâneos nas redes de compartilhamento BitTorrent pode chegar a 30 milhões dependendo do dia da semana. Em comparação, as redes P2PTV somam juntas "apenas" dezenas de milhões de usuários como apresentado em [18], e sendo a maioria dos usuários originários da China, onde a limitação do idioma restringe fortemente a propagação do conteúdo dessas redes de compartilhamento mundo a fora.

Uma questão importante sobre as aplicações P2P *streaming* é que existe pouco conteúdo disponível, especialmente se comparado ao acervo de mídias compartilhadas por BitTorrent. Porém, o usuário mais exigente não deseja esperar o tempo de transferência total do arquivo para desfrutá-lo. O que pode se colocar diante desse contexto, então, é a seguinte questão: será que é possível ter uma aplicação P2P que opere no modo *streaming* e que desfrute de toda disponibilidade de acervos como o BitTorrent?

Uma abordagem que tem se mostrado promissora é fazer com que os programas clientes (*softwares* que os usuários utilizam para se conectar as redes BitTorrent) explorem a popularidade do acervo BT para conseguir recuperar o conteúdo em *streaming*. Essa estratégia é discutida por diversos autores em trabalhos como [36, 52]. Para isso, os aplicativos clientes se conectam às redes BitTorrent e alteram a política de seleção dos blocos, que passam a solicitar os blocos da mídia em questão de forma sequencial, ao invés de obedecer as políticas de distribuição de dados definida pelo protocolo BT. Algumas ferramentas que fazem uso desse conceito já estão disponíveis na Internet e são conhecidas como "view as you download", como por exemplo o BTStream [22].

1.1 Motivação do trabalho

Os estudos que tratam da possibilidade de explorar essa capacidade das redes BitTorrent de atender a usuários "view as you download" de modo geral o fazem considerando unicamente a ótica desse usuário interessado na reprodução antecipada e do carregamento sequencial da mídia, avaliando apenas fatores como atraso, tempo de reprodução, número e tamanho das interrupções e outros aspectos que interfiram na qualidade de serviço, ou QoS.

No entanto, a mudança na lógica do compartilhamento pode interferir no desempenho geral da rede, como apresentado em [21], afinal, existe um motivo pelo qual a política de seleção de trechos é feita de forma aleatória ou escolhendo um bloco mais raro. Estudos apresentados em [52, 27, 8] demonstram que o uso dessas políticas levam ao aumento da disponibilidade do conteúdo.

O problema de modificar a política de compartilhamento se configura ao constatar a inexistência de mecanismos de controle de entrada nas redes BitTorrent ou a falta de um limite de usuários que não respeitem as políticas de compartilhamento de conteúdo, tanto a nível do protocolo BitTorrent, quanto no nível de implementação, tratando-se da capacidade dos programas clientes de lidar com essas questões de gerenciamento geral da rede BitTorrent.

Pensando no funcionamento desse tipo de rede de uma forma mais ampla, existem algumas questões em aberto:

1. Ainda não se sabe como identificar esses usuários "view as you download" dentro dessas redes BitTorrent;

2. Não se conhece o impacto da presença desse tipo de usuário "view as you download" nas redes BitTorrent;

3. Se existe hoje um número significativo de usuários conectados às redes convencionais do BitTorrent, mas operando em modo *streaming*.

1.2 Objetivos e contribuições da dissertação

Dentro desse contexto, o objetivo deste estudo será avaliar o impacto de ferramentas "view as you download" nas redes BitTorrent. Pretende-se definir métricas, que possam ser implementadas [16] para melhoria do protocolo BT. A partir dessas métricas, possibilitar que futuramente usuários ordinários avaliem rapidamente um determinado enxame (*swarm*), permitindo a detecção de usuários "view as you download", fornecendo informações e parâmetros sobre o desempenho dessa rede BitTorrent. Para responder tais questões, esse trabalho tem como contribuições específicas:

(i) Desenvolvimento de uma arquitetura de monitoramento de enxames BitTorrent, que inclui uma versão espiã de uma aplicação que possa se conectar a essas redes e monitorar a troca de informação e dados realizada entre os demais usuários conectados;

(ii) A proposta de uma metodologia para classificar os usuários das redes BitTorrent como regulares (com requisição aleatória) ou como usuários "view as you download" (requisição sequencial) a partir da caracterização da dinâmica da evolução dos dados recebidos e capturados pelo cliente BitTorrent espião;

(iii) Uma análise, através de experimentos reais, sobre o impacto da presença de usuários "view as you download" conectados às redes BitTorrent no desempenho do sistema como um todo;

(iv) A realização de uma investigação em larga escala, através dos dados coletados com a arquitetura de monitoramento desenvolvida, com a finalidade de identificar a existência de usuários *streamers* em enxames reais do BitTorrent.

1.3 Organização do texto

O trabalho está organizado em um total de 6 capítulos. Após este capítulo introdutório, são apresentados, no Capítulo 2, uma introdução aos conceitos das redes P2P, um aprofundamento no funcionamento do sistema BitTorrent, e também é abordado o uso das redes BT como *streaming*. O Capítulo 3 propõe um método para identificar usuários em modo *streaming* dentro das redes BitTorrent. No Capítulo 4, esse trabalho apresenta os experimentos feitos para identificação de usuários *streamers* em redes reais do BitTorrent. O Capítulo 5 avalia o impacto dos usuários do modo *streaming* dentro das redes. E no Capítulo 6, são apresentadas as conclusões dessa dissertação.

Capítulo 2

Sistemas P2P: do compartilhamento de arquivos ao streaming

Nesse capitulo serão abordados aspectos fundamentais do funcionamento das redes de compartilhamento *peer-to-peer* (P2P), em especial sobre as redes BitTorrent, através da apresentação do seu protocolo e dos principais *softwares* clientes utilizados por seus usuários. Seguido pela motivação do uso de redes BitTorrent por usuários "view as you download", será descrito o funcionamento geral do modo de operação *streaming*, que torna possível a existência desses usuários, e por último a revisão de algumas das ferramentas cliente que possuem essa implementação.

2.1 Redes P2P

As redes *peer-to-peer* se tornaram a principal alternativa à arquitetura cliente-servidor na distribuição de dados. Isso por terem como característica fundamental o fato de que cada usuário (também chamado de nó ou de *peer* ou simplesmente de par) se comporta tanto como cliente e/ou como servidor. Isto é, um determinado usuário não só recebe dados, mas também é responsável pela distribuição para outros *peers*. Kurose e Ross em [20] apresentam detalhes da escalabilidade da arquitetura *peer-to-peer* e demonstram a sua superioridade quando comparada à arquitetura cliente-servidor.

O conceito desse tipo de rede é bem antigo e usado para outras finalidades que vão muito além do simples compartilhamento de arquivo. Mas, foi com o Napster [56] que o *peer-to-peer* ganhou popularidade. Criado em 1999, esse programa, além do conceito de P2P, implementou uma busca no acervo dos arquivos disponibilizados pelos demais usuários da rede e um espaço de chat para a troca de mensagens entre os usuários do

sistema.

Embora seja descrita como distribuída, a rede do Napster usava um servidor que concentrava os endereços IP e o conteúdo que cada usuário deixava disponível. Logo, quando os usuários buscavam no servidor por um determinado arquivo, ele respondia com o IP de quem detinha aquele material. Assim, então era estabelecida a conexão entre os *peers* para que o conteúdo fosse compartilhado, e o servidor não era onerado com a transferência.

No entanto, a necessidade de um servidor central para indexar o conteúdo também era a principal fragilidade da rede, uma vez que o servidor não estivesse disponível, excluindo as transferências que já haviam sido estabelecidas, todas as funcionalidades da redes parariam. O serviço saiu do ar em 2001, após uma série de processos jurídicos por parte da indústria fonográfica. Esse evento foi relatado em [56].

No começo do século XXI, foi criado o Gnutella, cujo nome é uma junção de GNU (*General Public License*) com Nutella (um doce avelã com chocolate). O projeto foi descontinuado logo no começo, pois temia-se uma repetição dos processos jurídicos que o Napster sofreu, porém a comunidade de *software* livre, usando de engenharia reversa, remontou seu funcionamento e chamou o novo projeto de protocolo Gnutella. As principais características do protocolo é ser uma rede descentralizada e não estruturada, que faz *broadcasting* de mensagens com algoritmo de inundação (*flooding*) das consultas, sendo que essas mensagens tem um tempo de vida definido com TTL (*time-to-live*).

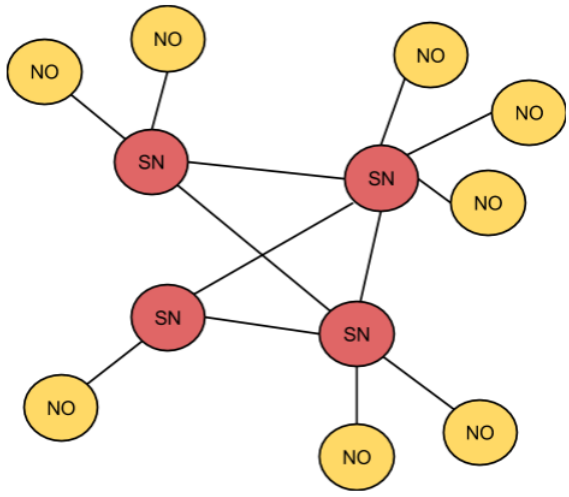

Figura 2.1: Representação de nós ordinários e super nós de redes como o KaZaA

Com uma implementação que funde algumas das ideias do Gnutella e do Napster, surgiu o KaZaA, que classificava os *peers* em, comuns e líder de grupo. Essa classificação é baseada na capacidade e largura de banda disponível para cada *peer*. Esses nós líderes, conhecidos como *supernodes*, ficam responsáveis por intermediar a comunicação entre os *peers* ordinários, como apresenta a Figura 2.1. Nessa imagem os *supernodes* estão em vermelho com nome de SN, enquanto os nós ordinários estão na cor amarela com nome de NO, e é possível perceber que os *peers* líderes funcionam como núcleo da rede.

Naturalmente existem muitas outras ferramentas. O eDonkey, por exemplo, é uma rede P2P híbrida e descentralizada, cujo servidor não compartilha dados, apenas gerencia a rede; o eMule é uma rede similar à rede eDonkey (extensão), porém permite a troca de informações entre os nós sobre os servidores, um servidor não se comunica com o outro, e utiliza uma política própria de compartilhamento; etc.

Partilhando algumas dessas características, foi criado em 2001 por Bram Cohen o BitTorrent, que é um protocolo de compartilhamento de arquivos P2P de segunda geração. Considerado por [52] como "King of P2P protocols" (Rei dos protocolos P2P), ele terá seu funcionamento explicado em detalhes na próxima seção.

E por último, porém não menos importante, existem as redes P2P que não compartilham arquivos armazenados, mas conteúdo em fluxo contínuo (P2PTV), como o PPLive [30], SopCast [39], Streamer [40], dentre outros. As redes P2PTV *streaming* de modo geral garantem escalabilidade, e podem ser organizadas para que mesmo em cenários de *flash crowd* funcionem bem, pois se adaptam dinamicamente ao tamanho da rede, existem muitos trabalhos que propõem soluções e melhorias a esse tipo de rede, como apresentado em [11].

Esse tipo de rede P2PTV é uma ótima alternativa para transmissão *streaming*, porém seu conteúdo não é facilmente universalizável devido às barreiras de idioma, pois como já dito, a grande maioria dos usuários de redes P2P de transmissão ao vivo está localizada na China.

O idioma, no entanto, não é o único problema, o tamanho do acervo, a disponibilidade e a variedade de conteúdo nas redes P2PTV são muito limitadas quando comparados ao grande acervo compartilhado por BitTorrent. Além disso, o número de usuários ainda é muito baixo, se comparado à quantidade de pessoas que usam as redes BitTorrent, como já apresentado no capítulo anterior.

2.2 BitTorrent: funcionamento

Criado por Bram Cohen, o protocolo BitTorrent [9] define o funcionamento de uma rede de compartilhamento de arquivos *peer-to-peer*. Para compreender de forma ampla essa aplicação da rede, é fundamental ter em mente que quase toda entidade do universo BitTorrent pode ser chamada de mais de um nome, e as vezes, entidades diferentes compartilham o mesmo termo, diferenciado apenas pelo contexto, além das traduções do original em inglês para o português.

As redes de compartilhamento BitTorrent são usualmente chamadas de "torrent". Nessas redes, todo usuário é chamado de *peer* ou par, e outro bom exemplo da riqueza de nomenclaturas dessa aplicação da rede é a classificação dos usuários quanto ao comportamento:

1. Leeches ou leechers – usuários que estão baixando conteúdo, por isso se comportam tanto como cliente como quanto servidor em relação aos demais *peers* que pertencem à rede;

2. Seed ou seeders ou semeadores – usuários que já tem o conteúdo completo, logo, se comportam apenas como servidor na transmissão de dados;

Juntos esses dois grupos formam os *swarms*, ou enxames. Um enxame é propriamente uma instância da rede BitTorrent, na qual um conjunto de arquivos específico é compartilhado entre os *peers* que o integram. Dentro do universo BitTorrent, o termo "torrent" também pode significar enxame, a depender do contexto. Para evitar problemas de interpretação, esse estudo não usará o substantivo "torrent" quanto este for equivalente ao conceito de enxame.

Um enxame é criado quando um usuário resolve compartilhar um arquivo (ou um conjunto de arquivos e pastas) na Internet, a esse primeiro *peer* é dado o nome de publicador ou *publisher*. Para realizar essa tarefa, o publicador, que precisa necessariamente ter o arquivo completo (logo ele é um *peer-seeder*), cria um arquivo com a extensão .torrent. Esse arquivo, por sua vez, contém metadados que permitem a outros usuários, que a eles tenham acesso, ingressar no enxame e efetivamente compartilhar a informação. A esse arquivo também pode ser atribuído o termo "torrent", que também não será usado nesse trabalho.

Esses arquivos .torrent normalmente ficam disponíveis na Internet nas páginas dos rastreadores ou *trackers*. Existe uma outra opção a esse tipo de arquivo conhecida como *link* magnético. No entanto, a diferença entre essas duas tecnologias não é relevante a esse estudo, já que as duas formas permitem o mesmo tipo de ingresso aos enxames.

O processo de criação de um enxame é ilustrado na Figura 2.2. O Publicador cria um enxame para compartilhar um determinado conjunto de arquivos. Nesse processo de criação, ele pode adicionar rastreadores que irão manter a lista com os endereços dos *peers* no enxame. E para finalizar, o publicador pode submeter o arquivo .torrent a portais de divulgação, compartilhamento e busca, como por exemplo o The Pirate Bay.

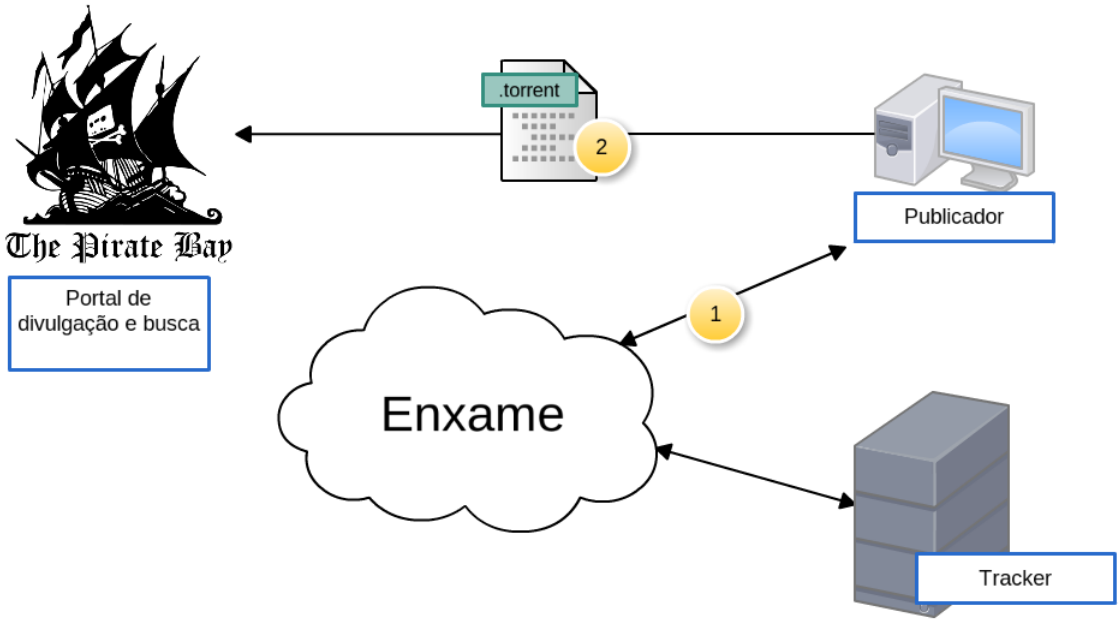

Figura 2.2: Criação e divulgação de um enxame BitTorrent

A Figura 2.3 ilustra o fluxo que um usuário interessado em um determinado conteúdo normalmente faz. Ele primeiro encontra a referência do conteúdo em um site de busca e então baixa o arquivo .torrent (ou copia o *link* magnético). Feito isso, o programa cliente BitTorrent solicita ao(s) rastreador(es) a lista de endereços de outros *peers* que compõem aquele enxame. Essas etapas (representadas por 2 e 3 na imagem) normalmente são transparentes ao usuário.

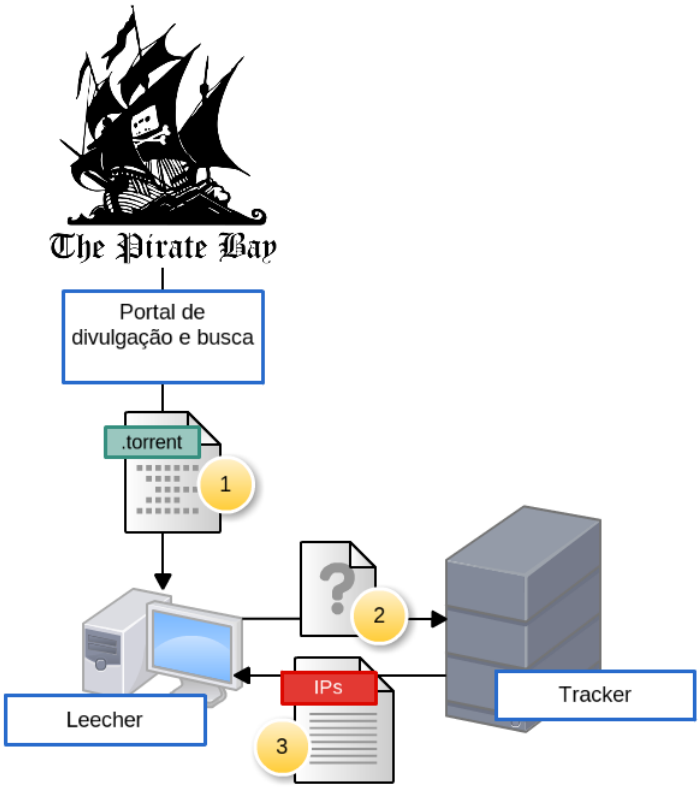

Figura 2.3: Descoberta do enxame e processo de entrada por parte de novo *peer*

Quando o publicador cria o arquivo .torrent, ele divide os arquivos em *chunks* ou blocos de 16kB. Esses blocos compõem trechos ou *pieces* de 256kB. O tamanho desses blocos e trechos pode ser alterado dependendo do *software* cliente utilizado. Essa divisão permite aos usuários *leechers* baixar trechos de diferentes *peers* concomitantemente. Para que isso seja possível, todos os *peers* precisam manter seus vizinhos informados de quais trechos possuem, usando uma estrutura de dados chamada *bitfield*.

O *bitfield* (ou *bitmap* ou mapa de *bits*) é uma estrutura de controle associada a cada *peer* que registra a evolução do *download*. Quando uma vizinhança é estabelecida, ou seja, quando um *peer* passa a ter uma conexão com outro, é enviada uma mensagem do tipo *"bitfield"* em cada sentido. A medida que um leecher amplia seu *bitmap*, ele precisa informar seus vizinhos, e isso é feito através de mensagens de *"have"*.

As mensagens de *"have"* oriundas de um determinado *peer* são concatenadas ao *bitfield* desse mesmo usuário. Então, a qualquer tempo, um *peer* sabe o que cada vizinho tem disponível. Com essa informação ele pode decidir qual trecho irá requisitar.

A Figura 2.4 representa um enxame BitTorrent. Essa representação mostra um *bitfield* associado a cada *peer*, nos quais os blocos em cinza representam os trechos já carregados pelo usuário e os em branco os blocos ainda faltantes. O *peer seeder* possui todos os blocos na cor cinza e os demais *peers* possuem apenas parte do conteúdo.

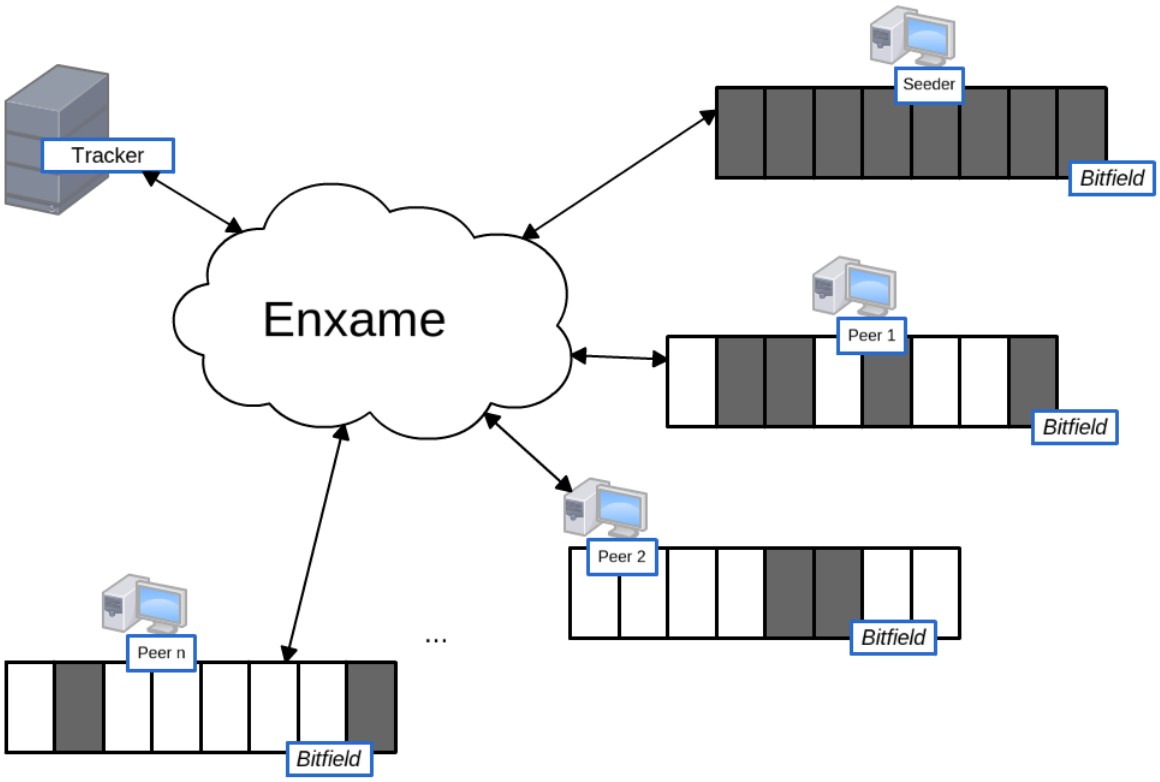

Figura 2.4: Representação de um enxame BitTorrent

Uma vez que o publicador divulgue os metadados e outros usuários se conectem ao enxame por ele criado, todos passarão a obedecer as principais políticas usadas pelo BitTorrent, que são:

(i) Política de prioridade: o trecho compartilhado é sempre o mais raro ou aleatório, o que [9] define como fundamental para performance da rede;

(ii) Política de compartilhamento *Chocking/unChocking* (tit-for-tat) – O que um determinado *peer* recebe depende do que ele compartilha, isso impede que "free riders" (termo usado para se referir pessoas que se beneficiam de um recurso sem pagar por ele) usem a rede sem dar alguma contrapartida.

É importante ressaltar que todos os usuários informam e provêm os *chunks* que possuem para os demais usando as mensagens de *"bitfield"* e *"have"*. Além disso, todo *peer* é responsável por gerir a relação com os demais, bloqueando (*Chocking*) ou desbloqueando (*unChocking*) a conexão.

(iii) Política de compartilhamento altruísta *unChocking* – Define uma política altruísta que permite que novos usuários recebam dados sem contrapartida, para então terem conteúdo para compartilhar e barganhar novos blocos. A política de prioridades, descrita anteriormente, se faz importante para esse tipo de usuário, pois amplia consideravelmente a chance do trecho adquirido por esse novo *peer* interessar a outro *leecher*. Isso não aconteceria caso todos os usuários possuíssem aquele exato trecho, como seria esperado se a política de seleção fosse sequencial.

A única estrutura centralizada da arquitetura BitTorrent é o *tracker* ou rastreador, que armazena os metadados do arquivo(s) compartilhado na rede. No entanto, a dependência do *tracker* foi reduzida drasticamente desde a implementação de *features* como DHT (*Distributed Hash Table*) e PEX (*Peer Exchange*), que já possibilitam descentralizar parcialmente as informações contidas no *tracker*. Para maiores detalhes, [20, 34] apresentam informações sobre BitTorrent e suas tecnologias.

2.3 Aplicações clientes BitTorrent e suas características

Para fazer parte das redes BitTorrent, os usuários precisam de um *software* especial que implemente as regras descritas pelo protocolo [9]. Esses programas são conhecidos como clientes BT, ou simplesmente clientes.

Nessa seção serão descritos o funcionamento e características relevantes dos principais e mais populares clientes BitTorrent. Para que a escolha dos clientes estudados não fosse totalmente definida ao acaso, foram avaliadas listas de sugestão [29, 45, 48], características relevantes (modo de operação, sistema operacional, ...) [55], além de uma lista formada a partir de uma coleta realizada durante experimentos que serão descritos mais adiante (Capítulo 4).

Quase todos os clientes possuem comportamento e usabilidade parecidos. O funcionamento do modo *streaming*, quando implementado por esses clientes, serão vistos no fim do capítulo.

- **Bitcomet** [14]

  Programa cliente BitTorrent de código proprietário, e escrito em C++. Opera apenas em ambiente Windows.

  A interface gráfica do Bitcomet não possui nenhum grande diferencial quando comparado aos demais aplicativos.

  Esse programa possui uma característica única dentre os *softwares* clientes estudados: por padrão e em qualquer modo de *download* (sequencial ou aleatório), ele sempre baixa inicialmente os primeiros e últimos blocos do *bitmap* antes dos demais trechos.

- **BitLord** [24]

  Esse cliente BitTorrent foi desenvolvido em Python e C++. Suas primeiras versões foram escritas como um *branch* do Bitcomet, mas atualmente usa o núcleo do Deluge. Essa informação deve ser levada em consideração futuramente, na identificação dos clientes.

  Software de licença GPL (*General Public License*), possui versão para Mac OSX e para ambiente Windows.

  Esse programa oferece em sua última versão para ambiente Windows, a capacidade de realizar *streaming* de mídia (áudio e vídeo), e o grupo House of Light, responsável pelo desenvolvimento dessa aplicação, afirma em seu site [25] que essa ferramenta ainda está em fase teste e eles ainda estão trabalhando para aprimorar essa funcionalidade, que não está disponível para Mac OSX.

  O BitLord oferece ainda como recurso opcional a priorização do início e do fim do arquivo. No entanto, diferente do Bitcomet, no BitLord a função se torna opcional e deve ser explicitamente definida pelo usuário marcando como opção desejada na interface da aplicação.

- **Deluge** [41]

  Esse cliente de código livre é popular, pois possui versões para PC, seja ambiente Windows ou Linux (atende a diferentes distribuições), e MAC OS X.

  Escrito em Python, o Deluge usa a biblioteca libtorrent em sua implementação.

  Por ser um cliente robusto, leve, multiplataforma e *open source*, é frequentemente usado como base para outras ferramentas.

- **Bittorrent** [4] e **μTorrent** [6]

  Bittorrent é a ferramenta referência dentro do universo torrent, e é considerada a ferramenta oficial para transmissão de arquivos dentro das redes BitTorrent [4].

  Já o μTorrent é a ferramenta mais popular dentre os clientes BitTorrent, ele compartilha muitas características com o BitTorrent, como: interface, usabilidade, modo de operação, etc.

  Ambos são desenvolvidos pelo mesmo grupo e possuem versão para Windows, Android e Mac OS X, embora só o μTorrent funcione em ambiente Linux.

- **Transmission** [49]

  Escrito em C com GTK sob a licença GPL, o Transmission é o cliente padrão de muitas distribuições Linux, também opera no Mac OS X.

  A diferença do Transmission para os demais clientes é a simplicidade da sua interface, embora implemente todas as características necessárias para executar em ambientes reais atuais, como DHT, PEX, o uso de UDP ou TCP no *handshaking* com o *tracker*. Além de suportar *links* magnéticos.

- **Ktorrent** [19]

  Popular cliente para Linux, faz parte do pacote KDE, disponível para muitas distribuições Linux. Sua interface possui as mesmas funcionalidades presentes na maioria dos demais clientes, mas se organiza de uma maneira própria e por isso possui uma usabilidade diferente.

  É desenvolvido por um grupo independente, em C++ e com licença GNU GPL.

- **qBitTorrent** [31]

  Essa versátil ferramenta para BitTorrent, tem versão para ambiente Linux, e para os sistemas operacionais Mac OS X e Windows. Assim como o BitLord, o usuário tem a opção de priorizar o início e o fim do arquivo. E, assim como o Deluge, é escrito sob o libtorrent-rasterbar, com licença de *software* livre GPL, e desenvolvido por uma equipe de voluntários.

- **Vuze** [2]

  O Vuze é um programa cliente BitTorrent recém batizado, era conhecido anteriormente como Azureus. Esse *software* de código proprietário foi desenvolvido e é mantido pela Azureus Software, Inc.. O Vuze foi implementado usando a linguagem de programação Java.

Esse cliente BitTorrent tem versão para ambiente operacional Windows, Linux e Mac OS X.

O Vuze possui implementação de diversas funcionalidades, porém as mais incomuns estão disponíveis apenas no *plugin* +VuzePlus, no qual o usuário precisa pagar uma licença de 29,90 US$ por ano para ter acesso.

- **Xunlei - Fast Thunder** [44]

  Desenvolvido pelo grupo Thunder Networking Technologies (anteriormente conhecido como Sandai Technologies), o cliente BitTorrent Xunlei é um *software* chinês de código fechado com licença de uso baseado em propaganda (*adware*).

  O nome Xunlei significa "Trovão Rápido" em mandarim, e por isso muitas vezes esse cliente BT é chamado de Thunder.

  Xunlei está disponível para os sistemas operacionais Windows e Mac OS X, e segundo o site TorrentFreak.com [47] em 2010 chegou a estar na frente do μTorrent em popularidade.

- **Popcorn Time** [28]

  Desenvolvido recentemente por um grupo de programadores argentinos, o Popcorn Time já causou grande alarde na comunidade BitTorrent e na indústria fonográfica, como Nobre descreve em seu site [17]. A proposta da ferramenta é ser um cliente BitTorrent que torne o processo de uso do BitTorrent o mais simples e leigo possível. Assim como em uma locadora on-line, similar ao serviço do Netflix [23], é apresentado aos usuários uma interface intuitiva e orientada a imagens, o que garante uma melhora significativa na usabilidade, pois o usuário pode através de poucos cliques escolher o que deseja assistir. Isso torna o processo de aquisição dos metadados (seja pelo arquivo .torrent ou por *link* magnético), a conexão e o *download* dos trechos da mídia totalmente transparentes para os usuários.

  O alarde foi tão grande na indústria fonográfica que, logo após seu lançamento em março de 2014, o projeto foi fechado, como apresentado na reportagem [43], por medo de ações judiciais. No entanto, num período de menos de uma semana a comunidade *software* livre, em um movimento orquestrado pelos desenvolvedores do site YTS [59], comprou a ideia e retomou o projeto, como também noticiado nessa reportagem [42] do site TechCrunch.

2.4 BitTorrent como *streaming*

BitTorrent é uma das principais aplicações da Internet, responsável por grande parte do tráfego na rede, chegando a um terço do tráfego mundial da Internet em 2009, como apresentado em [35]. De acordo com estudos da Cisco apresentados em [7], até 2018 o uso das redes P2P se manterá estável. No entanto, esse mesmo estudo aponta um grande crescimento do volume de dados oriundos de aplicações de vídeo sob demanda. Ele ainda prevê que em 2018 entre 80% e 90% do tráfego de toda a Internet terá essa finalidade.

Aliada à toda essa popularidade, a estrutura *peer-to-peer* garante escalabilidade quando comparada à arquitetura cliente-servidor, como afirmado nos trabalhos [36, 20, 52].

É evidente que redes de compartilhamento P2P desenvolvidas para transmissão em *streaming* ao vivo de IPTV têm uma orientação para garantir a qualidade de serviço desde sua concepção. Porém o conteúdo disponível nesse tipo de ambiente é muito pequeno. Isto porque:

1. Dependem de um elemento articulador, responsável pelo conteúdo transmitido ou de um publicador inicial na transmissão, como um canal de TV ou uma estação de rádio;

2. Sua popularidade depende fortemente da barreira do idioma. Logo, uso de redes como PPLive e SopCast faz pouco sentido fora do escopo de transmissão ao vivo.

Então, quando o conteúdo se tratar de mídia armazenada, segundo apresentado em [52], existirão dois motivos principais para se beneficiar da implementação "view as you download":

1. Reduzir o tempo gasto pelo usuário para desfrutar do vídeo (ou da mídia em questão) ao se beneficiar da alta disponibilidade que as redes BitTorrent possuem, além da imensidão de conteúdo disponível. Garantindo que o usuário desse serviço possa ir assistindo/ouvindo enquanto está baixando, sem precisar esperar o fim do carregamento para começar a reproduzir a mídia;

2. E permitir que o usuário avalie a qualidade do arquivo antes de finalizar o seu *download*, o que é extremamente útil em ambientes muito poluídos com arquivos falsos, pois garante que o usuário não vai perder tempo baixando algo que não é bem o que quer. Neste caso, poderá baixar em *streaming* apenas uma pequena fração da mídia para verificação.

2.4.1 Funcionamento

Para que os usuários possam usufruir da popularidade dos enxames para ver enquanto ainda carregam o conteúdo, é necessário modificar a política de *download* do *software* cliente BitTorrent usado. A maioria dos clientes BitTorrent solicita e faz o *download* dos trechos dos arquivos de forma aleatória ou *rarest-first* , no entanto, para que seja possível iniciar e ter uma continuidade na reprodução, a mídia precisa ser recuperada sequencialmente.

A Figura 2.5 representa a abstração de um enxame BitTorrent com um tracker e 3 *peers*. Na ilustração é apresentada uma representação do *bitmap* associado a cada *peer*. Esse mapa de *bits*, como dito anteriormente, representa os trechos já baixados pelo cliente. Na Figura 2.5, os blocos em cor cinza indicam os que já foram carregados, e os em branco os que ainda não foram. Logo, o *peer* com todos os blocos cinzas é um seeder, o usuário com blocos iniciais todos cinzas é um *peer streamer* e o último é um *peer leecher* ordinário.

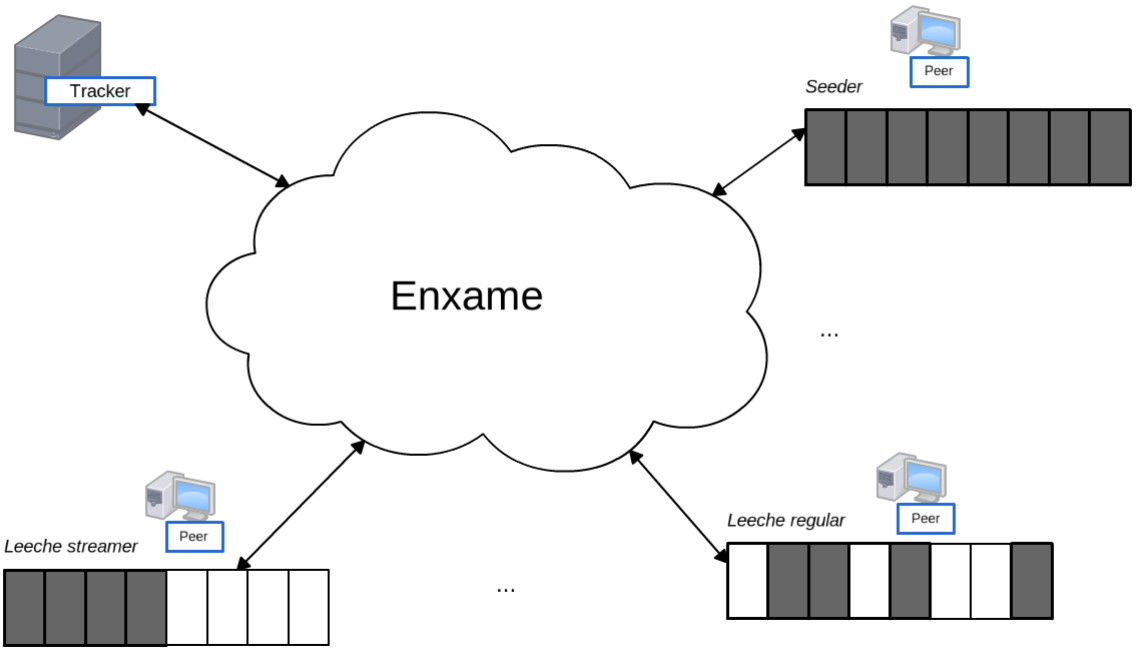

Figura 2.5: Comportamento dos *peers* aleatórios e sequenciais em um enxame

Segundo a abstração da Figura 2.5, o *peer streamer* solicitará o quinto bloco. Já o *peer* regular pode: 1) baixar o 1^{o}, 4^{o}, 6^{o} ou 7^{o} bloco, se sua política for aleatória; 2) ou pode escolher entre o 6^{o} ou o 7^{o} se o *download* seguir a determinação de requisitar sempre o bloco mais raro (*rarest-first*).

Alguns outros estudos sugerem que, junto dessa mudança no modo de *download* utilize-se também outras técnicas. A abordagem apresentada em [36] mostra que é necessário introduzir um tempo de atraso entre o início do *download* e o início da reprodução (buffer de reprodução), e modificar a política de seleção, que se dá através de uma janela deslizante que descarta os blocos que chegarem depois do atraso entre o *download* e a reprodução.

Já a implementação sugerida em [52] pretende criar um novo protocolo baseado no BitTorrent, que implemente uma janela deslizante, que crie uma classificação dos blocos (com, recebidos, prioridade e remanescentes); e que modifique a política de seleção de forma que, quando houver vários blocos com a mesma raridade, seja escolhido mais próximo da reprodução.

2.4.2 Ferramentas view as your download

Nessa seção serão apresentadas as ferramentas que oferecem *download* sequencial e como cada uma delas funciona especificamente.

Nenhuma das implementações descritas na seção anterior foi percebida de forma plena nos clientes avaliados. Esses *softwares* simplesmente mudam a forma de pedidos para modo sequencial, embora alguns até implementem janela deslizante nas suas requisições e/ou buffer de reprodução. Já a estratégia de ignorar blocos que não chegaram a tempo para a reprodução não foi percebida em nenhuma ferramenta, pois o *download* é sempre concluído em sua totalidade.

Além disso, não existe nenhuma punição da rede, e nem controle de acesso a esse tipo de usuário, o que mostra a grande distância entre produtos acadêmicos e as soluções implementadas e disponíveis na Internet.

A seguir, os programas clientes BitTorrent serão revisados quanto a funcionalidade de realizar *streaming*.

- **BitComet**

  A Figura 2.6 e a Figura 2.7 apresentam a evolução do quadro de *download* de um arquivo de vídeo, primeiro em modo padrão de *download* e depois em modo sequencial. Em ambas as imagens, o carregamento é apresentado em três momentos distintos, em 8%, 30% e 80%. Os traços azuis representam os blocos já carregados pelo cliente Bitcomet, enquanto que a barra verde apresenta visualmente a fração

de *download* realizada até então.

Conforme descrito em [15] e verificado por esse estudo através da Figura 2.6 e da Figura 2.7, o Bitcomet faz sempre primeiro o *download* dos blocos iniciais e finais, para só então depois baixar os blocos intermediários, como já mencionado anteriormente. Mesmo com essa especificidade fica clara a diferença entre os dois modos de carregamento usados por esse cliente BitTorrent.

É importante ressaltar que nos carregamentos apresentados pelas Figura 2.6 e Figura 2.7, durante a transferência o arquivo, tinham total disponibilidade e foram feitos nas mesmas condições.

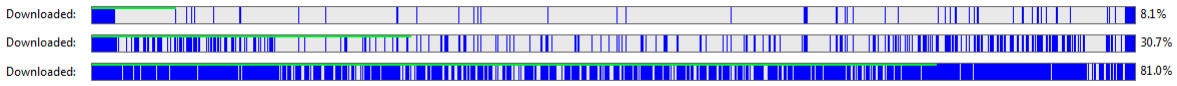

Figura 2.6: Modo aleatório do Bitcomet

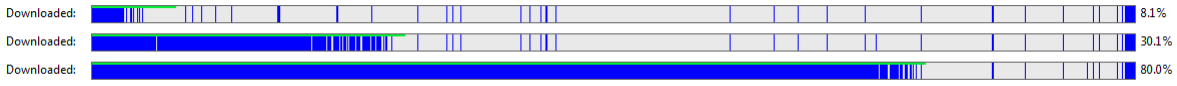

Figura 2.7: Modo sequencial (*streaming*) do Bitcomet

Para realizar o *download* em modo sequencial (*streaming*), o usuário precisa habilitar tal opção no menu da aplicação (vide em destaque na Figura2.8). No BitComet o *download* sequencial se chama "Preview Download Mode".

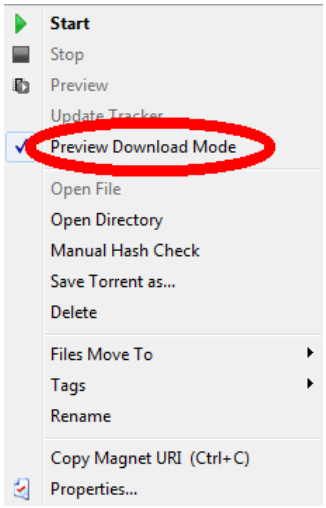

Figura 2.8: Ativação do modo aleatório do BitComet

- **BitLord**

  Essa ferramenta possui modo sequencial, que para funcionar o usuário deve confirmar manualmente no botão "Stream" na guia "Files", como apresentado na Figura 2.9. Outra forma de ativar o modo *streaming* é inserir o registro do enxame em uma playlist e então reproduzir a mídia em um *player* qualquer.

  Esse cliente BitTorrent possui um *player* próprio para reprodução dos arquivos em *download*, o que indica o propósito da mudança da política de requisição dos blocos.

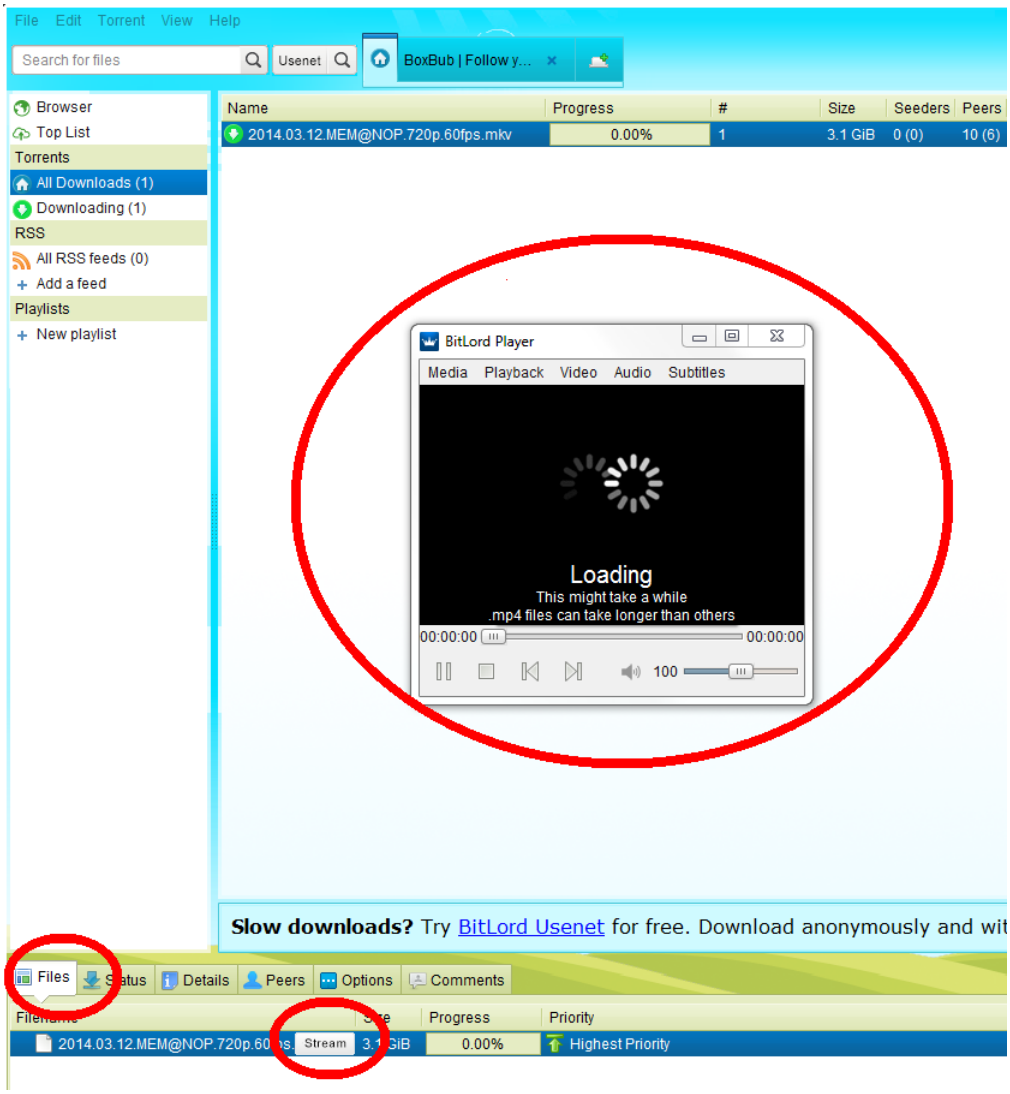

Figura 2.9: Modo *streaming* do Bitlord

- **BitTorrent**

  Esse cliente implementa uma versão limitada de *streaming* que permite apenas poucos segundos de *download* sequencial. Para ativar essa funcionalidade o usuário deve clicar no botão "Play" da coluna *Playback*, conforme mostra Figura 2.10.

  Em seu FAQ [54] existe apenas uma pergunta sobre o modo *streaming* e a resposta desencoraja o uso de mecanismo por conta da política de compartilhamento "toma-lá-dá-cá" do protocolo BitTorrent.

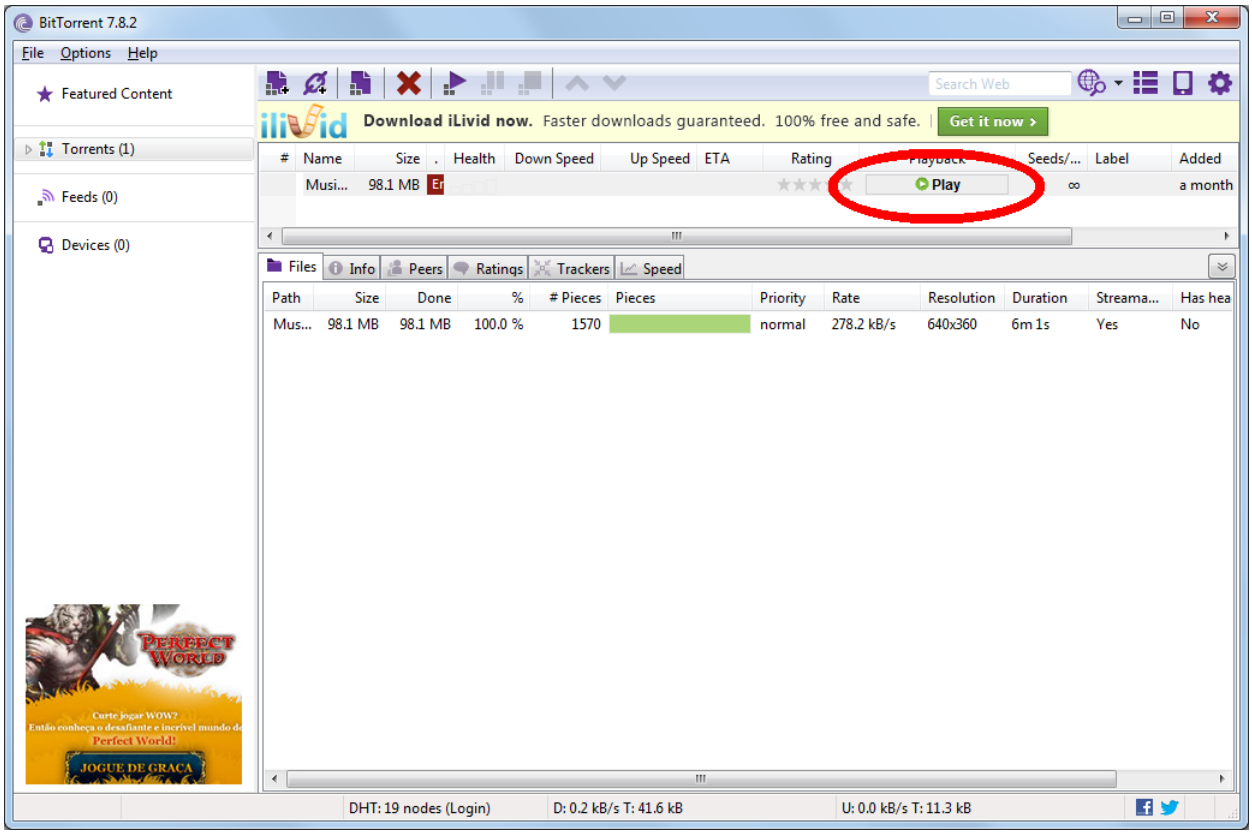

Figura 2.10: Modo *streaming* do BitTorrent

- **μTorrent**

  Seu modo *streaming* (e o cliente BitTorrent possivelmente também) foi limitado para evitar uma perda de desempenho na rede, conforme explicam os desenvolvedores desse programa cliente no site oficial [50]. A ativação do modo *streaming* desse cliente é similar à ativação do cliente Bittorrent.

  O μtorrent implementou na versão 3.0 uma configuração que permitia modificar o tempo em que as requisições seriam sequenciais e o tamanho do *buffer*, como pode ser verificado na Figura 2.11. Nessa versão, uma vez ativado o modo sequencial,

ele perduraria até o fim do carregamento, seguindo os parâmetros indicados pelo usuário.

Porém, na versão atual 3.4 esses parâmetros são determinados *hardcoded*, para que o usuário use esse recurso apenas para verificar a qualidade do arquivo, como é possível notar na Figura 2.12. E o modo *streaming* fica ativo apenas por alguns segundos, tempo suficiente apenas para avaliar a qualidade e veracidade do conteúdo.

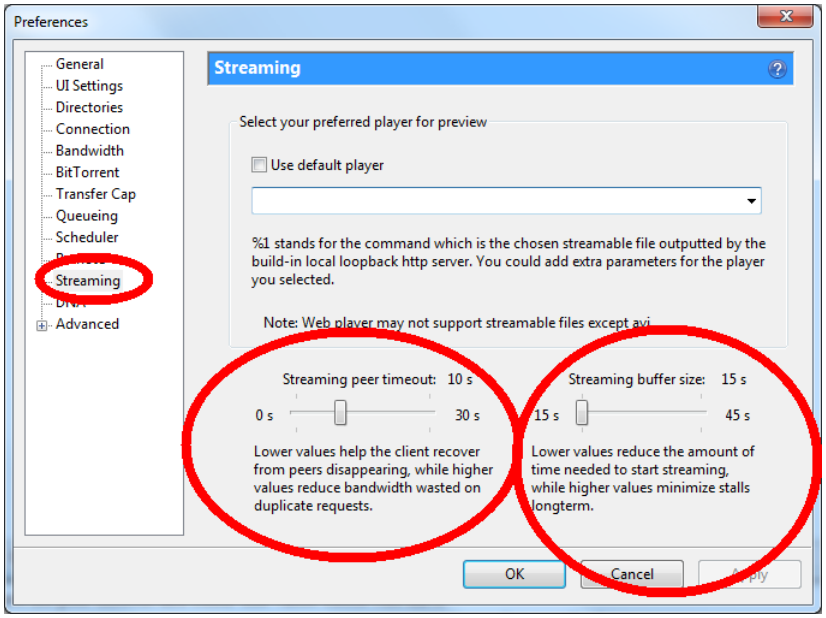

Figura 2.11: Configuração de *streaming* do μTorrent versão 3.0

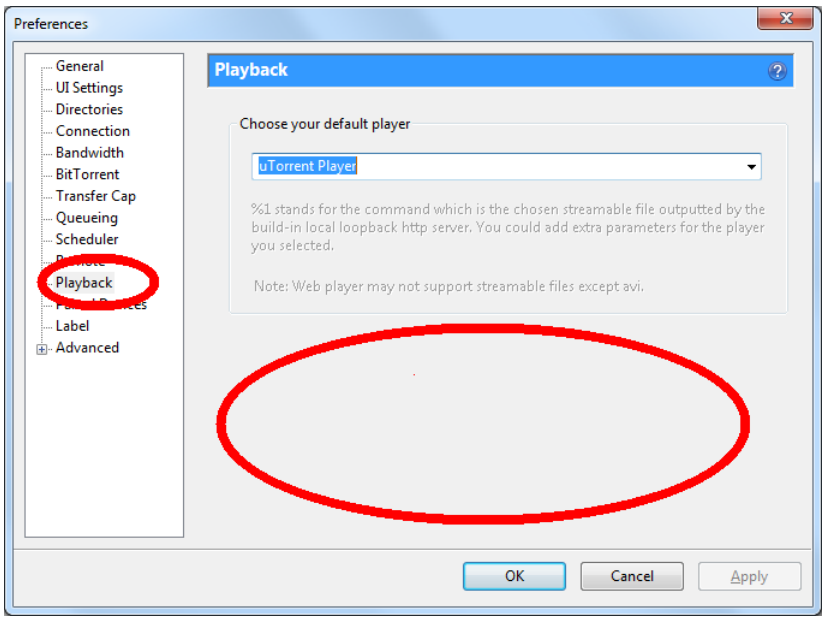

Figura 2.12: Configuração de *streaming* do μTorrent versão 3.4

- **KTorrent**

  KTorrent permite a instalação adicional de um *plugin* de *MediaPlayer* que possibilita ao usuário reproduzir o vídeo enquanto realiza o *download*, pois esse *add-on* modifica a política de *download* para sequencial. A Interface com a presença desse *plugin* é vista na Figura 2.13.

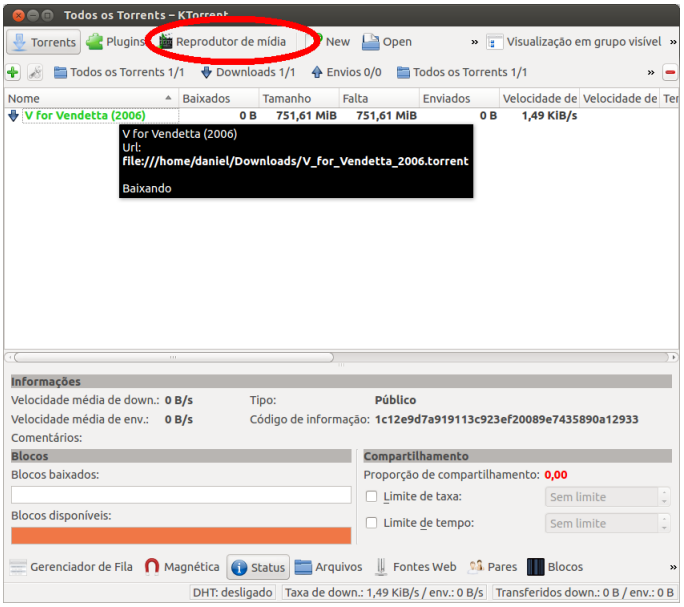

Figura 2.13: Modo *streaming* do KTorrent

- **qBitTorrent**

  Possui um modo *streaming* nativo e não requer instalação de *plug-ins* adicionais. No qBitTorrent o nome dessa *feature* é "Download in sequential order" e pode ser ativada pelo menu da aplicação como, mostra a Figura 2.14. Esse cliente BitTorrent não possui *software* de reprodução próprio, e reprodução evoca o tocador padrão do sistema operacional.

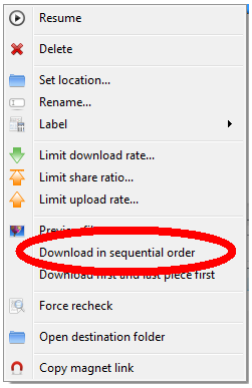

Figura 2.14: Ativação do *streaming* do BitTorrent

- **Xunlei**

  Além da interface BitTorrent que implementa a *feature* "view as you download", o Xunlei Thunder integra uma solução de vídeo sob demanda chamada Xunlei Kankan, o que torna esse *software* um produto focado em distribuição de mídia por *streaming* na Internet.

  Possui *player* próprio, como pode ser percebido na Figura 2.15, que mostra também o botão "Play", usado para reproduzir o arquivo durante o *download*.

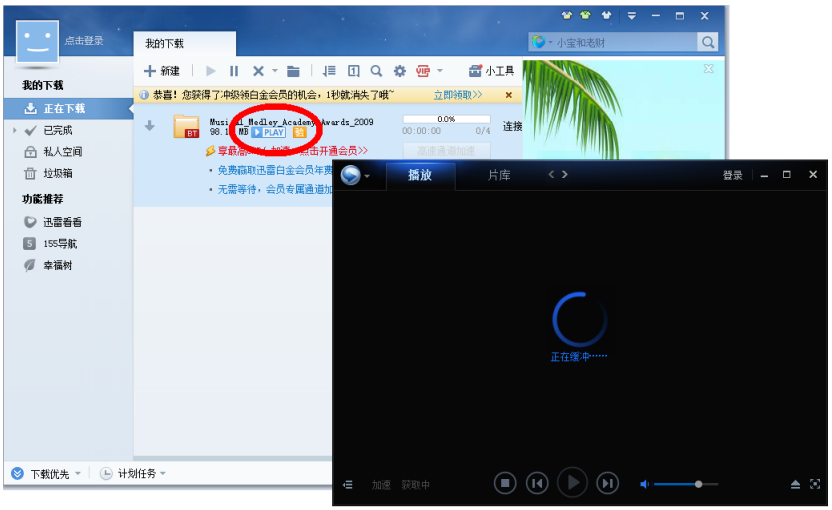

Figura 2.15: Modo *streaming* do Xunlei

- **Vuze**

  O modo sequencial do Vuze está disponível apenas através da instalação do *plug-in* Vuze Plus já citado anteriormente. O Vuze Plus disponibiliza também um *player* próprio, diferente da maioria dos outros clientes. Essa funcionalidade está descrita em [53]. A interface em modo *streaming* é mostrada na Figura 2.16.

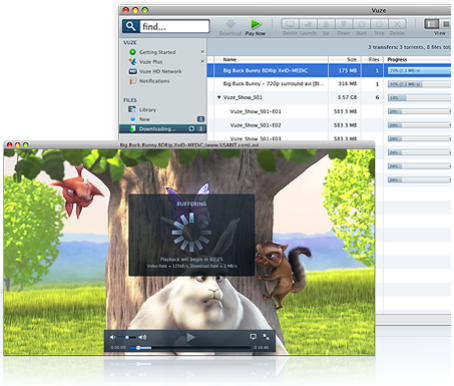

Figura 2.16: Modo *streaming* do Vuze

- **Popcorn Time**

  Segundo o FAQ do *site* desse incomum cliente BitTorrent, o Popcorn Time permite que o usuário escolha dentro de uma seleção de vídeos do portal YTS [59] qual filme deseja ver.

  Vale ressaltar que a interface *desktop* do programa não requerer do usuário arquivo *.torrent* ou *link* magnético, ela simplesmente exibe quais filmes estão disponíveis para reprodução em modo *streaming*.

  O Popcorn Time também oferece ao usuário a lista de legendas disponíveis, o que reduz consideravelmente as barreiras de idioma, que é normalmente um grande problema desse tipo de acervo.

  Essa ferramenta que não possui modo de recuperação dos blocos que não seja o seqüencial (aleatório ou *rarest-first*, por exemplo). Ela, faz uso do tempo de espera previsto por [36] entre o *download* e a reprodução, como inclusive pode ser percebido na Figura 2.18.

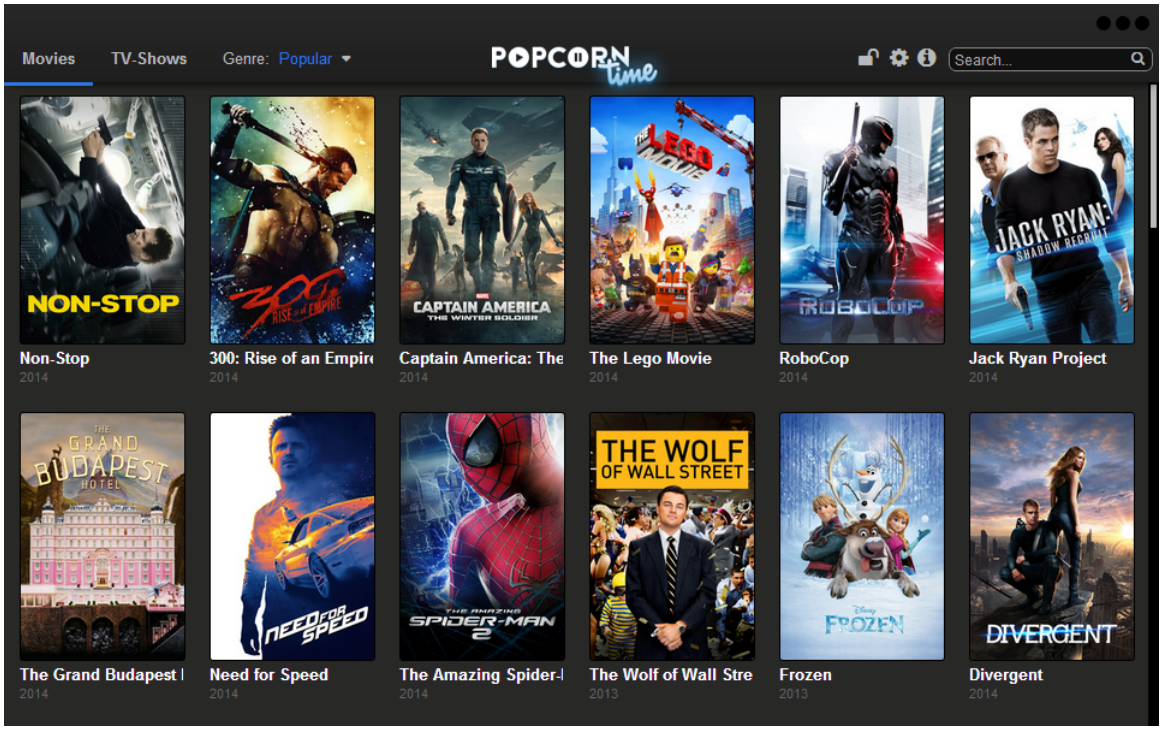

Figura 2.17: Interface inicial do Cliente Popcorn Time

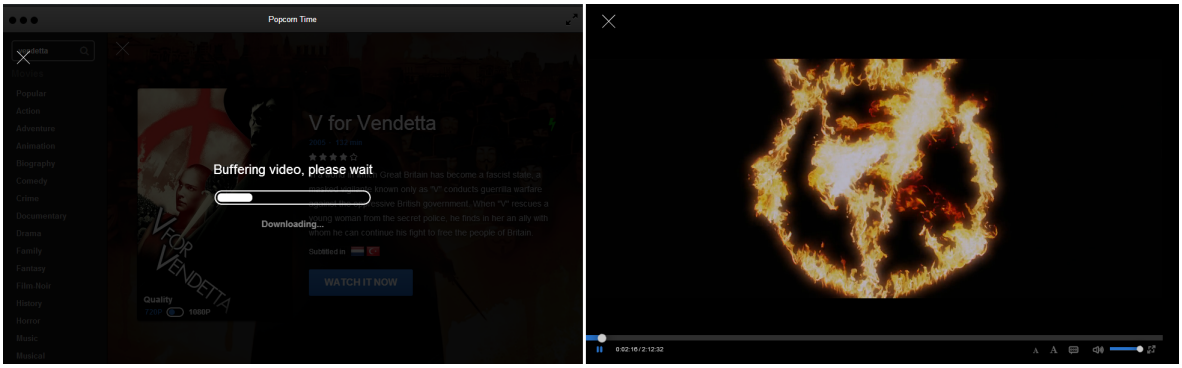

Figura 2.18: Interface do Cliente Popcorn Time durante atraso da reprodução e durante a reprodução

- **Central Multimídia baseada em BT *streaming***

  Além dessas implementações de *software* apresentadas nas seções anteriores desse capítulo, existem soluções embarcadas que concatenam *hardware* específico e uso de diversas aplicações para criar uma central multimídia baseado em *streaming* sob demanda usando acervo BitTorrent.

  O uso desse tipo de integração promete promover uma usabilidade ainda mais simples que soluções como o Popcorn Time ou de IPTV, pois:

  1. Não depende um repositório pré-definido. O usuário pode adicionar os metadados de um enxame específico ou apenas desfrutar de mídias sugeridas pela solução. No Popcorn Time, por exemplo, o usuário só tem disponível os filmes do portal YTS;
  2. Todo o controle pode ser feito pelo controle remoto da televisão, o que amplia consideravelmente o universo de usuários, incluindo idosos ou portadores de necessidades especiais, que usualmente têm mais dificuldade com esse tipo de ferramenta. O usuário não precisa interagir com o computador nem tão pouco fazer qualquer alteração na instalação física;
  3. Seu uso é orientado a imagens, e organiza o acervo pelo critério desejado.
  4. Oferece legenda em diversos idiomas, o que torna a solução muito melhor que as IPTV cujo problema central é exatamente vencer as barreiras de idioma.

  Gomes apresentou em seu site [13] uma dessas soluções usando Raspberry PI [32], Transmission [49], CouchPotato [10] e Sick Beard [38].

Capítulo 3

Eye of the Swarm - Um método para identificar BT streamers

A revisão apresentada no capítulo anterior, sobre as ferramentas BitTorrent, leva a crer que a existência de usuários realizando *streaming* em enxames BitTorrent já é um fato. Logo, o desenvolvimento de técnicas que permitam a detecção de usuários "burlando" o modo de operação do sistema é de suma importância, não só para ajudar compreender a realidade atual dessa popular rede P2P, mas também para auxiliar na criação de mecanismo que permitam inibir tais usuários, se assim for desejado. A questão que se colocá é, será que é possível desenvolver uma técnica passiva (i.e., sem interferir no funcionamento regular do enxame) que permita identificar a existência deste tipo de usuários em um enxame BitTorrent?

A proposta apresentada nesta dissertação, que responde positivamente a questão colocada acima, permite identificar os usuários "view as you download" em simultâneo à sua operação no enxame e de forma não intrusiva ao sistema. Para isso, a proposta faz uso das mensagens de *"have"* recebidas de outros *peers* e lança mão de uma teoria derivada da Entropia para realizar a identificação dos usuários.

Logo, antes de descrever a metodo proposto (Seção 3.3), será destacada algumas questões relevantes sobre o funcionamento do protocolo (Seção 3.1) e introduzidos alguns conceitos fundamentais sobre Entropia (Seção 3.2).

3.1 Como definir um método não intrusivo?

O protocolo BitTorrent define que todos os *peers* que compõem um determinado enxame devem informar aos seus *peers* vizinhos (são considerados *peer* vizinhos os usuários que estão com uma conexão estabelecida entre si) quais partes do arquivo já foram por eles carregados. A primeira mensagem será sempre do tipo *"bitfield"*, que apresenta o mapa de *bits* em posse daquele *peer* responsável pelo envio da mensagem. Seguida dessa primeira mensagem, serão enviadas para os vizinhos as mensagens do tipo *"have"*, sempre que o usuário remetente obtiver posse de um novo trecho e, o mesmo tiver sua verificação *hash* completada.

Os receptores dessas mensagens somam às mensagens de *"have"* a mensagem inicial do *bitfield*, e podem determinar num dado instante a atualização do mapa de *bit* relativo ao *peer* remetente. De posse dessas informações, obtidas de forma não intrusiva, é possível ter uma noção de qual estratégia de *download* está sendo utilizada pelo usuário, já que em uma implementação aleatória ou *rarest-first*, as mensagens terão um comportamento independente e próximo de uma distribuição uniforme, enquanto que uma estratégia de *download* sequencial apresenta um comportamento contínuo.

No entanto, para a definição do método é preciso fazer uso de alguma fundamentação teórica que permita identificar de forma automática e com certo grau de confiança, se o usuário está ou não realizando *streaming* dos blocos.

Diferentes teorias poderiam ser usadas para automatizar a identificação do modo de operação do usuário, tais como índice de *herfindahl* [57] ou coeficiente de gine [58]. Porém, é imperativo usar alguma teoria que pudesse considerar a implementação de janela deslizante, comumente presente em aplicações na Internet, e por isso nesse estudo foi definido o uso do conceito de entropia.

3.2 Entropia de Shannon

Presente em diversas áreas, o conceito de entropia foi adaptado e generalizado pelo matemático americano Claude Shannon em 1948 no trabalho "A Mathematical Theory of Communication" [37]. Por isso, no contexto de teoria da comunicação (informação), esse tipo de entropia é muitas vezes chamada de entropia de Shannon (Shannon entropy). Shannon apresenta em seu trabalho que entropia é a medida de incerteza associada a uma variável aleatória, ao quantificar o valor esperado numa mensagem, geralmente em

unidades como *bits*, para definir quais opções podem ocorrer ou podem ser selecionadas, dado um determinado conjunto de possibilidades.

A entropia de Shannon considera um conjunto finito de elementos, onde a probabilidade de cada elemento é conhecida, e que seja possível calcular a quantidade das escolhas envolvidas na seleção do evento seguinte ou quão incerto é o resultado. A partir dessa configuração, Shannon define em [37] as seguintes propriedades:

1. H deve ser contínuo em p_i

2. Se todos os p são iguais, $p_i = \frac{1}{n}$, então H deve ser uma função monótona crescente de n. Com eventos igualmente prováveis, há mais escolha (ou incerteza) quando há mais eventos possíveis.

3. Se uma opção for dividida em duas escolhas sucessivas, o H original deve ser a soma ponderada dos valores individuais de H.

No já referido estudo, Shannon demonstra em seu "Teorema 2", que o H que satisfizer essas três propriedades terá a forma:

$$H = -K \sum_{i=1}^{n} p_i \log p_i$$

Shannon apresenta ainda em seu estudo a prova do teorema, embora não caiba a este trabalho revisá-lo. Então, considerando um conjunto de eventos, em que cada um desses eventos tenha apenas duas possibilidades (como o lançamento de moedas), eles devem seguir uma função binária entrópica, que é uma simplificação da expressão de Shannon para duas possibilidades:

$$H(x) = -p.\log_2 p - (1-p).\log_2(1-p) \tag{3.1}$$

Para exemplificar, considere as sequências:

a) $x_t = \{0,1,0,1,0,1\}$

H = -($\frac{1}{2}$.$\log_2$ $\frac{1}{2}$) -($\frac{1}{2}$.$\log_2$ $\frac{1}{2}$)

H = -($\frac{1}{2}$.-1) -($\frac{1}{2}$.-1)

H = 1

b) x_t = {1,1,0,1,0,1}

H = -($\frac{4}{6}$.$\log_2 4_{\frac{}{6}}$) -($\frac{2}{6}$.$\log_2 2_{\frac{}{6}}$)

H = 0,389975 + 0,52832083

H = 0,918295

c) x_t = {1,1,0,1,1,1}

H = -($\frac{5}{6}$.$\log_2 5_{\frac{}{6}}$) -($\frac{1}{6}$.$\log_2 1_{\frac{}{6}}$)

H = 0,219195 + 0,430827

H = 0,650022

d) x_t = {1,1,1,1,1,1} ou {0,0,0,0,0,0}

H = -(1.$\log_2$1)

H = -(1.0)

H = 0

Pelo valor da entropia desses conjuntos, é possível perceber que quanto maior o valor, mais complexo é a série e quanto menor a entropia, mais previsíveis são os valores de M, então:

0 → Previsível

1 → Aleatório (entropia de uma moeda honesta)

Generalizando, se x_t tem um número finito de valores M, a entropia clássica H de Shannon, mede o valor médio de futuro x_t+1, dado todo passado (..., x_t-1, x_t), tendo:

$0 \leq H \leq \log M$

H = 0 → Série previsível

H = log M → Independente e uniformemente distribuído

3.2.1 Entropia por permutação

O cálculo da entropia de Shannon se torna difícil à medida que o número de possibilidades e M aumentam. Existem algumas abordagens para resolver essa questão, como criar partições no conjunto M, embora nenhuma mantenha o cálculo tão simples

como a estratégia de avaliar o valor da entropia pela configuração de cada permutação de tamanho n, ao invés de avaliar a probabilidade de cada elemento do conjunto. Esse tipo de entropia é chamada de Entropia por permutação. Ela foi apresentada em [3] e revisada por [33], e não deixa de apresentar a medida da complexidade da série avaliada.

Então, considerando M finito, deve se negar valores iguais no conjunto (x_t* = x_t, t* <> t), e [3] recomenda usar n entre 3 e 7, onde n representa as dimensões embebidas (tamanho de cada permutação).

O valor mínimo da entropia de permutação é 0, para um conjunto totalmente sequencial, e o maior valor de H(n) será log n!, para um conjunto completamente aleatório.

$0 \leq H(n) \leq \log_2 n!$

Então, generalizando, estuda-se todos os n, sendo permutações tipos de ordenações de tamanho n, contabiliza-se a frequência relativa de cada permutação do conjunto M. A entropia de permutação de ordem n $\geq$ 2, é definida como:

$$H(n) = -\sum p(\pi) \log p(\pi)$$

Considerando valores de entropia por permutação da mesma série e criadas a partir de diferentes valores de permutação, é importante que seja usada uma escala comum quando for estabelecida a comparação entre essas entropias por permutação.

Como o maior valor possível para H aumenta com o tamanho de n, para normalizar os valores possíveis numa escala [0,1] basta dividir a entropia por permutação por $\log_2 n!$. A Tabela 3.1 apresenta com mais clareza os valores máximos para permutações de tamanho 2, 3, 4 e 5 e suas normalizações.

Tabela 3.1: Limites da entropia por permutação para diferentes valores de n

Range	**Menor valor**	**Maior valor [log(n!)]**
H(2)	0	1
H(3)	0	2.58496250072116
H(3) Normalizado	0	1
H(4)	0	4.58496250072116
H(4) Normalizado	0	1
H(5)	0	6.90689059560852
H(5) Normalizado	0	1

3.2.1.1 Exemplo de entropia por permutação

Para ampliar a compreensão do cálculo da entropia por permutação, considere o seguinte conjunto de números de tamanho 10, em que cada um dos elementos é único:

x_t = {9,32,50,71,13,99,5,17,8,64}

Para calcular entropia de ordem n = 2, são formados 9 (tamanho do conjunto subtraído de n-1) pares ordenados (a,b):

(09,32) → a < b;

(32,50) → a < b;

(50,71) → a < b;

(71,13) → a > b;

(13,99) → a < b;

(99,05) → a > b;

(05,17) → a < b;

(17,08) → a > b;

(08,64) → a < b.

6 pares [a < b] e 3 pares [a > b],

H(2) = -($\frac{6}{9}$.$\log_2$ $\frac{6}{9}$) -($\frac{3}{9}$.$\log_2$ $\frac{3}{9}$)

H(2) = 0,389975 + 0,52832083

H(2) = 0,9182958341

Considerando o mesmo conjunto, para calcular entropia de ordem n = 3, são formadas 8 (tamanho do conjunto subtraído de n-1) triplas ordenadas (a,b,c):

(09,32,50) → a < b < c;

(32,50,71) → a < b < c;

(50,71,13) → c < a < b;

(71,13,99) → b < a < c;

(13,99,05) → c < a < b;

(99,05,17) → b < c < a;

(05,17,08) $\rightarrow$ a $<$ c $<$ b;

(17,08,64) $\rightarrow$ b $<$ a $<$ c;

2 triplas [a $<$ b $<$ c] e 2 triplas [c $>$ a $>$ b], 2 triplas [b $<$ a $<$ c], 1 tripla [b $<$ c $<$ a] e 1 tripla [a $<$ c $<$ b],

H(3) = -3($\frac{2}{8}$.$\log_2$ $\frac{2}{8}$) -2($\frac{1}{8}$.$\log_2$ $\frac{1}{8}$)

H(3) = 3*0,5 + 2*0,375 = 1,5 + 0,75

H3 = 2,25

H(3 normalizado entre 0 $\rightarrow$ 1) = 0,87041882

Os valores limites de entropia com n = 3 são: 0 (zero) para conjunto totalmente previsível, e o valor máximo de H(3) será log 3! = 2,5849625, para um conjunto totalmente aleatório.

3.3 Descrição da Proposta

Só é viável criar uma forma de tornar automática a detecção de usuários "view as you download" nos enxames se:

1. For possível acompanhar a evolução do *bitfield (bitmap)* de cada *peer*.
2. E se for viável determinar o valor esperado para uma próxima mensagem, isso é, o novo mapa de *bits* associado à mensagem.

Então, considerando que se conheça a evolução do *bitfield*, é possível determinar se um usuário está requisitando trechos de forma sequencial ou aleatória (*rarest-first* tende a ser uniformemente distribuído em redes BitTorrent). Então, mesmo que o cliente use estratégia de janela deslizante em sua implementação a entropia por permutação consegue detectá-lo, como defendido em [3].

3.3.1 Espião no Enxame

Para estudar o comportamento dos usuários BitTorrent, foi desenvolvida uma ferramenta pra monitorar a troca de dados. O *software* escolhido inicialmente foi o *InstrumentedBT*, porém essa ferramenta não implementa algumas características para se

conectar a enxames dos principais publicadores (*links* magnético, usar UDP para conectar, DHT, PEX ...). Por esse motivo, a ferramenta Transmission passou a ser usada, uma vez que se trata de um cliente com frequente atualização, popular e de *software* livre. Os detalhes sobre a implementação do espião estão descritas no Apêndice A desta dissertação.

O cliente espião funciona como um cliente regular, e é visto pelo enxame como tal. Porém ele manipula fundamentalmente os dados de controle, e registra a evolução do *bitfield* de seus vizinhos. Isso é, em qual ordem os *bits* que compõem o arquivo foram recebidos pelos outros *peers*.

Existem três questões centrais sobre o funcionamento do espião, a primeira é que ele assume somente posse e não intenção, ou seja os pedidos que o *peer* tenha feito; A segunda, é que pode existir uma diferença entre o tempo das requisições, recebimentos e a mensagem de have. A representação do *bitfield* é binária e garante uma verificação mais precisa, mas também pode ser feita considerando blocos; E a última é que o espião compartilha dados de controle. O motivo pelo qual foi feita essa redução na troca de dados é para que fique clara a intenção da aplicação e evitar eventual violação a legislação que rege direitos autorais.

3.3.2 Validação do método

Para validar o método foram feitos experimentos tanto em ambientes controlados como em ambientes públicos. Nestes experimentos foram testados alguns dos clientes BitTorrent já descritos nesse trabalho nas Seções 2.3 e 2.4.2. Para cada um desses clientes foram feitas baterias considerando os modos de *download* aleatório e sequencial (se existia). Logo, foram executadas até quatro baterias por programa cliente:

1. Ambiente controlado em modo aleatório
2. Ambiente controlado em modo *streaming*
3. Ambiente real em modo aleatório
4. Ambiente real em modo *streaming*

3.3.2.1 Ambiente Controlado

Realizado em ambiente controlado dentro de uma rede local, esse experimento levou em consideração as configurações especificadas a seguir. É imperativo ponderar na análise que esse é um ambiente de alta disponibilidade, e que nesse cenário o cliente BitTorrent testado poderá pedir o bloco que melhor atende seu modo de *download*.

1. Publicador qBitTorrent em uma máquina Linux Ubuntu 13.10;
2. Espião Transmission em uma máquina Linux Ubuntu 13.04;
3. LAN sem controle de banda;
4. Arquivo de vídeo no formato avi com 102,9MB;
5. Velocidade ilimitada;

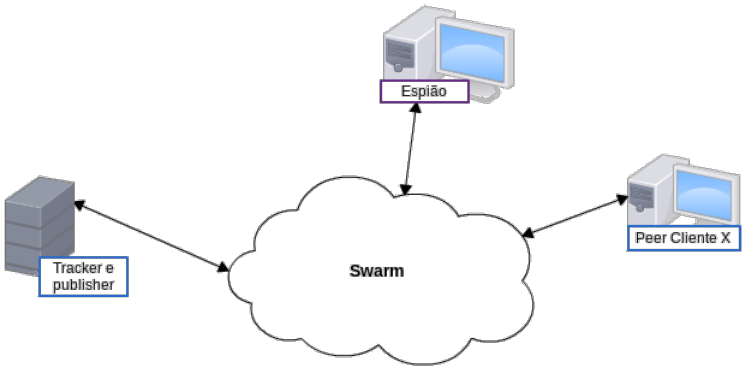

Figura 3.1: Esquema do experimento para calcular a entropia base dos clientes BitTorrent em ambiente controlado

Como é possível perceber na Figura 3.1, o experimento consiste em registrar a troca de mensagens na rede entre o *peer* com o cliente estudado (identificado na imagem como "*Peer* Cliente X") e o publicador. O primeiro passo é ligar o publicador/rastreador, em seguida, conecta-se o espião ao publicador. E por último, o cliente estudado é conectado com velocidade ilimitada.

3.3.2.2 Ambiente Público

Para realizar esse experimento, foi escolhido um enxame com poucos *peers*, de forma a garantir a vizinhança entre o cliente espião e o cliente em estudo, ou seja um ambiente de provável baixa disponibilidade. Nesse cenário a natureza do modo de *download* pode ser camuflada de acordo com os trechos disponíveis. Além disso, o experimento seguia as seguintes configurações:

1. Publicador público de configuração desconhecida;
2. Espião Transmission em uma máquina Linux Ubuntu 13.04;
3. Conexão de Internet doméstica;
4. Arquivo de vídeo no formato avi com 115,3MB;
5. Velocidade ilimitada;

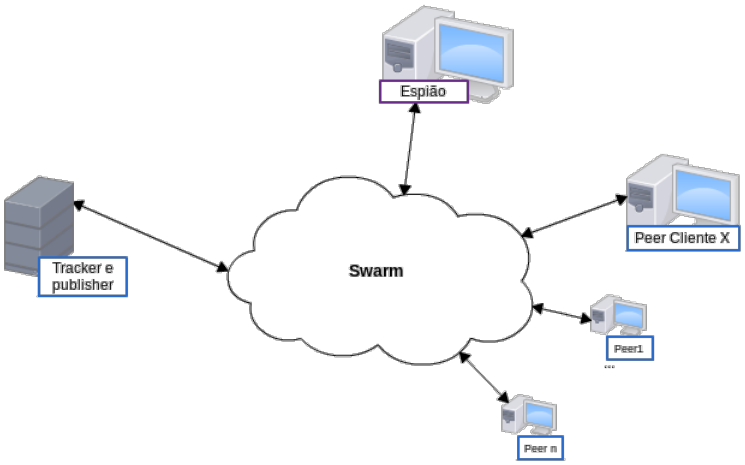

Figura 3.2: Esquema do experimento para calcular a entropia base dos clientes BitTorrent em ambiente público

Como é possível perceber na Figura 3.2, o experimento consiste em registrar a troca de mensagens na rede entre o *peer* com o cliente estudado (identificado na imagem como "*Peer* Cliente X") e um *swarm* público. O primeiro passo foi definir um enxame, para tal foi escolhido um que não possuísse muitos *peers*, para garantir a vizinhança do espião com o *peer* que estava usando o cliente em estudo. Além dessa característica, o arquivo também devia ser um vídeo com tamanho parecido ao do usado no experimento similar feito em ambiente controlado.

Depois de definir o enxame, conecta-se o espião ao publicador. E por último, o cliente estudado é conectado com velocidade ilimitada ao enxame, usando o mesmo arquivo .torrent ou *link* magnético usado pelo espião.

3.3.2.3 Análise dos resultados

De posse dos registros de troca de mensagem realizados nas baterias do experimento, considerando ambiente controlado e ambiente real público, foram calculados os valores de entropia por permutação dos programas clientes BitTorrent (*peer* Cliente X) funcionando em modo aleatório e em modo sequencial.

Para o cálculo da entropia por permutação o estudo considerou 3 valores de dimensões embebidas, isso é, tamanhos de permutação n = 2, n = 3 e n = 4, bem com suas normalizações, para cada um dos clientes estudados anteriormente, como apresenta a Tabela 3.2 e a Tabela 3.3. Esses tamanhos de permutação conseguem acompanhar um tamanho de janela deslizante de até 1024k para H(4), isso porque o cálculo da evolução leva em consideração os blocos concluídos, e cada bloco possui geralmente 255k.

Nessas tabelas (3.2 e 3.3) é possível verificar que existe uma grande diferença entre os valores, em ambas estratégias de *download* e entre os clientes. Isso reflete as diferentes implementações adotadas pelos desenvolvedores das aplicações.

Pode ser observado nos valores do cliente Bitcomet que existe uma diferença clara entre os dois modos de *download*. É possível também verificar que num ambiente de menor disponibilidade (ambiente real) o valor de entropia por permutação é ligeiramente maior, fato que se constata em todos os outros clientes.

No caso dos clientes μTorrent e BitTorrent a diferença entre um modo de requisição e outro é quase imperceptível. Isso pode ser compreendido ao resgatar a informação vista no Capítulo anterior de que esses dois clientes, em suas atuais versões, permitem *streaming* por apenas poucos segundos.

Tabela 3.2: Entropia dos clientes BitTorrent em ambiente controlado

Cliente	Modo	Nº Msg	H(2)	H(3)	H(3)Norm	H(4)	H(4)Norm
Bitcomet	Aleatório	163	0.9982	2.5729	0.9953	4.5191	0.9856
Bitcomet	Sequencial	1569	0.3966	0.7560	0.2924	1.0936	0.2385
BitLord	Aleatório	730	0.9991	2.5771	0.9969	4.5652	0.9956
BitLord	Sequencial	604	0	0	0	0	0
Deluge	Aleatório	1564	0.9999	2.5829	0.9992	4.5713	0.9970
BitTorrent	Aleatório	438	0.9999	2.5846	0.9998	4.5685	0.9964
BitTorrent	Sequencial	261	0.9999	2.5813	0.9986	4.5142	0.9845
μTorrent	Aleatório	1212	0.9990	2.5805	0.9982	4.5722	0.9972
μTorrent	Sequencial	1511	0.9993	2.5824	0.9990	4.5665	0.9959
Transmission	Aleatório	1567	0.9965	2.5720	0.9950	4.5492	0.9922
KTorrent	Aleatório	490	0.9998	2.5811	0.9985	4.5688	0.9964
KTorrent	Sequencial	973	0.0465	0.0967	0.0374	0.1471	0.0320
qBitTorrent	Aleatório	1550	0.9999	2.5840	0.9996	4.5799	0.9988
qBitTorrent	Sequencial	726	0.2422	0.5184	0.2005	0.7819	0.1705
Vuze	Aleatório	1570	0.5897	1.4082	0.5447	2.3105	0.5039
Vuze	Sequencial	1570	0.5539	1.3041	0.5045	2.1181	0.4619
Xunlei	Aleatório	3139	0.4338	1.0305	0.3986	1.7192	0.3749
Xunlei	Sequencial	1568	0.1975	0.4252	0.1645	0.6802	0.1483

Tabela 3.3: Entropia dos clientes BitTorrent em ambiente público

Cliente	Modo	Nº Msg	H(2)	H(3)	H(3)Norm	H(4)	H(4)Norm
Bitcomet	Aleatório	852	0.9952	2.5671	0.9931	4.5123	0.9841
Bitcomet	Sequencial	928	0.5647	1.2989	0.5025	2.0840	0.4545
BitLord	Aleatório	880	0.9991	2.5751	0.9962	4.5558	0.9936
BitLord	Sequencial	881	0.5306	1.2241	0.4735	1.9516	0.4256
Deluge	Aleatório	881	0.9999	2.5845	0.9998	4.5771	0.9982
BitTorrent	Aleatório	878	0.9999	2.5825	0.9990	4.5701	0.9967
BitTorrent	Sequencial	903	0.9990	2.5791	0.9977	4.5649	0.9956
μTorrent	Aleatório	880	0.9997	2.5828	0.9991	4.5682	0.9963
μTorrent	Sequencial	879	0.9997	2.5813	0.9986	4.5610	0.9947
Vuze	Aleatório	730	0.7386	1.8200	0.7041	3.0221	0.6591
Vuze	Sequencial	858	0.5812	1.3920	0.5385	2.2733	0.4958

Eis que para cada cliente, considerando o mesmo conjunto de mensagens, foi calculada a evolução do valor da entropia de permutação no tempo, conforme propõem a Figura 3.3. O eixo y indica o valor da entropia por permutação, é importante apontar que os valores plotados nos gráficos são normalizado para H(3) e H(4).

Cada ponto do eixo das abscissas representa a chegada de uma mensagem do tipo

"have", e por isso quando havia uma diferença muito grande do número de mensagens entre o modo aleatório e o modo sequencial, para apresentação dos gráficos, foi considerado o menor conjunto, e o corte foi feito respeitando a sequência temporal.

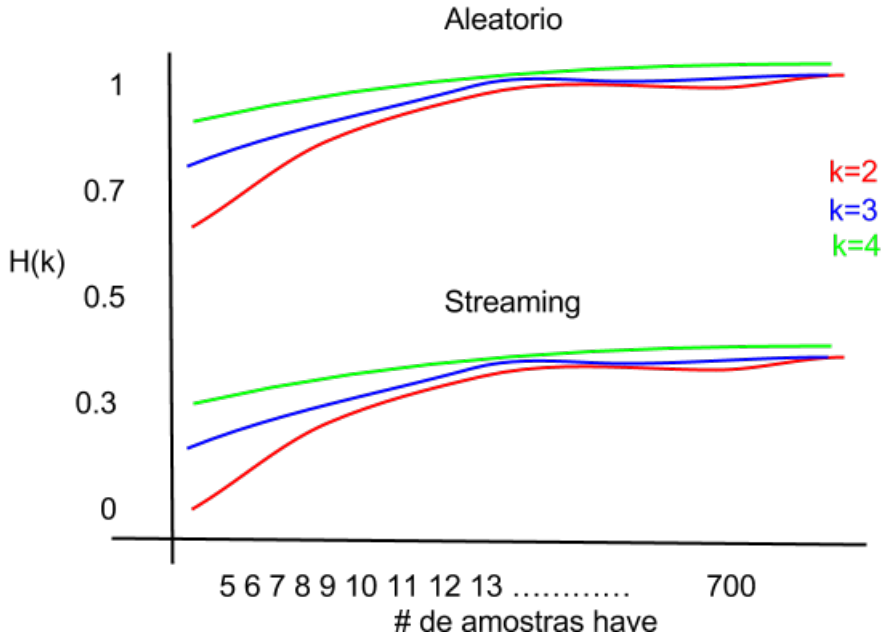

Figura 3.3: Exemplo da evolução dos valores de entropia no tempo

- **Bitcomet**

  É possível perceber como a característica desse cliente, de iniciar o *download* dos primeiros blocos e dos últimos blocos descrita no Capítulo 2, se apresenta no valor da entropia por permutação transcrita na Tabela 3.2 e na Tabela 3.3, pois o modo aleatório tem um valor ligeiramente menor que outros clientes no mesmo modo, bem como a valor do modo sequencial também é claramente maior que clientes totalmente sequenciais, como o BitLord.

  Essa informação também pode ser interpretada a partir da Figura 3.4 que mostra a evolução do valor de entropia considerando a sequência de chegada das mensagens, em que as curvas só tomam um formato consistente depois da chegada de um certo número de mensagens. Esse mesmo comportamento pode ser observado em ambiente real, apresentado na Figura 3.5.

  Todos os experimentos desse trabalho feitos com esse cliente usaram o Bitcomet versão 1.37, publicada em dezembro de 2013, instalado em máquina com sistema operacional Windows 7.

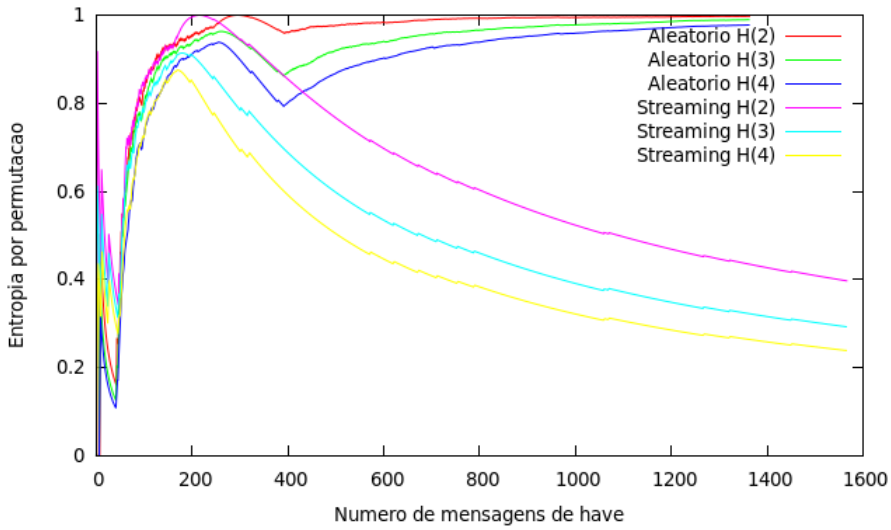

Figura 3.4: Evolução da entropia por permutação do cliente BitComet em ambiente controlado

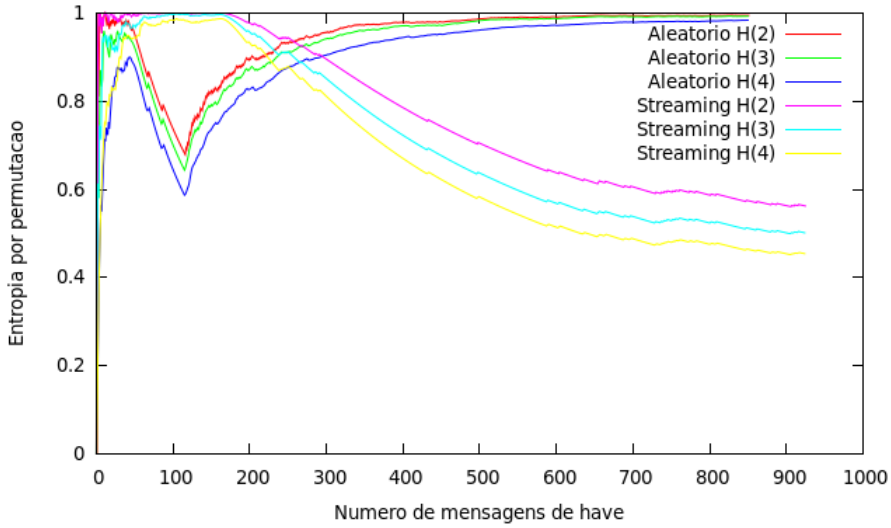

Figura 3.5: Evolução da entropia por permutação do cliente BitComet em enxame público

- **BitLord**

  Essa ferramenta é ideal para fazer testes de extremos, pois quando seu modo *streaming* é ativado, o cliente pede absolutamente em sequência. Isso é comprovado pela evolução do *bitfield*, pelas mensagens de *have*, e pelos seus valores de entropia. A Figura 3.6 apresenta o gráfico da evolução da entropia de permutação, as linhas da entropia por permutação H(2), H(3) e H(4) do modo *streaming* estão no eixo x.

  Trecho da sequência de mensagens have recebidas do cliente BitLord: ..., 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150,151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, ...

No entanto, em um ambiente com baixa disponibilidade, onde nem todos blocos estão disponíveis, o valor de entropia será naturalmente maior. Isso pode ser observado comparando a Figura 3.6, que apresenta a entropia em ambiente controlado, com a Figura 3.7, que apresenta a entropia em ambiente real.

Todos os experimentos desse trabalho feitos com esse cliente usaram o BitLord versão 2.3.2, publicada em agosto de 2013, instalado em máquina com sistema operacional Windows 7.

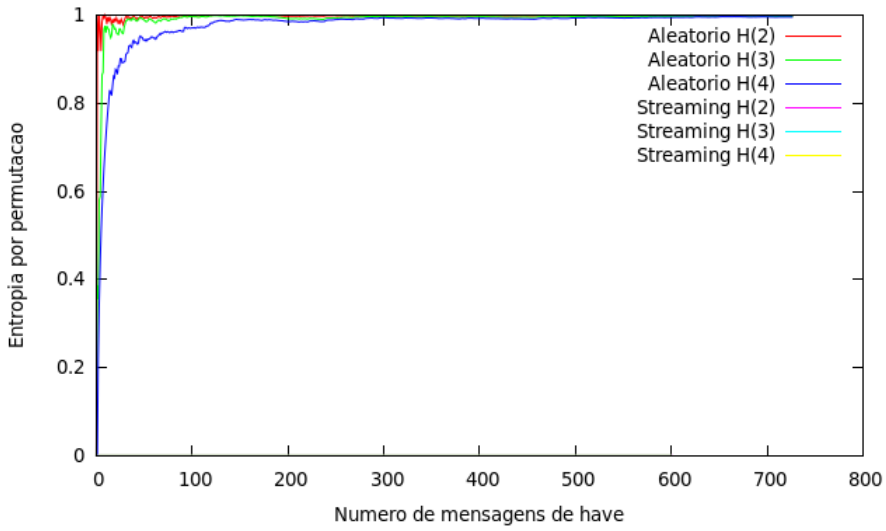

Figura 3.6: Evolução da entropia por permutação do BitLord em ambiente controlado

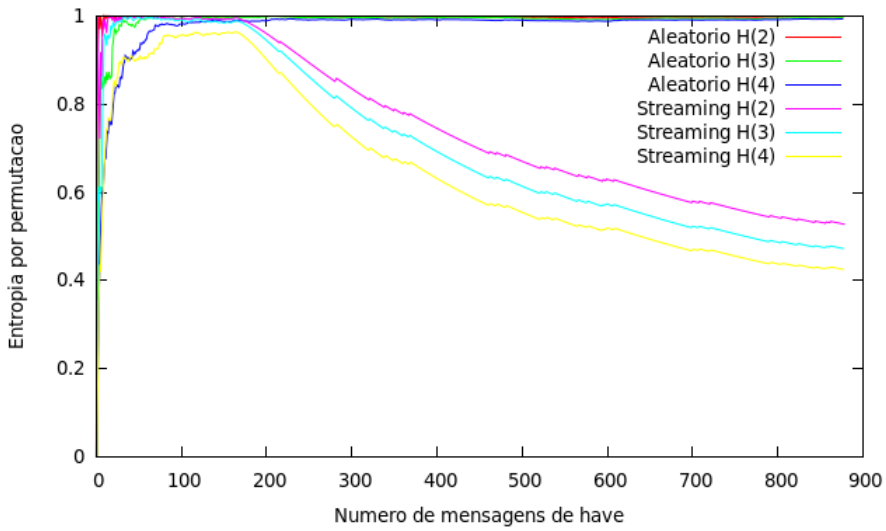

Figura 3.7: Evolução da entropia por permutação do BitLord em enxame público

- **Deluge**

  O cliente Deluge não possui modo *streaming*, apenas modo aleatório, e os valores do experimento refletem a natureza uniformemente distribuída da sua forma de requisitar trechos. Como é possível verificar nas Figuras 3.8 e na Figura 3.9, não existe grande diferença nos valores de entropia do Deluge, independente da disponibilidade, pois esse software opera apenas em modo aleatório.

  Todos os experimentos desse trabalho feitos com esse cliente usaram o Deluge versão 1.3.6 disponível para o sistema operacional Windows 7.

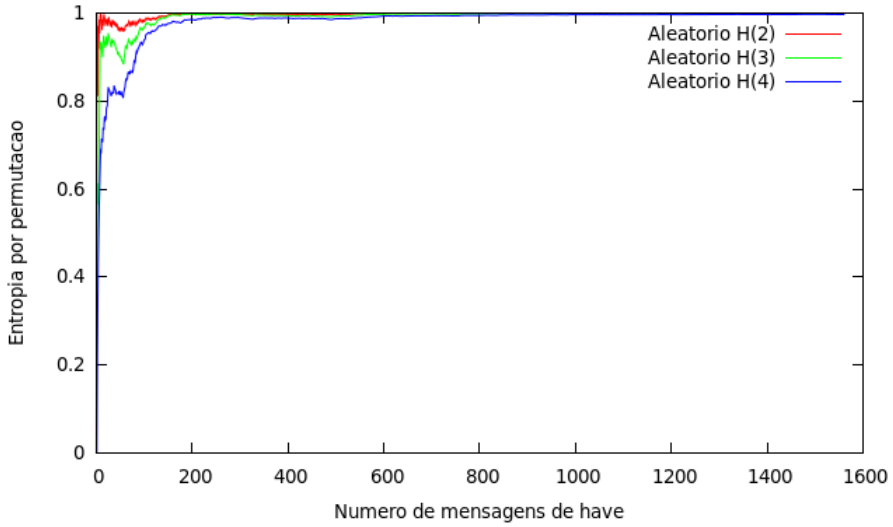

Figura 3.8: Evolução da entropia por permutação do cliente Deluge em ambiente controlado

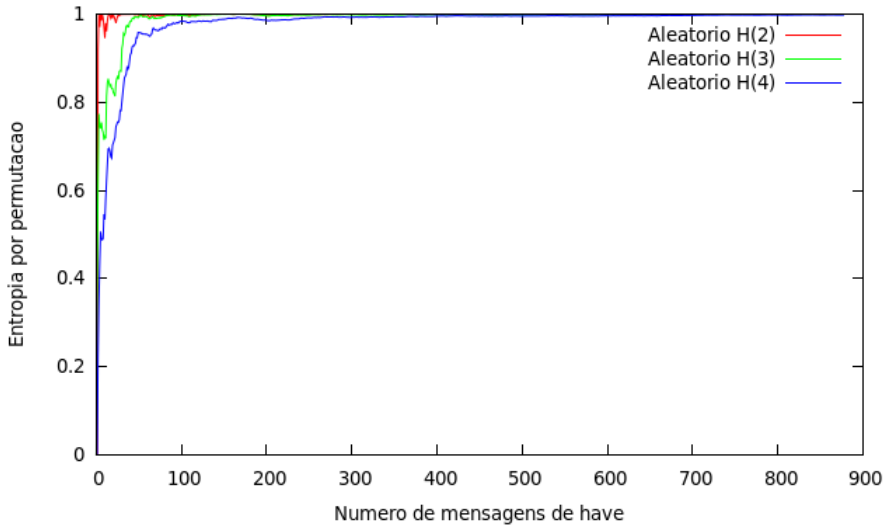

Figura 3.9: Evolução da entropia por permutação do cliente Deluge em enxame público

- **BitTorrent**

  A versão limitada de *streaming* implementada por esse cliente, permite apenas poucos segundos de *download* sequencial. Isso reduz o valor da entropia, se comparado ao modo aleatório (que aparece a partir da terceira casa decimal), mas não é suficiente para ser significativo, embora até 50 mensagens fique clara a diferença dos dois modos. Essa diferença é observável na Figura 3.10 e Figura 3.11.

  Para esse experimento o BitTorrent versão 7.9 foi instalado em uma máquina com Windows 7.

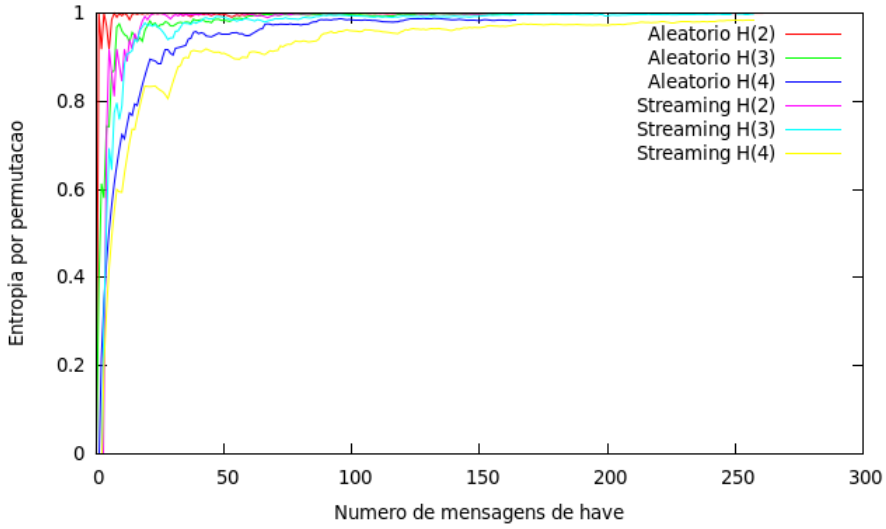

Figura 3.10: Evolução da entropia por permutação do cliente BitTorrent em ambiente controlado

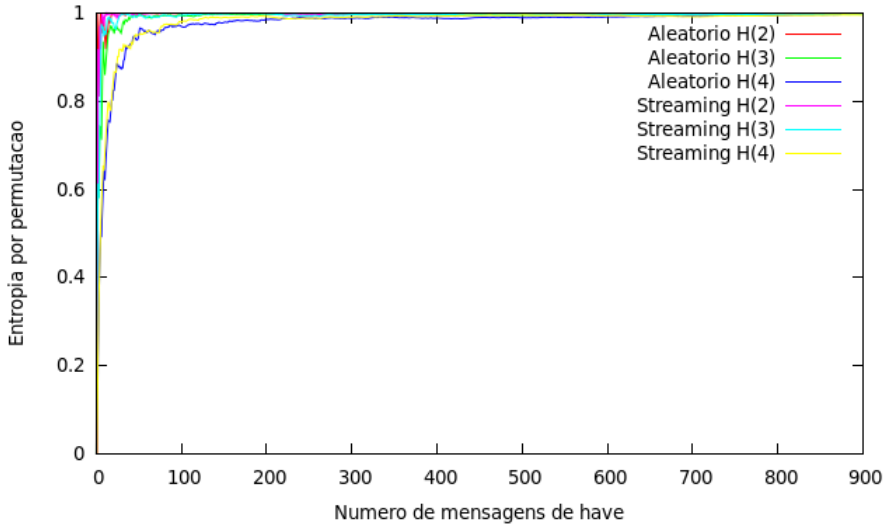

Figura 3.11: Evolução da entropia por permutação do cliente BitTorrent em enxame público

- **μTorrent**

  Devido a semelhança entre o μTorrent e o BitTorrent já descrita nos dois capítulos anteriores, a leitura dos valores de entropia por permutação desses clientes é a mesma. Inclusive a entropia é ligeiramente menor em modo sequencial até 50 mensagens, como pode ser observada na Figura 3.12 e na Figura 3.13, que representam a evolução da entropia por permutação em ambiente controlado e real, respectivamente. Para esse experimento o μTorrent 3.4 foi instalado em uma máquina com Windows 7.

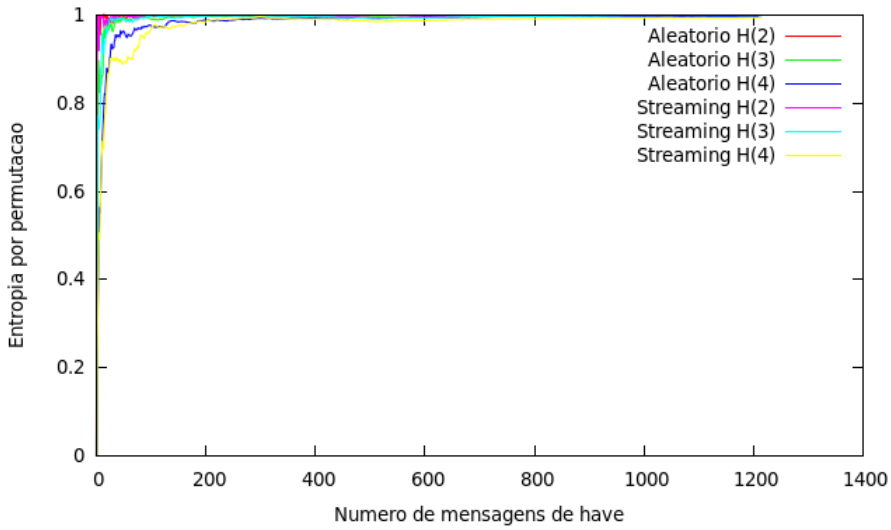

Figura 3.12: Evolução da entropia por permutação do cliente μTorrent em ambiente controlado

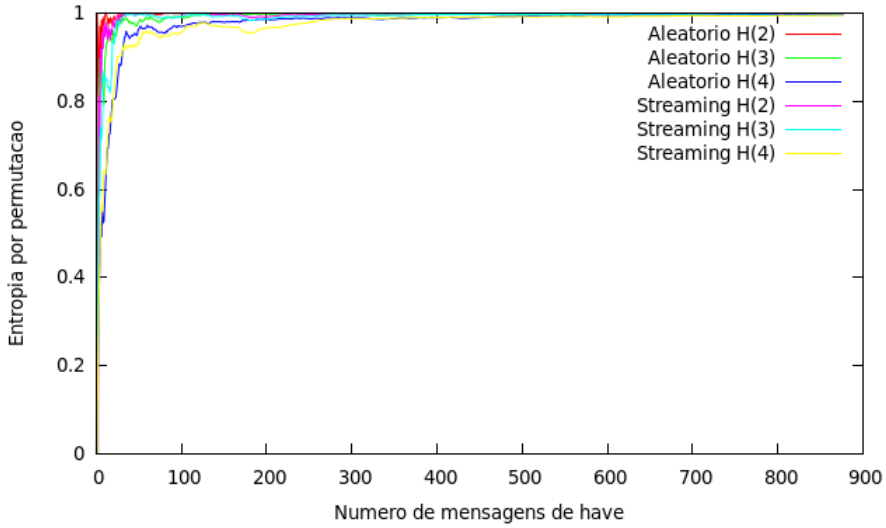

Figura 3.13: Evolução da entropia por permutação do cliente μTorrent em enxame público

- **Transmission**

  Assim como o cliente Deluge, o Transmission não possui modo *streaming*, apenas modo aleatório, e os valores do experimento refletem a natureza uniformemente distribuída da sua forma de requisitar trechos. A evolução da entropia por permutação do Transmission é apresentada na Figura 3.14.

  Para esse experimento foi usado Transmission 2.8.2 padrão de uma máquina com Linux Ubuntu 13.10.

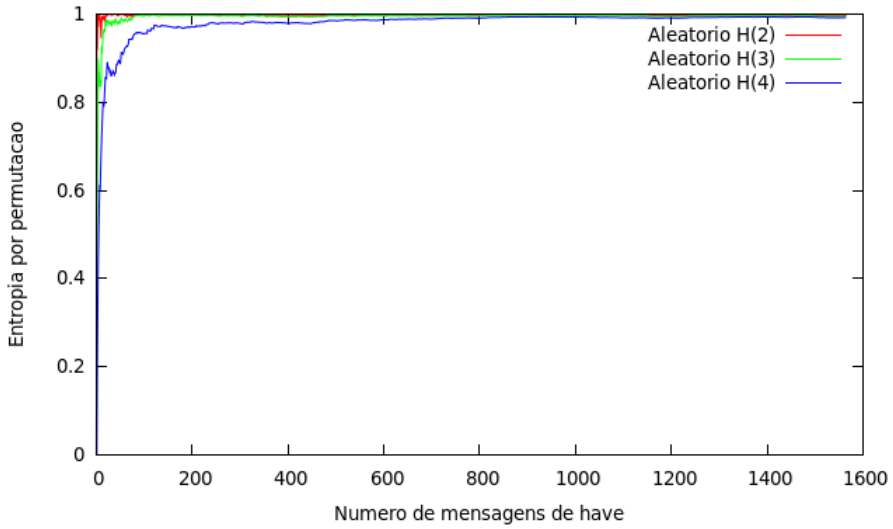

Figura 3.14: Evolução da entropia por permutação do cliente Transmission em ambiente controlado

- **KTorrent**

  A operação de *download* dessa aplicação não é totalmente sequencial, como mostra sua entropia, mas não apresenta requisições uniformemente distribuídas. Logo, deve fazer uso de alguma implementação de janela deslizante, ou houve entrega atrasada de alguma mensagem. A evolução da entropia por permutação em ambiente controlado do KTorrent é apresentada na Figura 3.15.

  Para esse experimento o KTorrent versão 4.3.1 foi instalado em uma máquina com Linux Ubuntu 13.10.

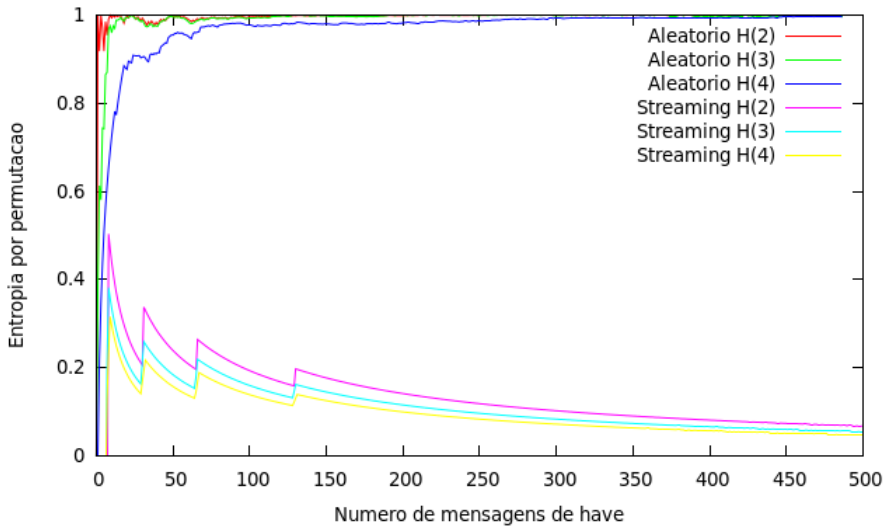

Figura 3.15: Evolução da entropia por permutação do cliente KTorrent em ambiente controlado

- **qBitTorrent**

  Essa versátil ferramenta, que foi usada também como *tracker* durante os experimentos desse estudo, e possui um modo *streaming* com características semelhantes ao Ktorrent, como pode ser observado na Figura 3.16.

  Para esse experimento o qBitTorrent versão 3.0.9 foi instalado em uma máquina com Linux Ubuntu 13.10 e versão 3.1.9.2 para Windows 7.

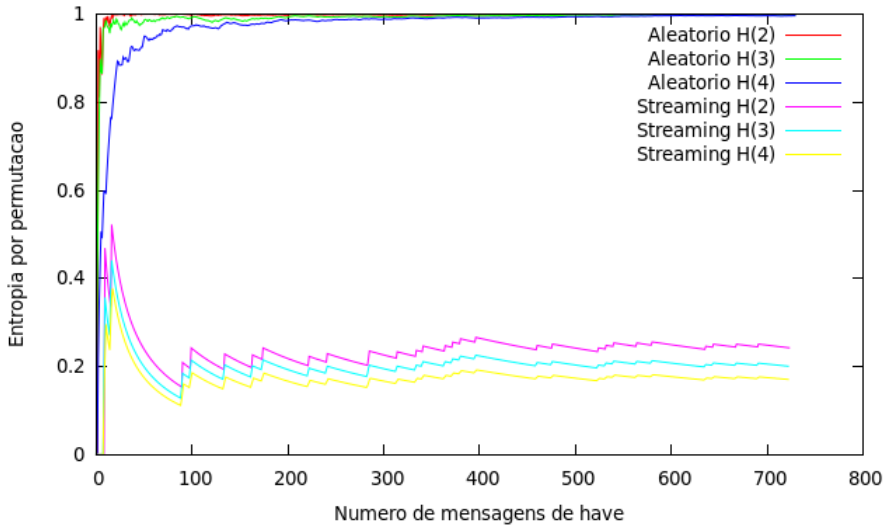

Figura 3.16: Evolução da entropia por permutação do cliente qBitTorrent em ambiente controlado

- **Vuze**

  Não fica claro no site, nem na documentação, que esse software usa alguma variação na política de requisição de trecho, mas o valor de entropia desse cliente em modo aleatório é consideravelmente baixo, quando comparado aos demais. Esse fato leva a crer que existe mudança na implementação do protocolo. No entanto, vale ressaltar que essa mudança não está associada à exibição do conteúdo durante o tempo de carregamento, pois a visualização do arquivo é uma *feature* disponível apenas na versão paga desse programa.

  A entropia por permutação do Vuze e Vuze plus em ambiente público tem uma clara separação, como mostra a Figura 3.18. Porém a entropia em ambiente de alta disponibilidade, apresentada na Figura 3.17, exige um número maior de mensagens para ser identificada.

  Durante esse estudo foi usada a versão Vuze 5.3.0.0, disponibilizada em fevereiro de 2014, e a versão do *plugin* +VuzePlus compatível com a versão ordinária desse *software*.

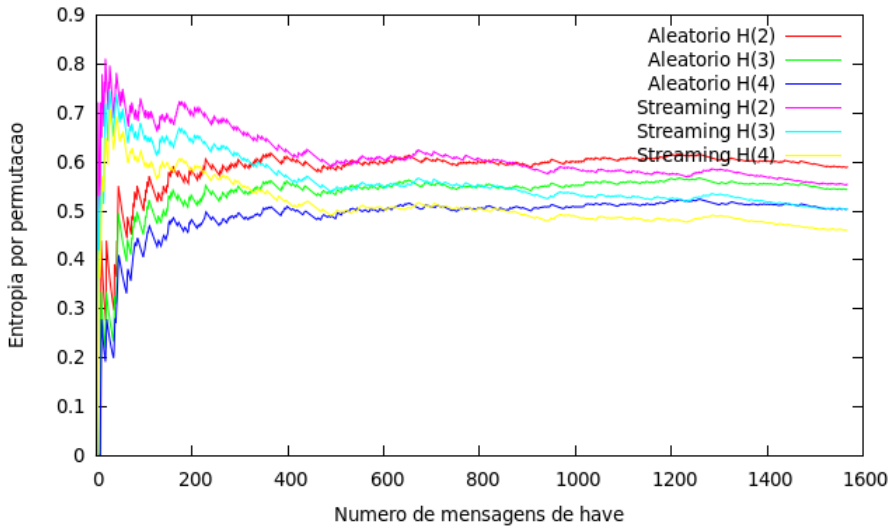

Figura 3.17: Evolução da entropia por permutação do cliente Vuze e Vuze plus em ambiente controlado

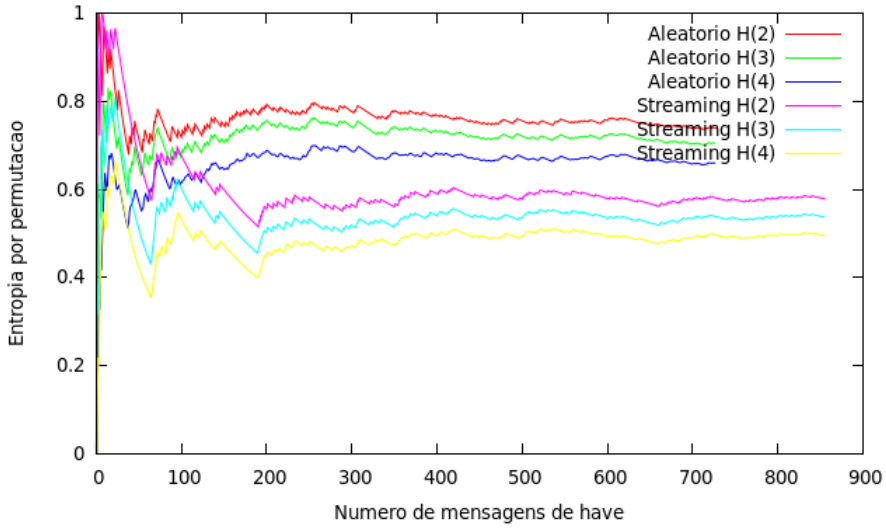

Figura 3.18: Evolução da entropia por permutação do cliente Vuze e Vuze plus em enxame público

- **Xunlei**

  Esse programa é todo em chinês, e possui um pacote que o traduz parcialmente para o idioma inglês. No entanto essa tradução só funciona na versão 5.8.14, que data o ano 2010. Por isso, para esse estudo foi usada a versão mais atualizada dentre as estáveis, de número 7.9.13 para sistema operacional Windows 7, disponível em fevereiro de 2014, que é totalmente em mandarim.

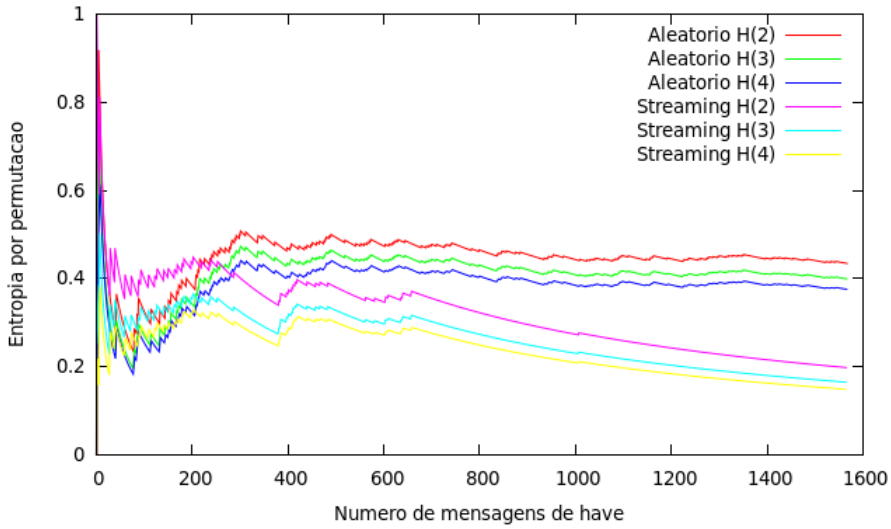

Figura 3.19: Evolução da entropia por permutação do cliente Xunlei em ambiente controlado

- **Popcorn Time**

  Como a ferramenta só disponibiliza ao usuário um conjunto limitado e pré-determinado de enxames, não foi possível realizar o experimento em ambiente controlado.

  Para entender o funcionamento desse programa, foi selecionado um enxame com poucos *peers* no portal YTS. Conectou-se a ferramenta espiã com intenção de monitorar as trocas de mensagens dentro da rede e então conectou-se o Popcorn Time a esse enxame.

  Durante os testes realizados com essa ferramenta, foi observado, que ela aparentemente não faz *upload*, pois foi verificado que a conexão era reiniciada regularmente, o que levava a rede a acreditar que se tratava de um novo usuário, e com isso fazia com que o usuário do Popcorn Time recebesse os benefícios da política altruísta a cada nova conexão. Além disso, o cliente Popcorn Time também não enviou mensagem nenhuma de have durante toda a reprodução, ou ao menos uma que o espião tivesse registrado.

3.4 Conclusão parcial

Ao longo desse capítulo foi apresentado o programa cliente BitTorrent espião, responsável pela coleta de dados nos enxames. Além desse *software*, foi apresentado como, através das mensagens capturadas por esse aplicativo, é possível classificar um *peer* em sequencial ou aleatório, usando os conceitos entropia por permutação.

Na última parte, essa arquitetura de monitoramento foi posta em prática ao capturar em ambiente controlado e ambiente real dados base de diversos clientes. Esses dados apresentam uma clara diferença do comportamento *streamer* para o comportamento aleatório ordinário. E esses valores serão usados como referência futura na classificação dos membros dos enxames BitTorrent.

Foi também observado que os valores de entropia considerando tamanhos diferentes de permutação são muito parecidos, o que sugere que os clientes BitTorrent estudados não implementam janelas deslizantes. Então, defido a complexidade do cálculo, é recomendável usar entropia com valor baixo de permutação.

Capítulo 4

Detecção de streamers em enxames reais

Nesse capítulo serão apresentados os experimentos realizados para detectar a presença de usuários "view as you download" e seu comportamento em enxames populares. Para tal, será descrito o processo de monitoramento, bem como os dois tipos de experimentos realizados.

Disclaimer: embora as ferramentas desenvolvidas por esse trabalho tenham se conectado às redes de compartilhamento de arquivos cujo conteúdo é protegido por direitos autorais, nenhum dos dados transferidos foram usados para fins comerciais, e serviram apenas para compreender o funcionamento desses ambientes.

4.1 Descrição do processo de monitoramento com a ferramenta Espiã

Para realizar os experimentos com enxames populares foi usada uma máquina Linux Ubuntu 13.10. O computador contava com o cliente espião (Transmission customizado) descrito no Capítulo anterior na Seção 3.3.1, ligado à Internet para monitorar a rede.

O espião captura a cada mensagem enviada pelos vizinho os seguintes dados:

1. IP: IP do *peer* vizinho, para identificar e diferenciar cada *peer*;
2. PeerID: Identificador único criado pelo espião para identificar seus vizinhos por sessão. Garante sua identificação mesmo que o *peer* esteja atrás de um NAT;
3. Ordem: Representa a ordem de chegada da mensagem, é fundamental para o cálculo da entropia por permutação;

4. Enxame: Identificador único criado pelo espião para identificar o enxame quando o experimento considerar mais de um arquivo .torrent por vez;

5. Cliente: Identifica o *software* cliente BitTorrent usado pelo *peer* vizinho.

6. *Bitfield*: Vetor de *bits* que representa blocos recebidos pelo *peer* remetente

   ou

6. Bloco: Bloco recebido

Embora tanto a ferramenta espiã quanto o cálculo de entropia por permutação possam ser usados para arquivos de qualquer natureza, esse trabalho objetiva compreender a propagação de arquivos de mídia por *streaming*. Por isso para os experimentos foram considerados apenas arquivos de vídeo, pois existe uma expectativa de que encontrar mais *streamers* nessa classe de enxame.

4.2 Análise de Enxames Populares

Como já apresentado anteriormente, uma das grandes vantagens de usar as redes P2P BT como plataforma *streaming* de mídia sob demanda é explorar a quantidade massiva de conteúdo e sua popularidade, pois quanto mais usuários a rede possui, melhor e mais rápida será a difusão da mídia.

Então, com intuito de descobrir se existem usuários "view as you download" em enxames públicos foi executado por 90 dias um experimento com o seguinte funcionamento:

1. A cada duas horas eram copiados os links magnéticos dos 100 enxames de vídeo mais populares do PirateBay no link `http://thepiratebay.se/top/200`;

2. Esses enxames eram carregados no cliente espião;

3. Ao fim do período de duas horas, os registros eram removidos do monitor, os arquivos e metadados apagados e os registros de troca de mensagem armazenados por sessão de cada enxame;

4. O processo de monitoramento reiniciava.

Além do ciclo descrito acima, o *script* responsável pela automatização do monitoramento verificava o funcionamento do espião a cada 10 minutos, para reiniciar o monitor

e seus *downloads* se necessário. A Figura 4.1 ilustra a arquitetura de monitoramento desenvolvida.

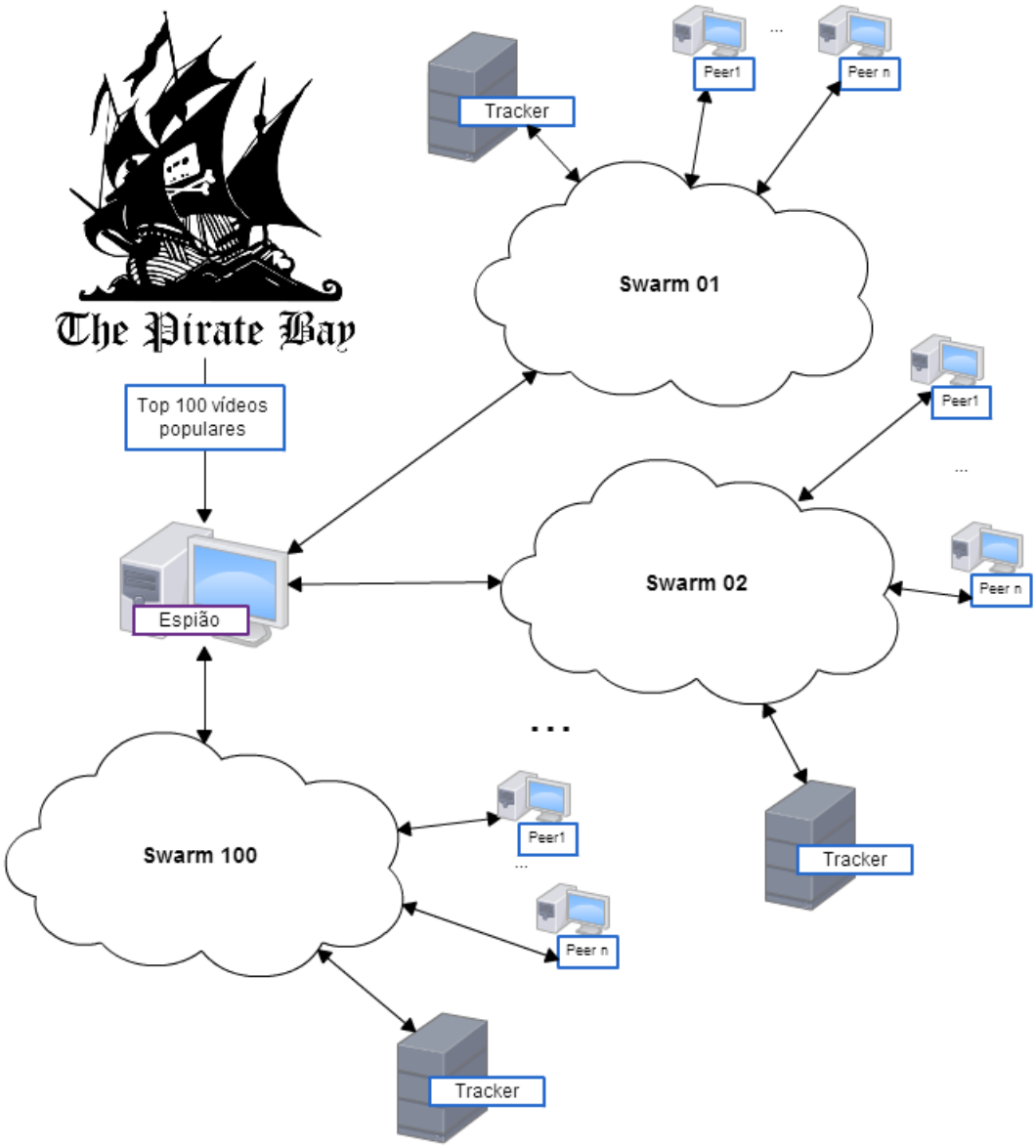

Figura 4.1: Esquema do experimento em enxames Populares do PirateBay

É importante ressaltar que, a lista de enxames populares do PirateBay (que é atualizada constantemente) não muda dramaticamente. Logo, a cada repetição da rotina o conjunto de enxames monitorados certamente possui grande interseção com o conjunto monitorado anteriormente.

O experimento sofreu poucos problemas de queda de luz e Internet, e o monitoramento foi interrompido por pouco mais de 2 dias e 20 horas ao longo de todos os dias de duração, sendo que esse tempo não foi contínuo.

O espião se conectou a mais de 1 milhão e meio de *peers*, pertencentes a mais de 100 mil enxames. Gerando um pouco menos de 40 gigabytes de registro de mensagens. Além disso, o tempo total para processar a evolução de entropia por permutação de todos esses *peers* chegou a 5 dias.

Outra informação pertinente ao estudo é quanto a popularidade dos programas clientes usados pelos usuários de enxames populares do The Pirate Bay. Esses dados são apresentados na Tabela 4.1 e ilustrados na Figura 4.2. Esse resultado foi usado na escolha dos clientes descritos nos capítulos anteriores.

Tabela 4.1: Clientes Populares nos enxames populares do PirateBay

Cliente	**%**	**Nº Peers**
Utorrent	58,02%	764185
Transmission	15,37%	202439
Vuze	11,03%	145285
BitTorrent	7,42%	97761

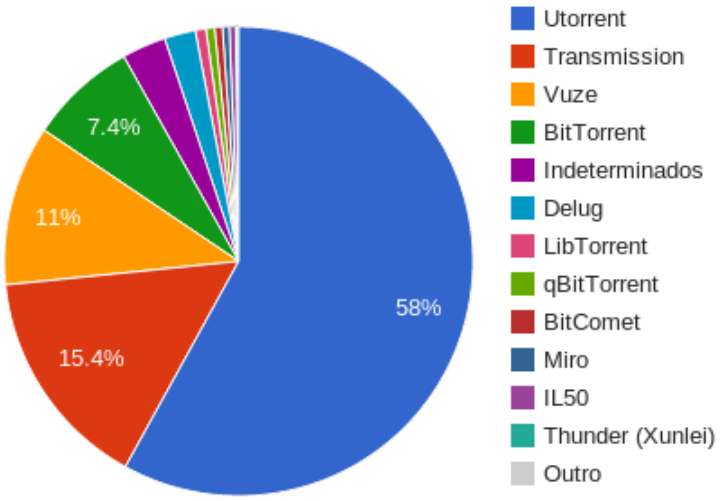

Figura 4.2: Clientes mais usados nos enxames populares do PirateBay

4.2.1 Análise em duas etapas

A partir dos registros coletados, foi calculada a entropia por permutação e a evolução da entropia por permutação considerando a chegada de novas mensagens para H(2), H(3) e H(4) de cada um dos *peers* que se conectaram ao espião. *Peers* com menos de 20 mensagens não foram levados em consideração por esse estudo, pois com um número menor de registros não é possível detectar a tendência da evolução da entropia, como

pode ser observado nos gráficos da evolução da entropia por permutação apresentados no capítulo anterior.

Durante o experimento, dos mais de 1,3 milhões *peers* avaliados, foram detectados apenas 14.443 *peers* cuja entropia por permutação era menor que 0,8. Esse valor de entropia por permutação foi escolhido para garantir que os dados dos *peers* utilizando clientes como o Bitcomet, que possui em modo *streaming* um valor final de entropia de 0.7, fossem levados em conta nessa avaliação.

14.443 *peers*, cerca de 1,10%, apresentaram um valor de entropia por permutação menor que 0,8, porém ainda cabe avaliar quem são esses *peers* para determinar se de fato são *streamers*. Por isso é importante esclarecer quais clientes esses *peers* estavam usando, e quatro programas se destacam:

1. μtorrent -> 5212
2. Vuze/Azure -> 6067
3. BitTorrent -> 678
4. Xunlei -> 437
5. Outros -> 2040

A Tabela 4.2 amplia o recorte, e apresenta o número total de usuários percebidos no experimento usando um dos 12 programas clientes BitTorrent que possuiam algum registro de entropia por permutação abaixo de 0,8. Essa mesma tabela apresenta ainda a porcentagem de prováveis usuários *streamers* em relação a quantidade de usuários daquele mesmo cliente.

Feito o primeiro corte em 0,8, apresentado na Tabela 4.2, é necessário validar o valor de entropia por permutação dos *peers* levando em consideração o valor base de cada cliente apresentado no capítulo anterior.

O primeiro cliente é o popular μtorrent, e devido as semelhanças, podemos considerar os seus 1873 junto dos 262 do cliente BitTorrent. É surpreendente que ambos estejam entre os clientes populares dentre os usuários em modo sequencial, já que suas versões atuais permitem apenas uma versão muito limitada de *streaming*.

Os baixos valores de entropia por permutação de ambos leva a concluir que seus *peers* usavam versões que permitem configurar parâmetros do modo sequencial, como na versão

Tabela 4.2: Total de *peers* por cliente com entropia por permutação abaixo de 0.8

Cliente	**Nº Peers**	**Corte H2 0.8**	**%**
Vuze	145285	6067	4,18%
Transmission	202439	12	0,01%
Thunder (Xunlei)	2346	437	18,63%
qBitTorrent	7834	19	0,24%
KTorrent	62	0	0,00%
Deluge	28246	45	0,16%
BitTorrent	97761	687	0,70%
BitLord	181	0	0,00%
Bitcomet	7110	29	0,41%
Utorrent	764185	5212	0,68%
IL50	5791	89	1,54%
JS09	45	19	42,22%
Indeterminados	39976	1709	4,28%

3.0 do μtorrent apresentada anteriormente. Os valores de entropia (em modo aleatório e no modo *playback)* da versão atual (3.4) são sempre altos (acima de 0,9).

Esse fato pode também ser observado na Figura 4.3 ao perceber que o número de usuários com um determinado valor de entropia é aproximadamente o mesmo até o corte em 0,8. Essa ilustração apresenta o valor ordenado das entropias por permutação dos *peers* com μtorrent.

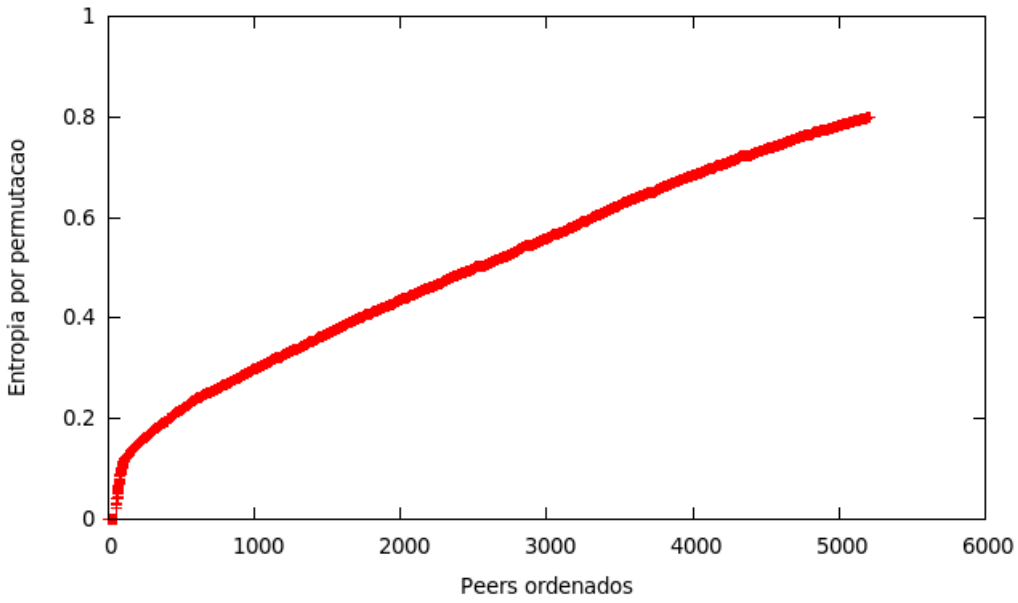

Figura 4.3: Valores de entropia dos *peers* com cliente μTorrent até 0.8

Esse cenário só poderia ser alcançado se houvesse uma configuração de parâmetros como mostra a Figura 2.11, posto que na Figura 4.4, em que são apresentados todos os valores de entropia dos *peers* que usam o μtorrent, essa dispersão "uniforme" nem mesmo é perceptível.

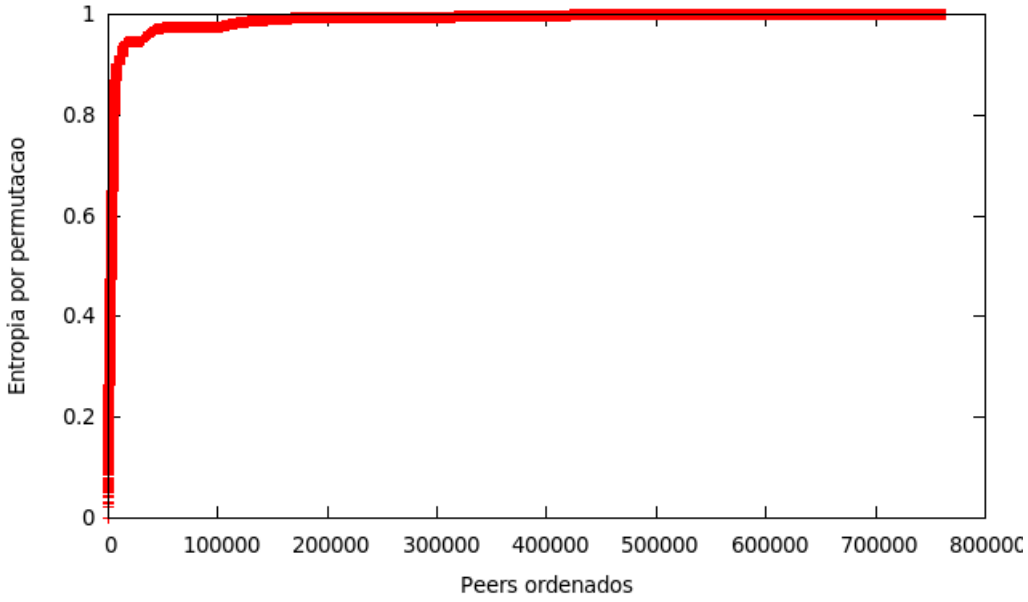

Figura 4.4: Valores de entropia dos *peers* com cliente μTorrent

O segundo conjunto é formado de *peer* que usam o Vuze. No entanto, o valor da entropia por permutação geral em modo aleatório do Vuze é baixa quando comparada a de outros *softwares*, pois fica entre 0,4 a 0,6. Levando isso em consideração, apenas 657 dos 6067 usuários poderiam estar de fato explorando a característica "view as you download".

No cliente chinês, Thunder Xunlei, dos 437 *peers* com entropia por permutação abaixo de 0.8, 367 apresentam um valor compatível com o valor geral de entropia por permutação desse cliente em modo de *streaming*.

Além desses 4 programas, vários outros tipos de clientes foram detectados, mas nenhum com número expressivo de usuários, inclusive com valores de entropia baixos.

Logo, dos mais de 1 milhão de *peers*, é possível afirmar que apenas 6971, cerca de 0,53% de todos esses *peers*, são usuários "view as you download". Esse resultado confere com a matéria publicada pelo portal Variety.com [51] que afirma que a ferramenta Popcorn Time, descrita nos Capítulos anteriores, ainda tem muito pouco uso quando comparada a outros clientes. Muito embora esse estudo do grupo Excipio não mensure se os outros clientes estão em modo *streaming* e sabendo que o Popcorn Time não se conecta ao PirateBay,

Tabela 4.3: Avaliação da existência de *streamers* em duas etapas

Cliente	**Nº Peers**	**Corte H2 0.8**	**%**	**Streamers**	**%**
Vuze	145285	6067	4,18%	657	0,45%
Transmission	202439	12	0,01%	1	0,00%
Thunder (Xunlei)	2346	437	18,63%	376	16,03%
qBitTorrent	7834	19	0,24%	9	0,11%
KTorrent	62	0	0,00%	0	0,00%
Deluge	28246	45	0,16%	0	0,00%
BitTorrent	97761	687	0,70%	687	0,70%
BitLord	181	0	0,00%	0	0,00%
Bitcomet	7110	29	0,41%	29	0,41%
Utorrent	764185	5212	0,68%	5212	0,68%
IL50	5791	89	1,54%	0	0,00%
JS09	45	19	42,22%	0	0,00%
Indeterminados	39976	1709	4,28%	0	0,00%

são informações relevantes para afirmar que o uso desse tipo de ferramenta/*feature* ainda não é amplamente explorado, respondendo a um dos questionamentos propostos por esta dissertação.

Uma outra forma de visualizar esses dados é a partir dos enxames. Esses 6971 *peers* em modo sequencial pertencem a pouco mais de 3 mil enxames. O número pode ser menor, mas como o espião precisa reconectar a cada duas horas de monitoramento, essa informação não pode ser obtida nesse contexto (embora seja avaliada no experimento seguinte).

Considerando que o experimento reiniciava a cada duas horas, a exceção de 2 enxames, um com 83 e outro com 26 *peers* em modo sequencial, com uma média de 1,4 *peers* por enxame, podemos afirmar que não existe um horário que atraia usuários a explorar a característica "view as you download" dos clientes BitTorrent. Pois dada uma bateria de duas horas, houve em média 1,4 *peers* conectados a um enxame.

4.3 Análise de Enxame Específico

É válido assumir que usuários sequenciais são pessoas que não desejam esperar para desfrutar do conteúdo. Então, uma hipótese é que esse tipo de indivíduo acessa às redes assim que uma determinada mídia de interesse é disponibilizada na Internet, e possivelmente faria uso de uma *feature* "view as you download".

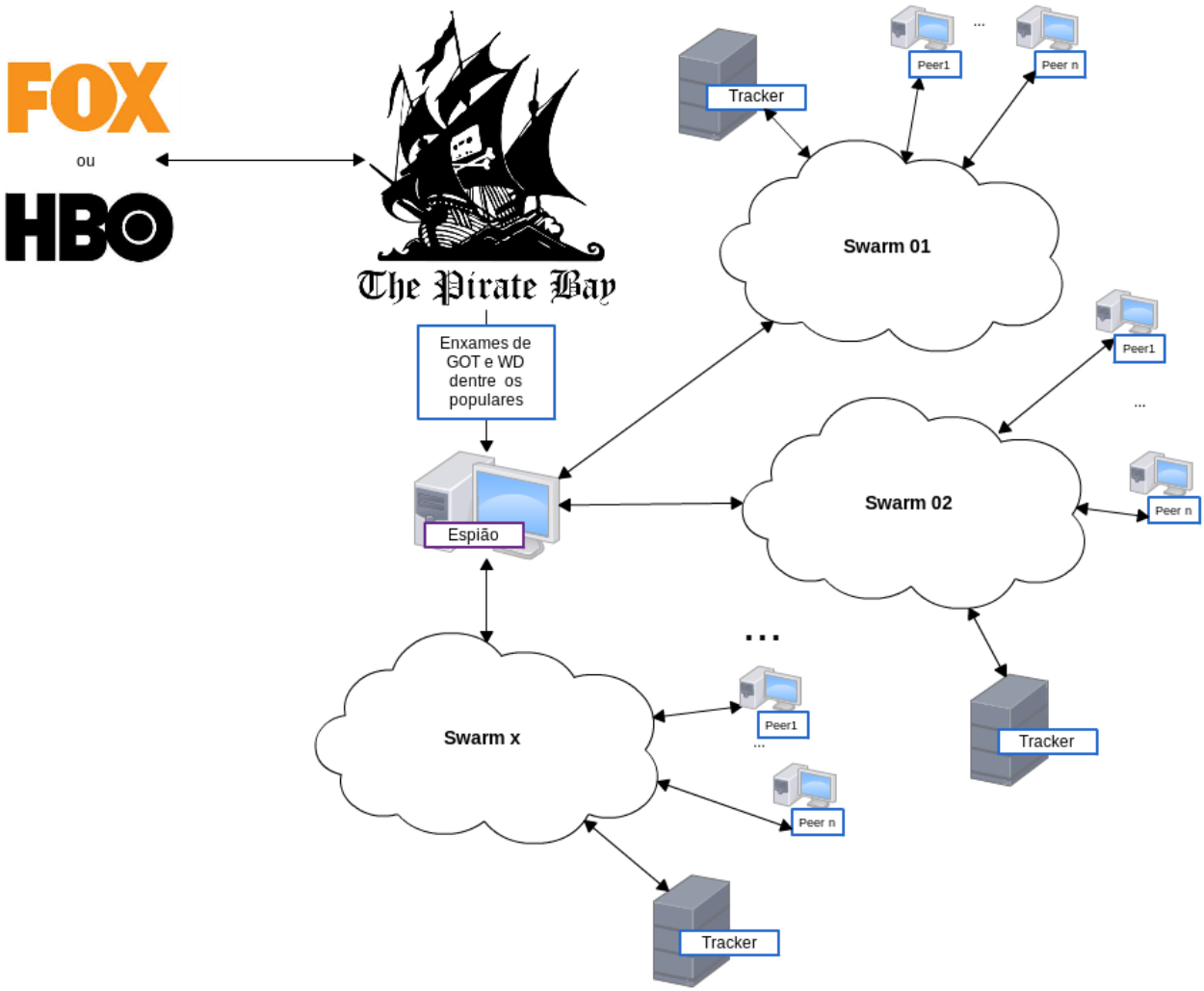

Figura 4.5: Esquema do experimento em enxames específicos

Para responder essa pergunta, esse estudo monitorou por 24 horas, os enxames dos últimos 5 episódios da quarta temporada da série Walking Dead e todos os episódios da quarta temporada de Game of Thrones assim que esses vídeos despontaram na lista de populares do site torrent PirateBay. Cabe aqui informar que a disponibilização dos conteúdos geralmente ocorria nas primeiras 6 horas após o fim do episódio inédito ser exibido pelo canal Fox e pela HBO respectivamente. O espião não se desconectava desses enxames durante todo o tempo, salvo eventual problema de infraestrutura.

O esquema do experimento é ilustrado pela Figura 4.5. O espião faz o mesmo tipo de coleta descrito no experimento anterior. Ambas as séries alcançam os primeiros lugares no ranking do PirateBay, normalmente com mais de uma ocorrência, cada uma com arquivos de diferentes qualidades, por isso no esquema são apresentado mais de um enxame.

Além da popularidade dessa classe de conteúdo nas redes BitTorrent, outro grande motivo para realizar esses experimentos é a percepção da popularidade desse tipo de programa. Essa percepção é tamanha que, segundo a revista Forbes [12], a série Game of Thrones já é a mais pirateada do mundo e o CEO da Time Warner, Jeff Bewke, considera esse grande uso de pirataria como boa publicidade segundo artigo do site Adweek [1].

No experimento que acompanhou a 4ª temporada da série Game of Thrones, a popularidade dos programas clientes usados pelos *peers* são apresentados na Tabela 4.4 e ilustrados na Figura 4.6. Os mesmos dados, porém referentes à série Walking Dead são mostrados na Tabela 4.5 e na Figura 4.7. Esses dados, assim como os coletados no experimento anterior com todos os enxames populares, foram usados nos capítulos anteriores como argumento na escolha de quais programas clientes seriam descritos.

Tabela 4.4: Clientes Populares nos enxames da 4ª temporada de Game of Thrones

Cliente	**%**	**Nº Peers**
Utorrent	50,76%	55697
Transmission	20,35%	22328
Vuze	12,87%	14119
BitTorrent	3,79%	4157

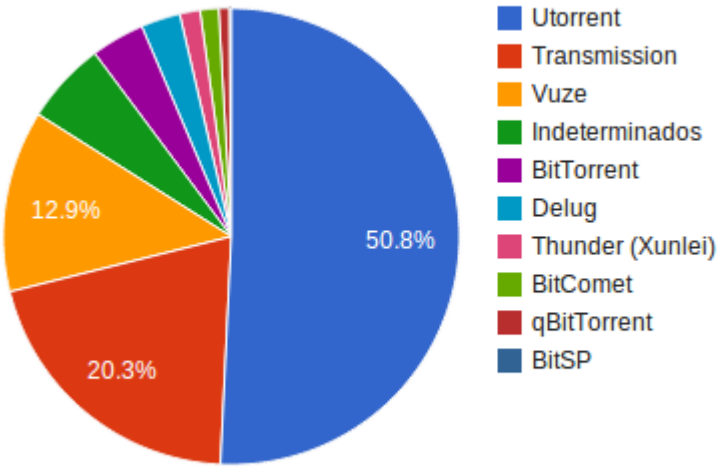

Figura 4.6: Clientes mais usados nos enxames da 4ª temporada de Game of Thrones

Tabela 4.5: Clientes Populares nos enxames dos episódios monitorados de Walking Dead

Cliente	**%**	**Nº Peers**
Utorrent	44,21%	5010
Transmission	25,53%	3356
Vuze	12,20%	1603
BitTorrent	8,77%	1153

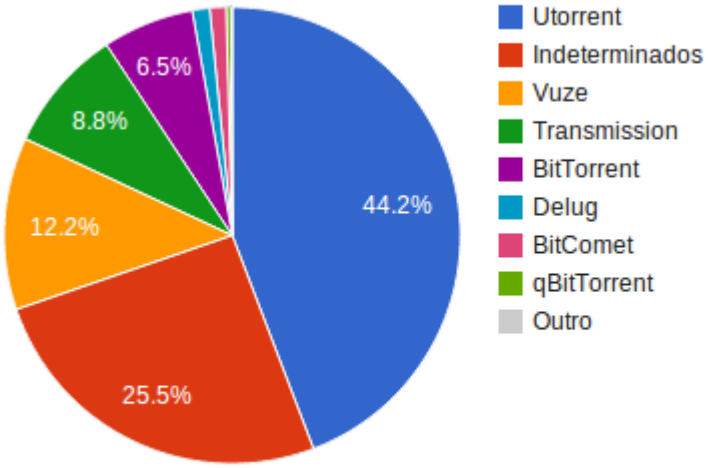

Figura 4.7: Clientes mais usados nos enxames dos episódios monitorados de Walking Dead

4.3.1 Resultados Game of Thrones

O experimento foi realizado durante todas as semanas da quarta temporada da série Game of Thrones. O espião se conectou a 109726 *peers*, e dentre esses 3606 possuem entropia por permutação menor que 0.8, o que representa 3,28% dos *peers*. É um aumento considerável, quando comparado ao valor inicial de 1,10% do experimento realizado com todos os enxames populares do PirateBay.

Porém, como visto no experimento anterior, esse número ainda precisa ser compreendido de forma plena. Sendo necessário conhecer os programas clientes usados por esses *peers*:

1. μtorrent -> 859
2. Vuze/Azure -> 1582
3. BitTorrent -> 79

4. Xunlei -> 1050

5. Outros -> 112

Usando a análise em duas etapas descrita anteriormente, é feita uma avaliação em relação a cada tipo de cliente detectado. Com isso, é possível chegar aos resultados apresentados na Tabela 4.6.

Tabela 4.6: Avaliação da existência de *streamers* em duas etapas nos enxames da 4º temporada de Game of Thrones

Cliente	**Nº Peers**	**Corte H2 0.8**	**%**	**Streamers**	**%**
Vuze	14119	1582	11,20%	131	0,93%
Transmission	22328	1	0,00%	0	0,00%
Thunder (Xunlei)	1577	1050	66,58%	1041	66,01%
qBitTorrent	807	3	0,37%	1	0,12%
Deluge	3055	5	0,16%	1	0,03%
BitTorrent	4157	79	1,90%	79	1,90%
Bitcomet	1436	5	0,35%	5	0,35%
Utorrent	55694	859	1,54%	859	1,54%
Indeterminados	39976	1709	4,28%	0	0,00%

Após o segundo corte, no valor base da entropia por permutação de cada um dos cliente usados, foram detectados 2134 *peers* em modo de reprodução. Isso corresponde a 1,94% dos usuários desses enxames que constituíram vizinhança com o espião.

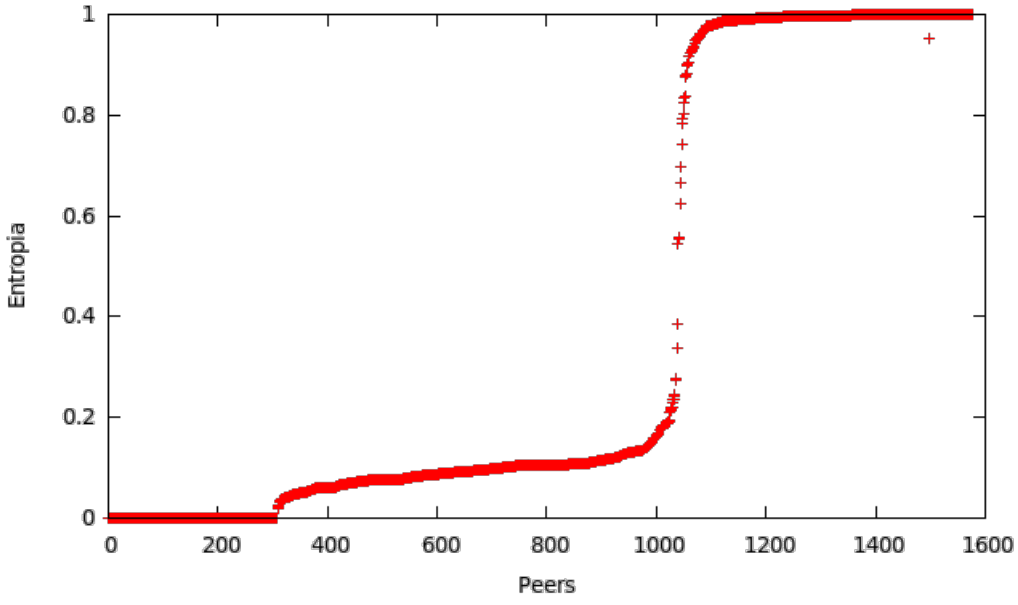

Figura 4.8: Valores ordenados de entropia por permutação dos *peers* com cliente Xunlei na série Game of Thrones

A Figura 4.8 ilustra os valores ordenados de entropia por permutação dos *peers* com cliente Xunlei Thunder. Essa imagem apresenta claramente a separação de usuários *streamers* dos usuários em modo aleatório.

O número de usuários sequenciais, que é proporcionalmente quase quatro vezes maior que a porcentagem de *streamers* nos enxames populares, valida a hipótese de que esses usuários possuem muito mais urgência em desfrutar dessas mídias quando inéditas.

4.3.2 Resultados Walking Dead

O mesmo experimento foi realizado com os últimos 5 episódios da 4ª temporada da série Walking Dead exibidos em 2014. O espião se conectou a 13143 *peers*, e dentre esses apenas 182 possuem entropia por permutação menor que 0.8, o que representa 1,38% dos *peers*. Um valor muito próximo ao 1,10% do experimento realizado com todos os enxames populares do PirateBay.

Após o segundo corte, no valor base da entropia por permutação de cada um dos cliente usados, foram detectados apenas 57 *peers* em modo de reprodução, o que corresponde a 0,43% dos usuários.

4.4 Conclusão Parcial

Foi possível perceber nos experimentos feitos em enxames reais apresentados nesse capítulo que, a presença de usuários *streamers* ainda é pequena. Ainda sobre a presença desses usuários nos enxames:

1. A presença de *streamers* pode ser maior quando o conteúdo popular compartilhado for inédito.
2. Não existe concentração de usuários sequenciais em um determinado horário do dia.
3. Muitos usuários utilizam versões antigas dos clientes.

Capítulo 5

Análise do impacto de streamers no desempenho de enxames BitTorrent

Embora a presença de usuários *streamers* em enxames populares seja muito pequena, conforme foi percebido nos experimentos do capítulo anterior, ainda não se pode afirmar o quanto esses usuários de fato interferem no desempenho geral das redes BitTorrent. E como já apresentado anteriormente, cresce cada vez mais o número de soluções de vídeo sob demanda baseados em *streaming* BitTorrent. Por isso não é difícil conceber a ideia que a presença desses usuários pode aumentar significativamente.

Também não se sabe se existe um número a partir do qual, e inclusive em termos percentuais, os usuários "view as you download" podem prejudicar a qualidade de serviço percebida nos enxames pelos demais usuários, ou tão pouco quanto o compartilhamento fica comprometido.

Bram Cohen defende em seu artigo sobre robustez das redes BitTorrent [8], a importância do algoritmo de seleção de trechos, porém não se aprofunda, e nem tão pouco apresenta dados de ensaios realizados, que nos garanta essa afirmação. Por isso, nesse capítulo serão apresentados experimentos feitos em laboratório considerando diversos fatores diferentes. O objetivo é entender se e como de fato esse usuários interferem nos enxames.

Para tentar compreender o provável impacto da presença de usuários "view as you download", foi realizado uma série de experimentos num laboratório com ambiente fechado, onde se podia controlar todos os *peers*.

Vários parâmetros poderiam ser levados em consideração e medidos durante esses experimentos, como por exemplo, tráfego de mensagens de *overhead*, variação do atraso

de *download* e tempo de *playback (Jitter)* para quem estiver em modo de reprodução de mídia, proporção de *download/upload*, *Mean Opinion Score* (MOS), etc. No entanto, para os usuários de redes BitTorrent, o tempo de *download* é a principal medida da qualidade de serviço. Por isso, os usuários sempre tendem a buscar enxames com mais *peers*, já que quanto mais usuários, maior é a disponibilidade dos trechos, bem como maior será a capacidade de *upload* da rede. Por isso, as seções de experimento descritas a seguir consideram o tempo de *download* como principal medida de desempenho do enxame.

O experimento descrito nesse capítulo foi executado em um dos laboratórios do Instituto Federal Fluminense. Cada sessão demorou um dia inteiro, devido às configurações exigidas no início de cada bateria e ao próprio tempo do experimento. Além disso, por se tratar de um espaço com uso compartilhado, não era possível realizar as baterias a qualquer tempo. Somadas as 15 sessões apresentadas a seguir, houveram outras tantas que foram descartadas por falha no processo. Por isso, o experimento levou mais de 8 meses para ser concluido.

5.1 Experimentos preliminares

Na primeira primeira sessão as seguintes configurações foram consideradas:

1. 1 publicador/*tracker*/*seeder*, com cliente qBitTorrent, rodando em máquina Linux Ubuntu 13.10, com velocidade ilimitada;
2. 1 *peer* espião, com cliente Transmission 2.8.2, rodando em máquina Linux Ubuntu 13.04;
3. 30 *peers leechers*, com cliente qBitTorrent, rodando em máquina Linux Ubuntu 13.10, com velocidade limitada a 100kB;
4. Todas as máquinas conectadas em rede local;
5. Vizinhança ilimitada;
6. O publicador fica on-line primeiro, seguido pelo monitor;
7. Os *peers leechers* entram na rede ao mesmo tempo;
8. Os usuários desconectam ao terminar seu *download*;

Usando essas configurações, foram realizadas 3 baterias iniciais, sendo a primeira sem nenhum usuário em modo sequencial, a segunda com 10% dos *peers* em modo "view as you download" (3 *peers*), e a última com 50% do *peers* como *streamers* (15 *peers*). Essa primeira sessão de experimento teve os resultados apresentados na Tabela 5.1.

Tabela 5.1: Medida dos tempos com vizinhança ilimitada e entrada simultânea, baterias com 0%, 10% e 50%

	0%	**10%**	**50%**
Tempo Total	16	15	22
Média	15,2258064516129	14,0645161290323	21,0967741935484
Moda	15	14	21
Variância	0,180645161290323	0,0623655913978494	0,0903225806451613

As baterias de 0% e 50% atenderam a expectativa de que quanto mais usuários em modo *streaming*, isso é, usando política de seleção sequencial de trechos, pior seria o desempenho geral da rede. No entanto, a bateria com 10% de usuários "view as you download" teve resultado, em relação ao tempo de *download*, ligeiramente melhor do que a bateria sem a presença de *peers* com comportamento sequencial.

Na tentativa de compreender o fenômeno apresentado, essa sessão foi repetida mais 2 vezes, e ambas tiveram resultados semelhantes. Optou-se então por modificar alguns parâmetros na repetição do experimento, e as seguintes sessões foram realizadas:

1. Sessão de 3 baterias (0%, 10% e 50%) com velocidade limitada a 50KB;
2. Sessão de 3 baterias (0%, 10% e 50%) com velocidade limitada a 100KB;
3. Sessão de 3 baterias (0%, 10% e 50%) com velocidade limitada a 200KB;
4. Sessão de 3 baterias (0%, 10% e 50%) com velocidade limitada a 50KB, usando Bitcomet;
5. Sessão de 3 baterias (0%, 10% e 50%) com velocidade limitada a 50KB, com intervalo entre as chegadas dos *peers* a cada 10 segundos;
6. Sessão de 3 baterias (0%, 10% e 50%) com velocidade limitada a 50KB, usando uma chegada de Poisson a cada 30s;

E supreendentemente todos os resultados foram similares à primeira sessão, a bateria com 10% de *streamers* apresentava resultados melhores ou muito parecidos com a bateria de 0%.

Então que para melhor compreender esse comportamento, esse experimento teve seu escopo ampliado, conforme apresentado a seguir.

5.2 Experimentos de escopo ampliado

Foram estabelecidas três diferentes configurações. A seguir é apresentada a interseção desses parâmetros:

1. 1 publicador/*tracker*/*seeder*, com cliente qBitTorrent, rodando em máquina Linux Ubuntu 13.10, com velocidade limitada a 50kB;
2. 1 *peer* espião, com cliente Transmission 2.8.2, rodando em máquina Linux Ubuntu 13.04;
3. 30 *peers leechers*, com cliente qBitTorrent, rodando em máquina Linux Ubuntu 13.10, com velocidade limitada a 300kB;
4. Todas as máquinas conectadas em rede local;
5. O publicador fica on-line primeiro, seguido pelo monitor;
6. Os usuários desconectam ao terminar seu *download*;
7. 6 baterias: 0%, 10%, 20%, 30%, 40% e 50% de usuários em modo sequencial;

Compartilhando esses parâmetros base, foram estabelecidas 3 configurações diferentes. Esses três formatos de experimentos foram definidos fundalmentalmente considerando a forma de entrada dos *peers* e o tamanho da vizinhança.

- **Vizinhança ilimitada e entrada simultânea**

  Os *peers leechers* entram na rede ao mesmo tempo. Não existe restrição quanto ao número de conexões simultâneas um *peer* pode estabelecer.

- **Vizinhança limitada e entrada simultânea**

  Os *peers leechers* entram na rede ao mesmo tempo. A vizinhança de cada *peer* foi limitada a 10 vizinhos.

- **Vizinhança ilimitada e entrada a cada 10 segundos**

  Os *peers leechers* entram na rede a cada 10 segundos. A vizinhança de cada *peer* foi limitada a 10 vizinhos.

5.3 Análise dos resultados

As três configurações tiveram 2 sessões, sendo cada sessão com 6 baterias (0%, 10%, 20%, 30%, 40% e 50%). Em todas as bateria foram registrados as seguintes informações:

1. Tempo de permanência na rede de cada *peer*;
2. Registro de mensagens de "*have*";
3. IP: IP do *peer* vizinho, para identificar e diferenciar cada *peer*;
4. PeerID: Identificador único criado pelo espião para identificar seus vizinhos por sessão. Garante sua identificação mesmo que o *peer* esteja atrás de um NAT;
5. Ordem: Representa a ordem de chegada da mensagem, é fundamental para o cálculo da entropia por permutação;
6. *Bitfield*: Vetor de *bits* que representa blocos recebidos pelo *peer* remetente;
7. Tempo total da bateria.

5.3.1 Vizinhança ilimitada e entrada simultânea

A Figura 5.1 ilustra os resultados de uma sessão com vizinhança ilimitada e entrada simultânea. Ela apresenta, de forma direta, o comportamento do tempo de *download* durante as seis baterias. Na imagem é perceptível a redução da média do tempo a 10%, conforme já detectado nos primeiros testes, embora exista um aumento no tempo total.

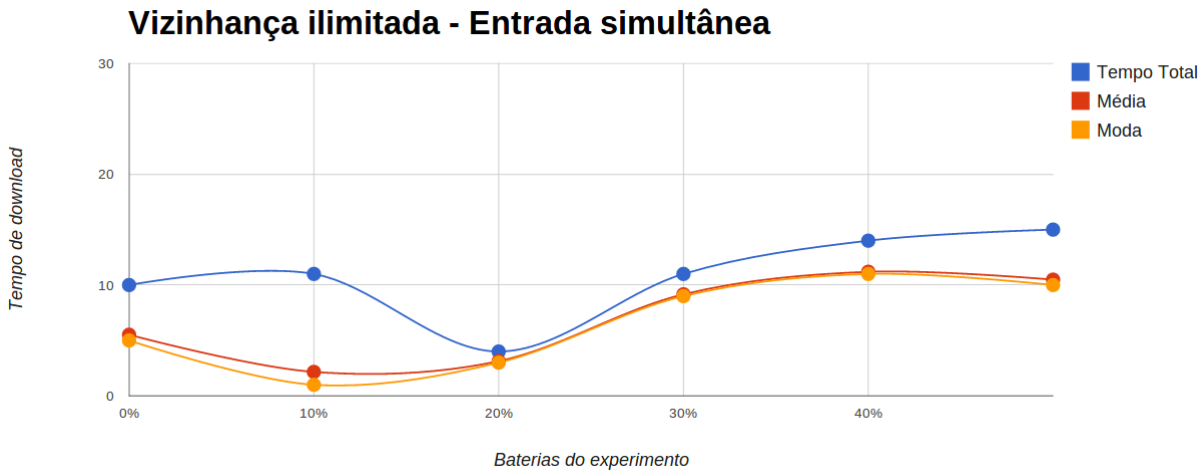

Figura 5.1: Desempenho do enxame com vizinhança ilimitada e entrada simultânea

A bateria de 20% tem um melhor desempenho geral, pois embora sua média e moda sejam um pouco maiores que a da bateria anterior, o tempo total do experimento é menor que a metade de todos os outros. E por fim, nas baterias a com 30% ou mais, existe um clara piora nas métricas de tempo.

5.3.2 Vizinhança limitada e entrada simultânea

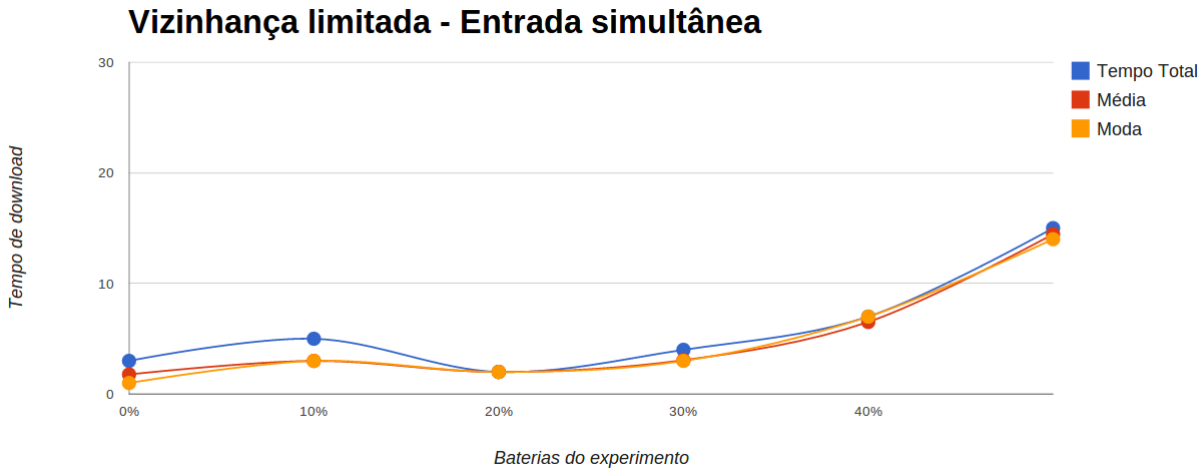

Figura 5.2: Desempenho do enxame testado em laboratório com vizinhança limitada e entrada simultânea

A Figura 5.2 apresenta as informações de uma sessão com vizinhança limitada e entrada simultânea. Embora as curvas sejam mais suaves, fica claro que o comportamento observado na configuração anterior e apresentado na Figura 5.1 se repete nessa configuração. É perceptível um aumento do tempo total na bateria a 10%, e embora não exista uma redução da média do tempo a 10%, ela também não aumenta muito, comparativamente.

Também na Figura 5.2 é possível perceber que a bateria de 20% também tem um melhor desempenho geral. Apesar de sua média e moda sejam um pouco maiores que as da bateria anterior, o tempo total do experimento é menor que a metade de todos os outros. E por fim, nas baterias a com 30% ou mais, existe um clara piora nas métricas de tempo, embora não muito direta em 30%.

5.3.3 Vizinhança limitada e entrada a cada 10 segundos

No exemplo de sessão com vizinhança limitada e entrada dos *peers* a cada 10 segundos, apresentada pela Figura 5.3 é apresentada uma curva um pouco diferente das anteriores. Nessa configuração, a bateria de 10% tem um resultado pior que que as de 0%, 20%. 30% e 40%, e a bateria a 50% continua apresentando o pior desempenho geral.

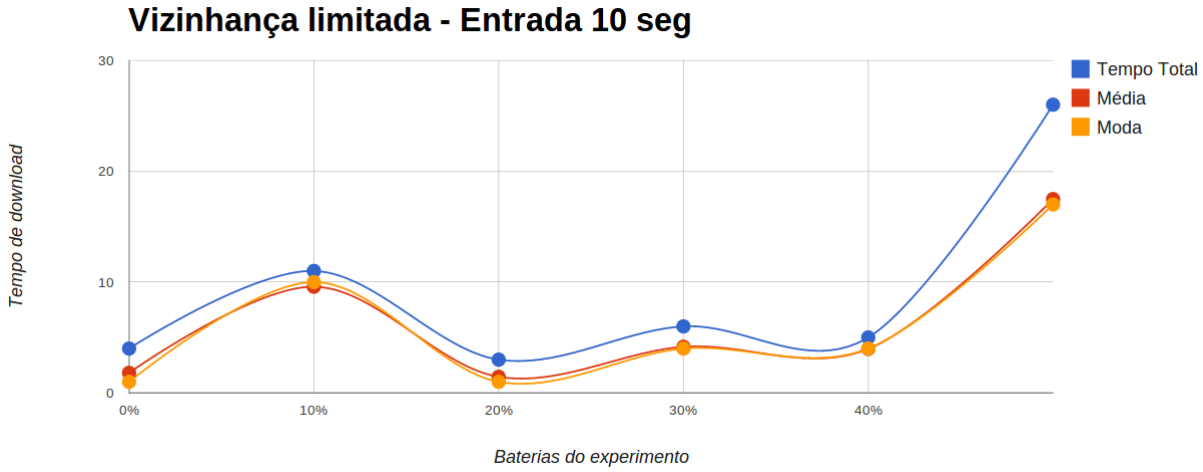

Figura 5.3: Desempenho do enxame testado em laboratório com vizinhança ilimitada e entrada a cada 10 segundos

5.4 Conclusão Parcial

Considerando as diferentes configurações de experimento, é possível afirmar que existe sim uma piora no tempo de *download* no enxame, seja ele individual ou total, a medida que se adiciona usuários operando em modo sequencial nessas redes.

Em casos extremos, o desempenho entre uma bateria sem a presença de *streamers* para uma com metade dos *peer*s em modo sequencial chegou a ser 17 vezes maior, e o tempo total da bateria chegou a ser 6 vezes pior com 50% dos *peers* em modo *streaming*.

No entanto, esse estudo detectou que, para alguns parâmetros específicos, a presença desses usuários pode ser até bem vinda, desde que não ultrapasse um limite definido pela configuração do enxame em questão.

Essa informação pode ser usada para implementar mecanismos de controle e acesso às redes BitTorrent, para caso a presença desse tipo de usuário aumente, a rede possa responder de forma apropriada e assim evitar uma piora geral de desempenho. Piora essa que pode inclusive ser percebida na reprodução das mídias dos usuários "view as you

download".

Além disso, como boa parte do conteúdo disponível nessas redes está presente sem a permissão de seus proprietários, as indústrias responsáveis pela produção desse tipo de material poderia se utilizar dessa informação sobre a queda de desempenho para prejudicar as redes BitTorrent, o que torna a implementação de tais mecanismos ainda mais importantes para longevidade do BitTorrent.

Foram também calculados os valores de entropia por permutação nesse experimento. A percepção da mudança dos valores é baixa por se tratar de um ambiente com muito pouca disponibilidade (apenas um *seeder*). No entanto ela existe e a influência dos *streamers* é perceptível nos números, pois existe uma ligeira redução geral nos valores da entropia a medida que a porcentagem de usuários sequenciais aumenta.

Capítulo 6

Contribuições e trabalhos futuros

O uso de redes BitTorrent como parte de soluções de mídia sob demanda, tem se apresentado cada vez mais frequente. Só nos últimos meses antes da apresentação desse trabalho, mais de 5 novas ferramentas foram apresentadas na Internet, além da quantidade gigante de reportagens sobre o assunto. Esse movimento foi motivado pela grande capacidade dos enxames de lidar com um colossal número de usuários, embora seja o seu gigantesco acervo o grande e verdadeiro responsável pelo uso de BT como solução *streaming* de vídeo sob demanda.

Foram apresentados nos capítulos anteriores, nos quais foi descrito o desenvolvimento desse trabalho, as ferramentas desenvolvidas, bem como os experimentos realizados para executar essa pesquisa. Nesse último capítulo, será apresentado de uma forma direta as contribuições e possíveis trabalhos futuros desta dissertação.

6.1 Contribuições

1. Apresentou-se nesse estudo os detalhes dos *softwares* clientes BitTorrent mais populares que implementaram soluções *streaming* de vídeo sob demanda;

2. Foi desenvolvido um cliente BitTorrent espião, que implementa todas as tecnologias atuais necessárias para se conectar a redes reais e monitorar a troca de informação entre os outros usuários da rede;

3. Usando como base matemática os conceitos de entropia e entropia por permutação, foi possível estabelecer, através da troca de mensagens dos usuários conectados à rede BitTorrent, um *threshold* que caracterize esses usuários em regulares (com requisição aleatória) ou "view as you download" (requisição sequencial), baseado

nos valores gerais de entropia do programa cliente BitTorrent usado pelo usuário analisado;

4. Através do monitoramento, foi possível acompanhar a evolução dos trechos recebidos por cada usuário, num experimento em larga escala, que acompanhou por 3 meses os enxames mais populares da Internet;

5. Foi percebida uma presença ainda muito pequena de usuários *streamers* nas redes populares, verificando apenas 0,53% desses usuários nesses enxames, embora essa porcentagem chegue a quase 2% no caso de enxames de séries recém publicadas, o que nos ajuda a compreender o perfil das pessoas interessadas nessa funcionalidade;

6. Foi verificado que existe um ponto aceitável de usuários sequenciais dentro dos enxames sem que a qualidade, medida através de parâmetros específicos, fosse alterada;

6.2 Trabalhos futuros

Embora essa pesquisa tenha conseguido responder suas questões fundamentais, muitas outras possibilidades se apresentam como objetos de estudos futuros, onde se destacam:

1. Desenvolver um mecanismo, baseado no modelo de construção de vizinhança do protocolo BitTorrent, que permita a qualquer *peer*, detectar com poucas mensagens, a presença de usuários em modo sequencial, e com isso avaliar a saúde do enxame conectado.

2. Repetir os experimentos de enxames privados em ambiente real da internet, que leve em consideração latência, distância geográfica, saltos entre enlaces, e outras características que podem influenciar no desempenho;

3. Nesse estudo foi usado prioritariamente vídeo, no entanto, é válido experimenta outras mídias, e /ou considerar a qualidade e tamanho desses arquivos;

4. Calcular a entropia de clientes específicos baseado no número de mensagens;

5. Verificar se existem outros tipos de enxame, além de verificados por esse estudo, que possuam destaque na presença de usuários sequenciais;

Referências

[1] Adweek.com. Bewkes: Game of thrones piracy 'better than an emmy'. http://www.adweek.com/news/television/bewkes-game-thrones-piracy-better-emmy-151738/, (acessado em agosto de 2014).

[2] Azureus Software, I. Vuze: o melhor aplicativo de bittorrent do mundo. http://www.vuze.com/, (acessado em agosto de 2014).

[3] Bandt, C.; Pompe, B. Permutation entropy: a natural complexity measure for time series. *Physical Review Letters 88*, 17 (2002), 174102.

[4] BitTorrent, I. Bittorrent. http://www.bittorrent.com/, (acessado em agosto de 2014).

[5] BitTorrent, I. Bittorrent and μtorrent software surpass 150 million user milestone; announce new consumer electronics partnerships. http://www.bittorrent.com/intl/es/company/about/ces_2012_150m_users, (acessado em agosto de 2014).

[6] BitTorrent, I. μtorrent. http://www.utorrent.com/, (acessado em agosto de 2014).

[7] Cisco. Cisco visual networking index: Forecast and methodology, 2013–2018. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html, (acessado em agosto de 2014).

[8] Cohen, B. Incentives build robustness in bittorrent. Em *Workshop on Economics of Peer-to-Peer systems* (2003), vol. 6, páginas 68–72.

[9] Cohen, B. The bittorrent protocol specification. http://www.bittorrent.org/beps/bep_0003.html, (acessado em agosto de 2014).

[10] Couchpota.to. Couchpotato. https://couchpota.to/, (acessado em agosto de 2014).

[11] Feng, C.; Li, B. On large-scale peer-to-peer streaming systems with network coding. Em *Proceedings of the 16th ACM international conference on multimedia* (2008), ACM, páginas 269–278.

[12] Forbes. 'game of thrones' sets piracy world record, but does hbo care? http://www.forbes.com/sites/insertcoin/2014/04/15/game-of-thrones-sets-piracy-world-record-but-does-hbo-care/, (acessado em agosto de 2014).

[13] Gomes, M. Como usar o raspberry pi para ver filmes e séries em pt-br na sua tv, com download automático de lançamentos em full hd! http://marcogomes.com/blog/2014/como-usar-o-raspberry-pi-para-ver-filmes-e-series-em-pt-br-na-sua-tv-com-download-automatico-de-lancamentos-em-full-hd/, (acessado em agosto de 2014).

[14] Group, B. D. Bitcomet. http://www.bitcomet.com, (acessado em agosto de 2014).

[15] Group, B. D. Preview of files while downloading. http://wiki.bitcomet.com/preview_while_downloading, (acessado em agosto de 2014).

[16] Harrison, D. The bittorrent enhancement proposal process. http://www.bittorrent.org/beps/bep_0001.html, (acessado em agosto de 2014).

[17] hbdia.com. Popcorn time, um app pra assistir filmes por streaming (e o que ele poderia significar pra hollywood). http://hbdia.com/sem-categoria/popcorn-time-um-app-pra-assistir-filmes-por-streaming-e-o-que-ele-poderia-significar-pra-hollywood/, (acessado em agosto de 2014).

[18] Hei, X.; Liang, C.; Liang, J.; Liu, Y.; Ross, K. W. A measurement study of a large-scale p2p iptv system. *Multimedia, IEEE Transactions on 9*, 8 (2007), 1672–1687.

[19] Ktorrent, C. Ktorrent. http://ktorrent.pwsp.net/, (acessado em agosto de 2014).

[20] Kurose, J. F.; Ross, K. W. *Computer Networking: A top-down approach featuring the Internet*, vol. 2. Addison-Wesley Reading, 2001.

[21] Li, Z.; Zhang, T. Understanding the quality-of-service of ubiquitous bittorrent-like media streaming. *Greenorbs* (2013).

[22] Mendonça, G. G.; Leao, R. M. Btstream–um ambiente para desenvolvimento e teste de aplicaç oes de streaming p2p. *XXX SBRC, Salao de Ferramentas, Ouro Preto, MG, Brazil* (2012).

[23] Netflix.com. Netflix. www.netflix.com, (acessado em agosto de 2014).

[24] of Life, H. Bitlord. http://www.bitlord.com/, (acessado em agosto de 2014).

[25] of Life, H. Bitlord 2.3.2 with video audio streaming. http://www.bitlord.com/?p=187, (acessado em agosto de 2014).

[26] Padmanabhan, V. N.; Wang, H. J.; Chou, P. A.; Sripanidkulchai, K. Distributing streaming media content using cooperative networking. Em *Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video* (2002), ACM, páginas 177–186.

[27] Parvez, N.; Williamson, C.; Mahanti, A.; Carlsson, N. Analysis of bittorrent-like protocols for on-demand stored media streaming. *ACM SIGMETRICS Performance Evaluation Review 36*, 1 (2008), 301–312.

[28] POPCORNTIME.IO. Popcorn time - watch movies and tv shows instantly! http://popcorntime.io/, (acessado em agosto de 2014).

[29] PPLWARE.SAPO. 5 fantásticos clientes para torrents. http://pplware.sapo.pt/linux/5-fantasticos-clientes-para-torrents/, (acessado em agosto de 2014).

[30] PPTV.COM. Pptv. http://www.pptv.com/, (acessado em agosto de 2014).

[31] QBITTORREN, C. qbittorrent official website. http://www.qbittorrent.org/, (acessado em agosto de 2014).

[32] RASPBERRYPI.ORG. Raspberry pi. http://www.raspberrypi.org/, (acessado em agosto de 2014).

[33] RIEDL, M.; MÜLLER, A.; WESSEL, N. Practical considerations of permutation entropy. *The European Physical Journal Special Topics 222*, 2 (2013), 249–262.

[34] ROCHA, A. A. A. *Sobre medidas de desempenho da Internet para o uso em aplicações de rede*. Tese de Doutorado, COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, Abril 2010.

[35] SCHULZE, H., M. K. Internet study 2008/2009. http://ipoque.com/en/resources/internet-studies, (acessado em agosto de 2014).

[36] SHAH, P.; PARIS, J.-F. Peer-to-peer multimedia streaming using bittorrent. Em *Performance, Computing, and Communications Conference, 2007. IPCCC 2007. IEEE Internationa* (2007), IEEE, páginas 340–347.

[37] SHANNON, C. E. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review 5*, 1 (2001), 3–55.

[38] SICKBEARD.COM. Sick beard - internet pvr for your tv shows. http://sickbeard.com/, (acessado em agosto de 2014).

[39] SOPCAST.ORG. Soapcast. http://www.sopcast.org/, (acessado em agosto de 2014).

[40] STREAMERP2P.COM. Streamerp2p.com. http://streamerp2p.com/, (acessado em agosto de 2014).

[41] TEAM, D. Deluge. http://http://deluge-torrent.org/, (acessado em agosto de 2014).

[42] TECHCRUNCH.COM. Popcorn time is back. http://techcrunch.com/2014/03/16/popcorn-time-is-back/, (acessado em agosto de 2014).

[43] TECHCRUNCH.COM. Popcorn time is dead. http://techcrunch.com/2014/03/14/popcorn-time-is-dead/, (acessado em agosto de 2014).

[44] TECHNOLOGIES, T. N. Xunlei. www.xunlei.com/, (acessado em agosto de 2014).

[45] TOPFREEWARES. Top 5 programas (clientes) de torrent gratuitos. http://www.topfreewares.com.br/top-5-programas-clientes-de-torrent-gratuitos/, (acessado em agosto de 2014).

[46] TORRENTFREAK.COM. Bittorrent users are most active on sundays. https://torrentfreak.com/bittorrent-users-active-sunday-140208/, (acessado em agosto de 2014).

[47] TORRENTFREAK.COM. Thunder blasts μtorrent's market share away. http://torrentfreak.com/thunder-blasts-utorrents-market-share-away-091204/, (acessado em agosto de 2014).

[48] TORRENTFREAK.COM. Top 10 μtorrent alternatives. http://torrentfreak.com/top-10-utorrent-alternatives-120819/, (acessado em agosto de 2014).

[49] TRANSMISSIONBT.COM. Transmission. https://www.transmissionbt.com/, (acessado em agosto de 2014).

[50] UTORRENT.COM. Faq μtorrent 3.0. http://www.utorrent.com/intl/pt/help/faq/ut3, (acessado em agosto de 2014).

[51] VARIETY.COM. Popcorn time reality check: Usage of the piracy app is actually tiny. http://variety.com/2014/digital/news/popcorn-time-reality-check-usage-of-the-piracy-app-is-actually-tiny-1201136445/, (acessado em agosto de 2014).

[52] VLAVIANOS, A.; ILIOFOTOU, M.; FALOUTSOS, M. Bitos: Enhancing bittorrent for supporting streaming applications. Em *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings* (2006), IEEE, páginas 1–6.

[53] VUZE.COM. Vuze play. http://www.vuze.com/features/playnow, (acessado em agosto de 2014).

[54] WIKI, T. Bittorrentfaq. https://wiki.theory.org/BitTorrentFAQ, (acessado em agosto de 2014).

[55] WIKIPEDIA. Comparison of bittorrent clients. http://en.wikipedia.org/wiki/Comparison_of_BitTorrent_clients, (acessado em agosto de 2014).

[56] WIKIPEDIA. Napster. http://pt.wikipedia.org/wiki/Napster, (acessado em agosto de 2014).

[57] WIKIPÉDIA. Herfindahl index. http://en.wikipedia.org/wiki/Herfindahl_index, (acessado em agosto de 2014).

[58] WOLFFENBÜTTEL, A. Índice de gini. http://desafios.ipea.gov.br/index.php?option=com_conte (acessado em agosto de 2014).

[59] YIFY TORRENT.COM. Home - yts. http://yify-torrents.com, (acessado em agosto de 2014).

APÊNDICE A – Espião Beholder

O monitor é a customização da versão 2.8.2 do cliente Transmission. Existe muita diferença entre as versões, a 1.7.6 também foi estudada, mas a versão 2.8.2 se provou mais fácil para modificar, além de ser mais nova. Existe pouca documentação (ver `http://www.transmissionbt.com/`) sobre a arquitetura do programa, apenas algum material sobre RPC (Remote Procedure Call).

Existe no Transmission, uma janela de log, que pode ser usada como interface para os logs gerados para o experimento, pois pode salvar os dados, e permite usar uma categoria de log específica (level).

Examinando o código do transmission 2.8.2, seguindo as chamadas da estrutura bitfield.h na biblioteca libtransmission, chegamos a duas chamadas em código .c, que fazem referência a estrutura. No bitfield.h foi criado um identificador para filtrar bitfields dos vizinhos.

```
typedef struct tr_bitfield
{
  uint8_t *  bits;
  size_t     alloc_count;
  size_t     bit_count;
  size_t     true_count;
  bool       have_all_hint;
  bool       have_none_hint;
  int        identificadorBeholder;
}
```

Snippet - bitfield.h

No construtor do bitfield no bitfield.c foi é iniciada com um valor único.

```
void
tr_bitfieldConstruct (tr_bitfield * b, size_t bit_count)
{
  b->bit_count = bit_count;
  b->true_count = 0;
  b->bits = NULL;
  b->alloc_count = 0;
  b->have_all_hint = false;
  b->have_none_hint = false;
  b->identificadorBeholder = __TIME__;
  assert (tr_bitfieldIsValid (b));
}
```

Snippet - bitfield.c

No controle de mensagem no arquivo peer-msgs.c, foi adicionada a função:

```
static int beholderOrder = 0;

static void beholdHavePiece(uint32_t ui32, int * hora,time_t peer_id_creation_time,char * clientName, tr_ad
{
    printf("PeerID: %i ",hora);
    printf(" IP %u", endereco->addr);
    printf(" Have %u", ui32);
    printf(" OrdemGeral: %i",beholderOrder);
    printf(" Torrent %u", peer_id_creation_time);
    printf(" cliente %s", clientName);
    printf("\n");

    char filename[255] = {0};
    sprintf(filename, "/etc/beholder/saida_%u.txt", peer_id_creation_time);
    FILE *fp;
    fp = fopen(filename,"a+");
    fprintf(fp,"PeerID: %i ",hora);
    fprintf(fp," IP %u", endereco->addr);
    fprintf(fp," Have %u", ui32);
    fprintf(fp," OrdemGeral: %i",beholderOrder);
    fprintf(fp," Torrent %u", peer_id_creation_time);
    fprintf(fp," cliente %s", clientName);
    fprintf(fp,"\n");
    fclose(fp);

    beholderOrder++;
}
```

Snippet - peer-msgs.c

E a função "readBtMessage"foi modificada, como mostra o próximo snippet code:

```
case BT_HAVE:
    tr_peerIoReadUint32 (msgs->io, inbuf, &ui32);
    dbgmsg (msgs, "got Have: %u", ui32);
    if (tr_torrentHasMetadata (msgs->torrent)
            && (ui32 >= msgs->torrent->info.pieceCount))
    {
        fireError (msgs, ERANGE);
        return READ_ERR;
    }
    if (!tr_bitfieldHas (&msgs->peer.have, ui32)) {
        tr_bitfieldAdd (&msgs->peer.have, ui32);
        fireClientGotHave (msgs, ui32);
    }
    updatePeerProgress (msgs);
    const struct tr_address *endereco = tr_peerIoGetAddress (msgs->io, NULL);
    //Beholder
    beholdHavePiece(ui32,&msgs->peer.have.identificadorBeholder,
        &msgs->torrent->peer_id_creation_time,&msgs->peer.clientName, endereco);
    //Beholder
    break;

case BT_BITFIELD: {
    uint8_t * tmp = tr_new (uint8_t, msglen);
    dbgmsg (msgs, "got a bitfield");
    tr_peerIoReadBytes (msgs->io, inbuf, tmp, msglen);
    tr_bitfieldSetRaw (&msgs->peer.have, tmp, msglen, tr_torrentHasMetadata (msgs->torrent));
    fireClientGotBitfield (msgs, &msgs->peer.have);
    updatePeerProgress (msgs);
    tr_free (tmp);
    //Beholder
    beholdBitfiled(&msgs->peer.have,tmp,&msgs->peer.have.identificadorBeholder);
    //Beholder
    break;
}
```

Snippet - peer-msgs.c

A.1 Compilando código

Os passos abaixo são descritos para ubuntu 13.10. A versão mais atualizada, disponível em `download-origin.transmissionbt.com/files/`, que inclui suporte a DHX, PEX, link magnético e UDP, disponível no site era 2.82, diferente do tutorial `trac.transmissionbt.com/wiki/Building` que considera 1.76. Do conjunto de dependências descritos no site:

sudo apt-get install build-essential automake autoconf libtool pkg-config intltool libcurl4-openssl-dev libglib2.0-dev libevent-dev libminiupnpc-dev libminiupnpc5 libappindicator-dev

A biblioteca libminiupnpc5 não foi instalada (diz que não está disponível). Porém, para instalar a nova versão (com a libevent):

sudo apt-get remove transmission transmission-daemon

transmission-common transmission-gtk transmission-cli transmission-qt

sudo apt-get install build-essential automake autoconf checkinstall libtool

pkg-config libcurl4-openssl-dev intltool libxml2-dev libgtk-3-dev

libnotify-dev libglib2.0-dev libgconf2-dev libcanberra-gtk-dev libappindicator3-dev

mkdir -v $HOME/transmission_build cd $HOME/transmission_build &&

wget `https://github.com/downloads/libevent/libevent/libevent-2.0.21-stable.tar.gz` &&

tar xvf libevent-2.0.21-stable.tar.gz &&

cd libevent-2.0.21-stable &&

./configure –prefix=$HOME/transmission$_b$*uild/libevent*&&

make && make install

A.2 Roteiro de uso

1. Instale o ubuntu 13.10 (nele o transmission já é 2.82 (14160))
2. Atualize o sistema
3. Abra o terminal
4. Inicie a sessão como root
5. Dê permissão de escrita ao usuário que executará o transmission na pasta /etc/beholder com o comando *chmod*
6. Instale as dependências descritas anteriormente
7. Navegue até a pasta do transmission-2.82 customizado
8. execute: ./configure -q make -s
9. execute: make install
10. execute: transmission-gtk
11. Na interface clique em "abrir"e localize o arquivo .torrent
12. Espere o cliente se conectar a rede
13. No terminal o cliente mostrará as mensagens de “have” dos vizinhos, e/ou como seu bitfield (dependendo da versão customizada)