Anatomia do BitTorrent

a Ciência da Computação por trás do protocolo

Paulo Cheadi Haddad Filho
Orientador: José Coelho de Pina

Trabalho de Formatura Supervisionado

IME-USP

Universidade de São Paulo
São Paulo, 2013

Sumário

Lista de Códigos-fonte

Lista de Figuras

Lista de tarefas pendentes

Capítulo 1

Introdução

Aqui vou explicar o objetivo do trabalho e o que será mostrado ao longo dele.

Capítulo 2

Napster, Gnutella, eDonkey e BitTorrent

Para entendermos como e por que o BitTorrent se tornou o que é hoje, devemos voltar um pouco no tempo e rever a história que precedeu à sua criação, que é no fim da década dos anos 1990.

2.1 Período pré-torrent

Entre o final dos anos 80 e o início dos 90 [47, 62], a Internet deixou de ser uma rede de computadores usada somente por entidades governamentais, laboratórios de pesquisa e universidades, passando a ter seu acesso comercializado para o público em geral pelos fornecedores de acesso a Internet (ISPs) [52]. Com o advento do formato de áudio MP3 (MP3) [57], no final de 1991, e do seu primeiro reprodutor de áudio MP3 Winamp, o tráfego da Internet cresceu devido ao aumento da troca direta desse tipo de arquivo.

Entre 1998 e 1999, dois sites de compartilhamento gratuito de músicas foram criados: o MP3.com [56], que era um site de divulgação de bandas independentes, e o Audiogalaxy.com [37, 43]. Mais popular que o primeiro, o Audiogalaxy era um site de busca de músicas, sendo que o download e upload eram feitos a partir de um software cliente. A lista de músicas procuradas era enviada pelo site para o computador onde o usuário tinha instalado o cliente, que então conectava com o computador do outro usuário, que era indicado pelo servidor. A lista possuía todos os arquivos que um dia passaram pela sua rede. Se algum arquivo fosse requisitado, mas o usuário que o possuísse não estivesse conectado, o servidor central do Audiogalaxy fazia a ponte, pegando o arquivo para si e enviando-o para o cliente do requisitante em seu próximo login.

Os 3 anos seguintes à criação desses dois sites foram muito produtivo ao mundo das redes

peer-to-peer (P2P) de modo geral, onde surgiram alguns protocolos desse paradigma e inúmeros softwares que os implementavam. Os mais relevantes foram o Napster, o Gnutella, o eDonkey e o BitTorrent.

Napster

Em maio de 1999, surgiu o Napster [58], um programa de compartilhamento de MP3 que inovou por desfigurar o usual modelo cliente-servidor, no qual um servidor central localizava os arquivos nos usuários e fazia a conexão entre estes, onde ocorriam as transferências. O Napster foi contemporâneo ao Audiogalaxy, e ambos fizeram muito sucesso por cerca de 2 anos, até que começaram as ações judiciais contra ambos os programas.

Não demorou muito tempo para a indústria da música entrar em ação contra a troca de arquivos protegidos por direitos autorais sem autorização dos detentores de tais direitos pela Internet. Seu primeiro alvo foi o Napster, em dezembro de 1999, quando a RIAA (do inglês *Recording Industry Association of America*) entrou com processo judicial representando várias gravadoras, alegando quebra de direitos autorais [22]. Em abril de 2000, foi a vez da banda Metallica processar, como retaliação à descoberta de que uma música ainda não lançada oficialmente já circulava na rede [18, 21]. Um mês depois, outra ação judicial, agora encabeçada pelo rapper Dr. Dre, que tinha feito um pedido formal para a retirada de seu material de circulação [11]. Isso fez com que o Napster recebesse atenção da mídia, ganhando popularidade e atingindo os 20 milhões de usuários em meados do ano 2000 [30].

Em 2001, esses imbróglios judiciais resultaram numa liminar federal que ordenou que o Napster retirasse o conteúdo protegido das entidades representadas pela RIAA. O Napster tentou cumprir a ordem judicial, mas a juíza do caso não ficou satisfeita, ordenando então, em julho daquele ano, o desligamento da rede enquanto não conseguisse controlar o conteúdo que trafegava ali [58]. Em setembro, o Napster fez um acordo [41], onde pagou 26 milhões de dólares pelos danos já causados, pelo uso indevido de músicas, e mais 10 milhões de dólares pelos danos futuros envolvendo royalties. Para pagar esse valor, o Napster tentou cobrar o serviço que prestava de seus usuários, que acabaram migrando de rede P2P, inclusive para o Audiogalaxy. Não conseguindo quitar o acordo, em 2002, o Napster decreta falência e é forçado a liquidar seus ativos. De lá para cá, foi negociado algumas vezes, e, atualmente, pertence ao site Rhapsody [34].

O sucesso do Napster, mesmo que por curto período tempo, mostrou o potencial que as redes P2P poderiam ter, e com isso, novos softwares e protocolos de redes foram sendo lançados, sempre tentando se diferenciar dos seus antecessores, a fim de não serem novos alvos de ações judiciais. A solução para isso foi tentar descentralizar os mecanismos de indexação e de busca, que foram os pontos fracos do Napster.

Gnutella

O sucessor foi o Gnutella, que em março de 2000 [49], surgiu como uma resposta de domínio público, feita com ‘‘gambiarras’’, para os problemas que o Napster encontrou com relação às acusações de violação de direitos autorais. Enquanto o Napster possuía em sua estrutura um servidor central, fato este que foi explorado em seu julgamento como prova de que o sistema encorajava a violação de direitos autorais, o Gnutella foi modelado como um sistema P2P puro, onde todos os nós da rede (*peers*) são completamente iguais, sendo responsáveis pelos seus próprios atos.

O Gnutella disponibiliza arquivos da mesma forma que o Napster [4] , mas sem a limitação de ser de em formato de música, ou seja, qualquer arquivo pode ser compartilhado. A diferença mais significativa entre os dois protocolos é o algoritmo de busca: a abordagem do Gnutella é baseada numa forma de anycast . Isso envolve duas partes: a primeira, sobre como cada usuário é conectado a outros nós e mantém a lista dessas conexões atualizada; a segunda, sobre como ele trata as buscas e trabalha inundando de pedidos para todos os nós que estão a uma certa distância do usuário (nó-cliente). Por exemplo, se a distância limite for de 4, então todos os nós que estiverem a 4 passos a partir do cliente serão verificados, começando a partir dos mais próximos. Eventualmente, algum nó possuirá o arquivo requisitado e responderá, e assim será feita a transferência desse arquivo. Muitos softwares que implementam o protocolo vão além dessa funcionalidade básica de download simples, tentando transferir de forma paralela partes diferentes do arquivo desejado de nós diferentes, de forma a amenizar eventuais problemas de velocidade de rede.

Assim, experiências sugerem que o sistema escala para um tamanho maior, tornando o mecanismo de anycast extremamente caro, e, em alguns casos, até proibitivo. O problema ocorre nas buscas por arquivos menos populares, onde será necessário um maior número de nós perguntados.

O Gnutella ainda teve uma segunda versão [48], no final de 2002, onde utilizou o mesmo protocolo que o original, porém, organizando a rede de peers em folhas (*leafs*) e *hubs*. Um *hub* poderia ter centenas de conexões à outras folhas, mas apenas 7 (em média) a outros *hubs*, enquanto uma folha se conectaria a apenas 2 *hubs* simultaneamente. Essa nova topologia, somada com uma nova tabela de índice de arquivos das folhas mantida pelos *hubs* onde estavam conectados, melhorou o desempenho das buscas, que era ruim na versão antiga.

eDonkey

O protocolo eDonkey inovou em muitos aspectos em relação aos seus precursores, tendo papel fundamental na história das redes P2P, consolidando-se como ferramenta de compartilhamento especializado em arquivos grandes.

O eDonkey implementou o primeiro método de download por enxame de peers (*swarming*), onde peers fazem downloads de diferentes partes de um arquivo e de peers diferentes, utilizando de forma efetiva a largura de banda de rede para todos os peers, ao invés de ficar limitado somente à banda de um único peer.

Outra melhoria deu-se na busca de arquivos: no seu lançamento, os servidores eram separados entre si, porém, nas versões seguintes, permitiu-se que eles formassem uma rede de buscas. Isso possibilitou que os servidores repassassem buscas de seus clientes conectados localmente a outros servidores, facilitando a localização de peers conectados em qualquer servidor da rede de buscas, aumentando a capacidade de download do enxame.

Diferentemente do Napster, o eDonkey utilizou-se de valores hash de arquivos nos resultados de busca ao invés dos simples nomes dos arquivos. As buscas geradas pelos usuários eram baseadas em palavras-chave e comparadas com a lista de nomes de arquivos armazenada no servidor, mas o servidor retornava uma lista de pares de nomes de arquivos com seus respectivos valores hash. Enfim, quando o usuário selecionasse o arquivo desejado, o cliente iniciaria o download do arquivo usando o seu valor hash. Desse modo, um arquivo poderia ter muitos nomes entre os diferentes peers e servidores, mas seria considerado idêntico para download se possuísse o mesmo valor hash.

A arquitetura da rede em dois níveis, usando cliente e servidor, alcançou um meio termo entre as redes centralizadas (como o Napster) e as descentralizadas (como o Gnutella), já que o servidor central no primeiro era um alvo garantido para ações judiciais, enquanto o segundo mostrou-se inviável à propositura de tais ações, devido ao tráfego massivo de buscas entre peers.

Por fim, a inovação mais importante foi o uso de tabelas hash distribuídas (DHTs), em específico o Kademlia, como algoritmo de indexação e busca nos servidores centrais dos arquivos através da rede eDonkey. Além de ser uma das razões da melhoria no desempenho das pesquisas, os DHTs possuem ainda outras características, tais como tolerância a falhas e escalabilidade.

2.2 Nascimento do BitTorrent

Em meados dos anos 1990, Bram Cohen era um programador que tinha largado a faculdade no segundo ano do curso de Ciência da Computação, da Universidade de Buffalo -- Nova Iorque, para trabalhar em empresas ‘‘ponto com’’. A última delas foi a MojoNation, uma empresa que desenvolvia um software de distribuição de arquivos criptografados por swarming.

Em abril de 2001, Bram saiu da MojoNation e começou a modelar o protocolo BitTorrent, lançando a primeira implementação usando a linguagem Python, em julho de 2001. Em fevereiro de 2002, ele apresentou o seu trabalho na CodeCon [8], e na mesma época começou a testá-lo,

usando como chamariz uma coleção de material pornográfico para atrair beta testers [36]. Assim, o software começou a ser usado imediatamente.

Nesse meio tempo, Bram ainda passou pela Valve [45], empresa de desenvolvimento de jogos, trabalhando no sistema de distribuição online do jogo Half Life 2. Em 2004, saiu da Valve e voltou o foco ao Torrent. Em setembro, fundou a BitTorrent Inc. com seu irmão Ross Cohen e o parceiro de negócios Ashwin Navin, sendo então responsável pelo desenvolvimento do protocolo. Ainda naquele ano, surgiram os primeiros programas de televisão e filmes compartilhados na rede através do BitTorrent, o que popularizou o protocolo.

Em maio de 2005, a empresa lançou uma nova versão do BitTorrent, que não precisava de rastreadores (*trackers*), juntamente com um site de buscas de conteúdo torrent na Internet. Em setembro, a empresa recebeu investimento na ordem de $8.5 milhões de dólares. No final desse ano, a BitTorrent Inc. e a MPAA (*Motion Picture Association of America*, associação americana de produtoras de filmes) fizeram um acordo [55] visando a retirada dos conteúdos não autorizados dos representados pela associação, o que não evitou a pirataria, pois já havia outros sites de busca de torrent sem restrições de conteúdo, como o TorrentSpy, Mininova, The Pirate Bay, etc.

2.3 Mundo pós-torrent

Desde o fechamento de seu site de buscas, a BitTorrent Inc. tem desenvolvido outros softwares baseados na tecnologia P2P [5], como transmissão de vídeos ao vivo (BitTorrent Live), sincronização de arquivos entre computadores ligados à Internet (BitTorrent Sync), publicação e distribuição de conteúdo de artistas a seus fãs (BitTorrent Bundles), entre outros serviços comerciais.

Como protocolo, o BitTorrent criou um novo paradigma de transmissão de informações pela Internet, sendo utilizado de inúmeras formas e motivos, tais como:

- alguns softwares de podcasting, como o Miro [28], passaram a usar o protocolo como forma de lidar com a grande quantidade de downloads de programas online;
- o site da gravadora DGM Live fornece o conteúdo via torrent após a venda [10];
- VODO [38] é um site de divulgação e distribuição de filmes sob a licença Creative Commons e que faz a publicação em outros sites de busca de torrents;
- canais como a americana CBC [7] e a holandesa VPRO [9] já disponibilizaram programas de sua grade para download. A norueguesa NRK o faz para conteúdos em HD [31] e, apesar de algumas restrições de direitos, tem aumentado a oferta;
- o serviço Amazon S3, de armazenamento de conteúdo via web service, permite o uso de torrent para a transmissão de arquivos [1];

- as empresas de desenvolvimento de jogos CCP Games (Eve Online) e Blizzard (Diablo III, StarCraft II e World of Warcraft) usaram o protocolo para distribuir o instalador de seu jogo [2], e distribuir os jogos e suas eventuais atualizações [6], respectivamente;
- o governo britânico distribuiu os detalhes de seus gastos [15], enquanto a Universidade do Estado da Flórida utiliza para transmitir grandes conjuntos de dados científicos aos seus pesquisadores [19];
- Facebook [12] e Twitter [14] o usam para atualizar os seus sites, enviando de forma eficiente o código novo para seus servidores de aplicação [13].

Em 2013, o BitTorrent é um dos maiores geradores de tráfego de rede do mundo, de forma crescente, ao lado do NetFlix, Youtube, Facebook e acessos HTTP [20].

Questões legais

Desde que surgiu, o BitTorrent, bem como os outros protocolos P2P, chamou a atenção dos defensores de direitos autorais, por conta do compartilhamento não autorizado de arquivos protegidos por tais direitos, sendo alvo de medidas judiciais. Porém, assim como o Gnutella, e ao contrário do Napster, por possuir uma estrutura descentralizada e não armazenar dados sobre os compartilhamentos realizados, dificulta o trabalho de identificação das pessoas que compatilham esses dados.

Ainda assim, não existe um consenso sobre os efeitos financeiros do compartilhamento de arquivos protegidos por direitos autorais, onde o principal argumento utilizado pelos reclamantes é que estes têm grandes prejuízos e, por isso, entram com ações indenizatórias de valores vultosos. Existem alguns estudos que tentam medir esses prejuízos; um dos mais recentes, mostrou que não existem evidências de diminuição das receitas das empresas cujo conteúdo é pirateado, e que o combate aos usuários infratores não tem o impacto esperado, que é o de reduzir o compartilhamento desses arquivos [33].

Estudos acadêmicos

Academicamente, o protocolo é bastante estudado desde o seu surgimento, sendo focos de pesquisa os efeitos do algoritmo original e ajustes finos de seu funcionamento. Os pontos principais são a parte algorítmica da troca de pedaços dos arquivos, estudos sobre as topologias das redes formadas pelos peers e melhoria da eficiência do protocolo com alterações nessas topologias.

Capítulo 3

Anatomia do BitTorrent

O BitTorrent é uma rede P2P, onde cada um de seus usuários assume o papel híbrido de servidor (que fornece os arquivos) e de cliente (que adquire os arquivos). Cada computador é chamado de peer.

Cada transferência por BitTorrent está associada a um arquivo de extensão .torrent, que contém metadados, que são informações sobre arquivos que constituem um pacote chamado torrent. Além disso, possui um ou mais endereços de trackers, que são servidores que mantém listas atualizadas dos peers que estão compartilhando aqueles arquivos, atualizadas em curtos períodos de tempo (usualmente 30 minutos).

Enquanto um peer estiver fazendo download de um torrent, ele é chamado de sugador (*leecher*), pois ainda consumirá dados de outros peers; quando o download acabar, passará a ser um semeador (*seeder*), que somente enviará dados.

Os arquivos .torrent ficam disponíveis em vários sites de índice (às vezes, chamados de comunidades), como o ThePirateBay, o Kickass ou Torrentz (muitas vezes em mais de um deles ao mesmo tempo). Apesar de todo conteúdo compartilhado possuir um arquivo .torrent, não necessariamente um arquivo .torrent está sendo compartilhado naquele momento, podendo até mesmo estar extinto.

Peers que participam do compartilhamento de um torrent específico fazem parte do swarm, onde os dados contidos nesse torrent são compartilhados por partes com outros peers.

A quantidade total de partes varia de acordo com cada torrent: o tamanho total dos arquivos contidos nesse torrent é dividido em blocos de tamanho fixo (geralmente 256kB) e transmitido de forma independente das outras partes, seguindo uma ordem estabelecida pelo algoritmo de troca de partes (explicado na seção 3.4).

Essa ordem varia de acordo com o estado atual do swarm desse torrent.

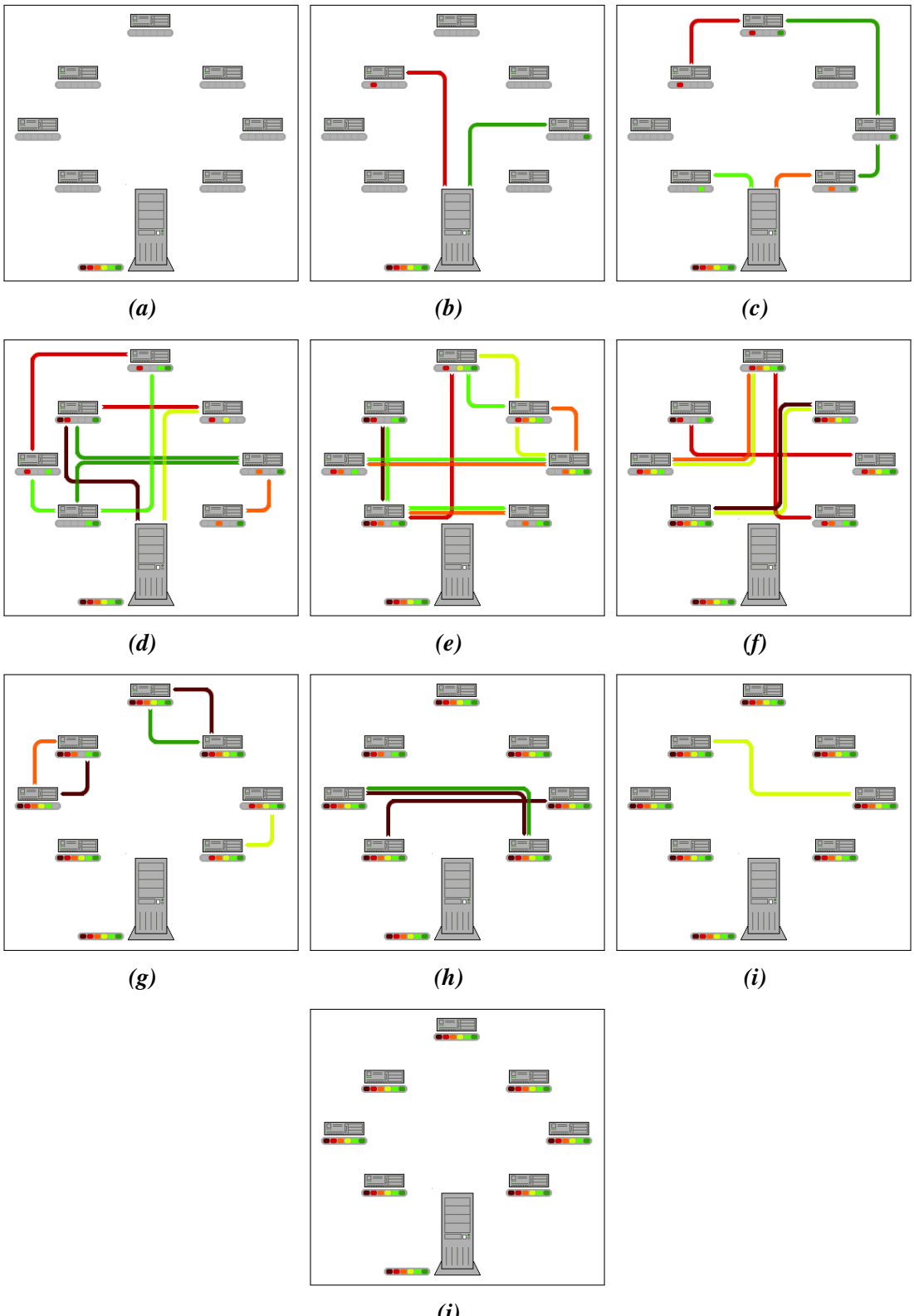


Figura 3.1: *simulação de uma transferência torrent: o seeder, na parte inferior das figuras, possui todas as 5 partes de um arquivo, que os outros computadores - os leechers - baixam de forma independente e paralela. Fonte: [63]*

Todos esses agentes possuem relações múltiplas entre si. Por exemplo, um mesmo arquivo .torrent pode estar indexado por vários sites indexadores. Como veremos nos capítulos seguintes, eles contêm uma informação que os identificam unicamente, gerando consistência entre esses vários sites de busca. Outra observação a ser feita é que um peer pode estar baixando um ou mais torrents simultaneamente, ou seja, participando de vários swarms ao mesmo tempo. Por fim, em alguns casos, um mesmo torrent pode possuir uma grande quantidade de peers participantes, havendo necessidade de dividí-los em vários swarms, para fins de escalabilidade da rede formada.

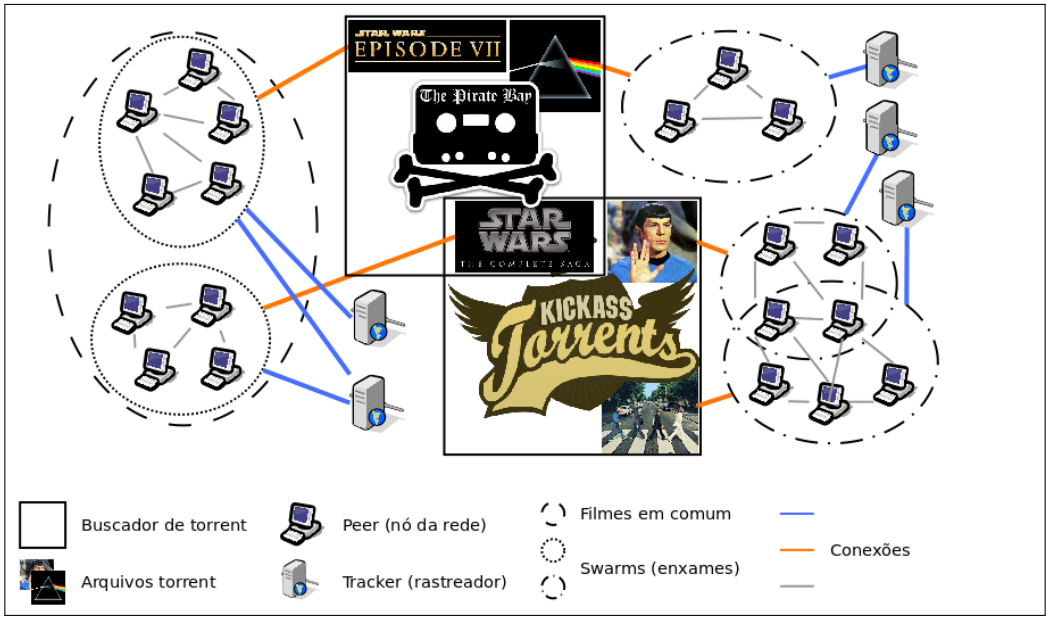


Figura 3.2: *amostra de uma rede de conexões BitTorrent*

Arquivo .torrent

Ao se adicionar um arquivo .torrent em um programa cliente, ocorrem muitas transmissões de dados antes do download de fato. Para demonstrar isso, usaremos um arquivo .torrent do filme "A Noite dos Mortos Vivos", de 1960 [29], que é de domínio público e livre de direitos autorais.

Se abrirmos esse arquivo, veremos uma grande sequência de caracteres (*string*), caracteres diferentes e incomuns, formando um conteúdo ilegível (na seção binária) e sob uma forma compacta, mostrado abaixo.

```
1  d8:announce36:http://bt1.archive.org:6969/announce13:announce-listll36:http://bt1.
2  archive.org:6969/announceel36:http://bt2.archive.org:6969/announceee7:comment13:crea
3  tiondatei1343715473e4:infod5:filesld5:crc328:030208fe6:lengthi4127671704e3:md532:627
4  f5a428f9e454ccfcb29d31b87169a5:mtime10:10794024804:pathl29:night_of_the_living_dead.
5  mpege4:sha140:5e44bb1b3f700240249a5287c64dc02dc56d034bee4:name24:night_of_the_living
6  _dead12:piecelengthi4194304e6:pieces23720:<binary>
7
8  (...)
9
10 e6:locale2:en5:title24:night_of_the_living_dead8:url-listl28:http://archive.org/
11 download/39:http://ia600301.us.archive.org/22/items39:http://ia700301.us.archive.
12 org/22/itemsee
```

Código-fonte 3.1: *trecho do conteúdo do arquivo .torrent do filme "A Noite dos Mortos Vivos", de 1960 [29], com a parte binária truncada*

Esse conteúdo está organizado usando a codificação B (bencode), que é uma codificação compacta de arquivos, especial para arquivos .torrent, e ininteligível. Com alguma formatação, podemos enxergar os componentes separadamente, como mostra o código 3.2.

Esse conteúdo tem significado, sendo utilizado da seguinte forma [64]:

- **strings** são prefixos de números na base 10, que representam comprimentos, seguidos por um caractere `:` e então o conteúdo. Por exemplo, na linha 2, `8:announce` corresponde à string `"announce"`.
- **números** são representados por um `i`, seguidos do valor na base 10 (sem qualquer limite de tamanho, mas sem zeros precedentes - como em `0003` - e pode ser negativo), terminados por um `e`. Por exemplo, na linha 11, `i1343715473e` corresponde ao número `1343715473`.
- **listas** são formadas por `l`, seguidos por seus elementos (também no formato bencode), e então terminados por `e`. Por exemplo, `l3:foo3:bare` corresponde a `["foo", "bar"]`. No código 3.2, é presente entre as linhas 43 e 47.
- **dicionários** são definidos por `d`, seguidos de uma lista alternada de chaves e seus valores correspondentes, terminando com `e`, onde as chaves devem estar ordenadas

usando-se comparação binária, ao invés da usual alfanumérica. Por exemplo, a string `d3:foo3:bar6:foobar6:bazbare` corresponde ao dicionário puro `{"foo": "bar", "foobar": "bazbar"}`, e a estrutura mais complexa dada por `d3:fool6:foobar3:bazee` equivale a `{"foo": ["foobar", "baz"]}`.

```
d
    8:announce
    36:http://bt1.archive.org:6969/announce
    13:announce-list
    l
        l36:http://bt1.archive.org:6969/announcee
        l36:http://bt2.archive.org:6969/announcee
    e
    7:comment
    13:creation date
    i1343715473e
    4:info
    d
        5:files
        l
            d
                5:crc32
                8:030208fe
                6:length
                i4127671704e
                3:md5
                32:627f5a428f9e454ccfcb29d31b87169a
                5:mtime
                10:1079402480
                4:path
                l29:night_of_the_living_dead.mpege
                4:sha1
                40:5e44bb1b3f700240249a5287c64dc02dc56d034b
            e
        e
        4:name
        24:night_of_the_living_dead
        12:piece length
        i4194304e
        6:pieces
        23720:<binary>
    e
    6:locale
    2:en
    5:title
    24:night_of_the_living_dead
    8:url-list
    l
        28:http://archive.org/download/
        39:http://ia600301.us.archive.org/22/items
        39:http://ia700301.us.archive.org/22/items
    e
e
```

Código-fonte 3.2: *trechos formatados de forma legível do conteúdo do arquivo .torrent do filme "A Noite dos Mortos Vivos", de 1960 [29], com a parte binária truncada*

Magnet Link

Além do arquivo .torrent, existe uma outra forma de se obter os metadados necessários para se iniciar a transmissão, utilizando-se de links magnéticos (magnet links).

Magnet links, ao contrário dos arquivos .torrent, não estão gravados em algum dispositivo de armazenamento. Basicamente, é um esquema de identificador uniforme de recursos (URI), usado exclusivamente para o protocolo.

No caso citado, o site de origem do arquivo .torrent que estamos usando não fornece um magnet link oficialmente. Porém, o Transmission consegue construir uma URI a partir do arquivo original, para fins de compartilhamento direto. O resultado, após decodificar o endereço para um formato legível (retirando a codificação de endereços URI e URL (*URL encode*)) [59], foi o seguinte:

```
magnet:?xt=urn:btih:72d7a3179da3de7a76b98f3782c31843e3f818ee
&dn=night_of_the_living_dead
&tr=http://bt1.archive.org:6969/announce&tr=http://bt2.archive.org:6969/announce
&ws=http://archive.org/download/
&ws=http://ia600301.us.archive.org/22/items/&ws=http://ia700301.us.archive.org/22/items/
```

Código-fonte 3.3: *link magnético do arquivo .torrent do filme "A Noite dos Mortos Vivos", de 1960 [29], com parâmetros divididos entre linhas para melhor visualização*

Esse endereço é composto por vários pares, compostos por nomes de parâmetros e seus respectivos valores, sem qualquer ordem específica, formando uma string de busca (*query string*). Podemos dividir esse endereço em partes, cada uma tendo o seu significado:

- **xt**: parâmetro para *exact topic*, ou tópico exato, que contém a informação mais importante do magnet link: o identificador único de torrents. Serve para encontrar e verificar os arquivos especificados. No caso, `urn:btih:<hash>` corresponde ao nome uniforme de recursos (URN) `btih` (*BitTorrent Info Hash*), que é a string valor hash resultado da função de hash SHA-1, convertida para hexadecimal;
- **dn**: parâmetro que contém o *display name*, ou nome de visualização, que é um texto de apresentação amigável para o usuário;
- **tr**: o *address tracker*, ou endereço do tracker, onde o programa cliente vai procurar as informações de peers;
- **ws**: endereço do arquivo para *webseed*, ou fornecimento web, que é o endereço de Internet de um servidor HTTP ou FTP, que será utilizado como alternativa a um swarm problemático [44];

3.1 Busca por informações

Quando adicionamos um torrent ao Transmission, o programa salva as informações em disco durante todo o período em que estas estiverem sendo gerenciadas por ele. Caso tenha sido por meio de um arquivo .torrent, uma cópia deste é salva em uma pasta pré-definida para seu controle interno; caso seja por magnet link, um novo arquivo é criado contendo as informações obtidas através dele (por questões de praticidade), para que não necessite fazer essa aquisição dos dados novamente ao ser aberto, ou quando alguma transferência for pausada e depois continuada.

Após esse arquivamento, o programa processa as informações salvas para deixá-las carregadas em memória, a fim de obter o valor hash que identifica o arquivo .torrent.

./libtransmission/metainfo.c:367

```
static const char*
tr_metainfoParseImpl(const tr_session * session, tr_info * inf,
bool * hasInfoDict, int * infoDictLength, const tr_variant * meta) {
  // variáveis temporárias
  int64_t i; size_t len; const char * str; const uint8_t * raw;
  tr_variant * infoDict = NULL;

  // flags
  bool b, isMagnet = false;

  /* info_hash: urlencoded 20-byte SHA1 hash of the value of the info key from the
   * Metainfo file. (...) */
  b = tr_variantDictFindDict(meta, TR_KEY_info, &infoDict);
  if (hasInfoDict != NULL) *hasInfoDict = b;
```

Se aquele arquivo para controle interno tiver sido criado por conta de um magnet link, não possuirá a chave `info` em seu dicionário, mas deverá conter as chaves `urn:btih:<hash>` e `info_hash`.

./libtransmission/metainfo.c:367

```
  if (!b) { // Não possui a chave "info" no dicionário.
    // Será que não é um magnet link?
    if (tr_variantDictFindDict(meta, TR_KEY_magnet_info, &d)) {
        isMagnet = true;

        if (!tr_variantDictFindRaw(d, TR_KEY_info_hash, &raw, &len) ||
            len != SHA_DIGEST_LENGTH) // Se tiver a chave "info_hash" válida, a usa.
          return "info_hash";

        memcpy(inf->hash, raw, len);
        tr_sha1_to_hex(inf->hashString, inf->hash);
        (...)
    }
    else return "info"; // Não é magnet link e não possui a chave "info".
  }
```

Porém, se o arquivo para controle interno tiver sido criado como cópia do arquivo .torrent, possuirá o mesmo dicionário, que contém a chave `info_hash`, que é utilizada para calcular o valor hash do arquivo.

./libtransmission/metainfo.c:367

```
  else {
    int len;
    char * bstr = tr_variantToStr(infoDict, TR_VARIANT_FMT_BENC, &len);
    tr_sha1(inf->hash, bstr, len, NULL);          // Calcula o hash SHA-1 do .torrent...
    tr_sha1_to_hex(inf->hashString, inf->hash);   // ...e converte de base2 para base16
    (...)
  }
```

Outras informações podem ser recuperadas, dependendo da origem do arquivo .torrent, tais como a privacidade do torrent, a lista de arquivos e seus respectivos tamanhos, valor hash de cada parte, entre outras. Por fim, termina coletando os endereços web de contato do tracker (*announces*).

./libtransmission/metainfo.c:367

```
  if (!tr_variantDictFindInt(infoDict, TR_KEY_private, &i)) // privacidade do torrent
    if (!tr_variantDictFindInt(meta, TR_KEY_private, &i)) i = 0;
  inf->isPrivate = i != 0;

  if (!isMagnet) { // quantidade de partes
    if (!tr_variantDictFindInt(infoDict, TR_KEY_piece_length, &i) || (i < 1))
      return "piece length";
    inf->pieceSize = i;
  }

  if (!isMagnet) { // hashes das partes
    if (!tr_variantDictFindRaw(infoDict, TR_KEY_pieces, &raw, &len) ||
        len % SHA_DIGEST_LENGTH) return "pieces";

    inf->pieceCount = len / SHA_DIGEST_LENGTH;
    inf->pieces = tr_new0(tr_piece, inf->pieceCount);
    for (i = 0; i < inf->pieceCount; i++)
      memcpy(inf->pieces[i].hash, &raw[i * SHA_DIGEST_LENGTH], SHA_DIGEST_LENGTH);
  }

  if (!isMagnet) { // lista de arquivos
    if ((str = parseFiles(inf, tr_variantDictFind(infoDict, TR_KEY_files),
                               tr_variantDictFind(infoDict, TR_KEY_length))))
      return str;

    if (!inf->fileCount || !inf->totalSize ||
        (uint64_t) inf->pieceCount != (inf->totalSize+inf->pieceSize-1)/inf->pieceSize)
      return "files";
  }

  if ((str = getannounce(inf, meta))) return str; // announce(s)
  (...)
  return NULL;
}
```

Announce

Para cada swarm gerenciado, o tracker possui uma lista dos peers que participam dele, que é enviada ao peer que a requer por meio de uma requisição de método GET do protocolo HTTP (HTTP GET). Quando essa requisição é recebida pelo tracker, este incluirá ou atualizará um registro para o peer solicitante, e devolverá uma lista de 50 peers aleatórios, de forma uniforme, que fazem parte do swarm. Não havendo essa quantidade total, a lista toda será enviada ao

requisitante. Caso contrário, a aleatoriedade proporcionará uma diversidade de listas enviadas, ocasionando robustez ao sistema [65].

Esse contato entre um peer e um tracker é chamado de announce, que pode ser feito usando-se tanto o protocolo TCP, bem como o protocolo UDP, e é como peers podem passar várias informações, usando um dicionário no formato bencode:

- **info_hash**: valor hash de 20 bytes resultante da função de hash SHA-1, com URL encode, do valor da chave `info` do arquivo arquivo .torrent;
- **peer_id**: string de 20 bytes, com URL encode, usado como identificador único do programa cliente, gerado no ínicio da sua execução. Para isso, provavelmente deverá incorporar informações do computador, a fim de se gerar um valor único;
- **uploaded**: a quantidade total de dados, em bytes, enviados desde o momento em que o cliente enviou o primeiro aviso ao tracker;
- **downloaded**: a quantidade total de dados, em bytes, recebidos desde momento em que o cliente enviou o primeiro aviso ao tracker;
- **left**: a quantidade total de dados, em bytes, que faltam para o requisitante terminar o download do torrent e passe a ser um seeder;
- **compact** (opcional): se o valor passado for 1, significa que o requisitante aceita respostas compactas. A lista de peers enviada é substituída por uma única string de peers, sendo que cada peer terá 6 bytes, onde os 4 bytes iniciais são o host e os 2 bytes finais são a porta de transmissão. Por exemplo, o endereço IP 10.10.10.5:80 seria transmitido como `0A 0A 0A 05 00 80`. Deve-se observar que alguns trackers suportam somente conexões deste tipo para otimização da utilização da banda de rede e, para isso, ou recusarão requisições sem `compact=1` ou, caso não as recusem, enviarão respostas compactas a menos que a requisição possua `compact=0`;
- **no_peer_id** (opcional): sinaliza que o tracker pode omitir o campo `peer_id` no dicionário de peers, porém será ignorado caso o modo compacto esteja habilitado;
- **event** (opcional): pode possuir os valores `started` (iniciado), `completed` (terminado), `stopped` (parado), ou vazio para não especificar.
  - *started* : a primeira requisição para o tracker deve enviar este valor;
  - *stopped* : avisa que o programa cliente está fechando;
  - *completed* : quando o download que estava ocorrendo termina numa mesma execução do programa cliente (não é enviado quando o programa cliente é iniciado com o torrent em 100%);
- **port** (opcional): o número da porta de conexão que o programa cliente está escutando por transmissões de dados. Em geral, portas reservadas para BitTorrent estão entre 6881 e 6889. Se esse for o caso, pode ser omitido;

- **ip** (opcional): o endereço IP verdadeiro do requisitante, no formato legível do IPv4 (4 conjuntos de número de 0 a 255 separados por `.`) ou do IPv6 (8 conjuntos de números hexadecimais de 4 dígitos separados por `:`). Não é sempre necessário, pois o endereço pode ser conhecido através da requisição. Assim, é usado quando o programa cliente está se comunicando com o tracker através de um servidor *proxy* ou quando ambos (cliente e tracker) estão no mesmo lado local de um roteador - com tradução de endereço de rede (*gateway NAT*) -, pois, nesse caso o endereço IP não é roteável;

- **numwant** (opcional): quantidade de peers que o requisitante gostaria de receber do tracker. É permitido valor zero. Se omitido, assumirá valor padrão de 50;

- **key** (opcional): mecanismo de identificação adicional para o programa cliente provar sua identidade, caso tenha ocorrido mudança no seu endereço IP;

- **trackerid** (opcional): se a resposta de um announce anterior continha o endereço IP de um tracker, deve ser enviado neste campo;

./libtransmission/announcer-common.h:127

```
typedef struct {
  (...)
  bool partial_seed;
  int port;                   // porta escutada para aguardo de chamada de peers
  int key;                    // chave de sessão
  int numwant;                // qtd de peers que o cliente deseja receber do tracker
  uint64_t up;                // qtd bytes enviados desde o último evento 'started'
  uint64_t down;              // qtd bytes "bons" recebidos desde o último 'started'
  uint64_t corrupt;           // qtd bytes corrompidos recebidos desde o último 'started'
  uint64_t leftUntilComplete; // qtd de bytes que faltam para o fim do download
  char * url;                 // a URL de announce do tracker
  char * tracker_id_str;      // chave gerada por um tracker HTTP e devolvida
  char peer_id[PEER_ID_LEN];  // o ID do peer
  uint8_t info_hash[SHA_DIGEST_LENGTH]; // info_hash do torrent
}
tr_announce_request;
```

./libtransmission/announcer.c:1200

```
static void announce_request_delegate(tr_announcer * announcer,
    tr_announce_request * request, tr_announce_response_func * callback,
    void * callback_data) {
  (...)
  if (!memcmp(request->url, "http", 4))        // announce começa com "http://"
    tr_tracker_http_announce(session, request, callback, callback_data);
  else if (!memcmp(request->url, "udp://", 6)) // announce começa com "udp://"
    tr_tracker_udp_announce(session, request, callback, callback_data);
  else
    tr_logAddError("Unsupported url: %s", request->url);
  (...)
}
```

Como resposta, é recebida um outro dicionário em bencode, podendo conter as seguintes chaves:

- **failure_reason**: se presente, não podem existir outras chaves no dicionário. Seu valor é uma string de mensagem de erro legível sobre o porque a requisição falhou;

- **warning_message** (opcional): similar à chave `failure\_reason`, mas com a requisição tendo sido processada normalmente. A mensagem é mostrada como um erro;

- **interval**: intervalo, em segundos, que o cliente deve esperar emtre requisições de announce ao tracker;
- **min_interval** (opcional): intervalo mínimo, em segundos, entre requisições de announce. Se presente, o pragrama cliente não deve efetuar essas requisições acima da frequência estipulada;
- **tracker_id**: string que o programa cliente deve enviar de volta nas próximas requisições. Se ausente e um valor tiver sido passado anteriormente, o uso desse valor antigo é continuado;
- **complete**: quantidade de seeders;
- **incomplete**: quantidade de leechers;
- **peers**: pode ser uma das opções
  1. lista de dicionários bencode, com as seguintes chaves:
     - **peer_id**: identificador de um peer na forma de string, escolhido por si próprio da mesma forma que a descrito pela definição de requisição;
     - **ip**: endereço IP do peer nos formatos IPv4 (4 octetos) ou IPv6 (valores hexadecimais), ou ainda o nome de domínio DNS (string);
     - **port**: número da porta utilizada pelo peer;
  2. string binária cujo tamanho é de 6 bytes para cada peer, onde os 4 primeiros representam o endereço IP e os 2 últimos são o número da porta, em notação de rede (*big endian*);

```
* About to connect() to exodus.desync.com port 6969 (#8)
*   Trying 208.83.20.164...
*
* Connected to exodus.desync.com (208.83.20.164) port 6969 (#8)
> GET /announce?info_hash=\%88\%15\%8c\%7bW\%e0\%85\%21\%86~\%d0\%b5\%de\%06\%5b\%
7dWI\%cf\%d7&peer_id=-TR2820-ne1joqgh8z9o&port=51413&uploaded=0&downloaded=0&left=52
406288292&numwant=80&key=6ee99240&compact=1&supportcrypto=1&requirecrypto=1&event=
started HTTP/1.1
User-Agent: Transmission/2.82
Host: exodus.desync.com:6969
Accept: */*
Accept-Encoding: gzip;q=1.0, deflate, identity

< HTTP/1.1 200 OK
< Content-Type: text/plain
< Content-Length: 136
<
* Connection #8 to host exodus.desync.com left intact
Announce response:
< {
    "complete": 1,
    "downloaded": 11,
    "incomplete": 6,
    "interval": 1732,
    "min interval": 866,
    "peers": \"<binary>\"
}
```

Código-fonte 3.4: *Logs do Transmission sobre uma requisição de announce e a respectiva resposta, com o conteúdo binário truncado*

Essa comunicação ocorre nas seguintes situações:

- no primeiro contato do peer, para que ele tenha acesso a um swarm;
- a cada período de tempo, estipulado pelo tracker, para que o peer continue mostrando que ainda está conectado, além de poder receber endereços de peers novos;
- quando a quantidade de peers conhecidos que estiverem ativos for menor do que 5;
- quando terminar o download, notificando que passou a ser um seeder;
- quando sair do swarm, seja por desconexão ou por encerramento do programa cliente;

Scrape

Além do announce, outra forma de troca de informação entre peers e trackers se dá pelo endereço web do tracker para informações básicas (*scrape*). Geralmente usado pelos programas cliente para decidir quando realizar um announce, informa o número de peers, leechers e seeders de uma lista de um ou mais torrents. É dessa forma que os sites de indexação sabem dessas informações e as apresentam nas em páginas.

A requisição de scrape pode ser sobre todos os torrents que o tracker gerencia ou sobre um ou mais torrents em específico, quando são passados seus respectivos valores hash. Sua resposta é um dicionário na forma bencode contendo as chaves

- **files**: um dicionário contendo um par chave-valor para cada torrent especificado na requisição do scrape através do valor hash do torrent de 20 bytes:
  - **complete**: quantidade de seeders;
  - **incomplete**: quantidade de leechers;
  - **downloaded**: quantidade total de vezes que o tracker registrou uma finalização (`event=complete` ao término de um download);
  - **name** (opcional): o nome interno do torrent, como especificado pelo respectivo arquivo .torrent, na sua seção `info`;

Da mesma forma que o announce, o scrape é um endereço do tracker usando-se o TCP ou o UDP, e é tratado de forma semelhante pelo Transmission.

./libtransmission/announcer-common.h:37

```
typedef struct {
  // lista de info_hash de torrents para serem pesquisados por scrape
  uint8_t info_hash[TR_MULTISCRAPE_MAX][SHA_DIGEST_LENGTH];
  char * url;  // URL de scrape
  (...)
} tr_scrape_request;

struct tr_scrape_response_row { // usado para conter os dados respondidos do scrape
  uint8_t info_hash[SHA_DIGEST_LENGTH]; // info_hash do torrent
  int seeders;     // qtd de peers que são seeders deste torrent
  int leechers;    // qtd de peers baixando este torrent
  int downloads;   // qtd de vezes que este torrent foi baixado
  int downloaders; // qtd de leechers ativos no swarm (suportado por alguns trackers)
};
```

./libtransmission/announcer.c:1397

```
static void scrape_request_delegate(tr_announcer * announcer,
    const tr_scrape_request * request, tr_scrape_response_func * callback,
    void * callback_data) {
  (...)
  if (!memcmp(request->url, "http", 4))
    tr_tracker_http_scrape(session, request, callback, callback_data);
  else if (!memcmp(request->url, "udp://", 6))
    tr_tracker_udp_scrape(session, request, callback, callback_data);
  else
    tr_logAddError("Unsupported url: %s", request->url);
}
```

```
* About to connect() to www.mvgroup.org port 2710 (#1)
*   Trying 88.129.153.50...
* Connected to www.mvgroup.org (88.129.153.50) port 2710 (#1)
> GET /scrape?info_hash=\%83F\%24\%b62\%e5Q\%cc4\%10h\%ba\%1e8\%e2C\%f7\%80\%01\%87 HTTP/1.1
User-Agent: Transmission/2.82
Host: www.mvgroup.org:2710
Accept: */*
Accept-Encoding: gzip;q=1.0, deflate, identity

* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
<
* Closing connection 1
Scrape response:
< {
    "files": {
        \"<binary>\": {
            "complete": 9,
            "downloaded": 59074,
            "incomplete": 2
        }
    }
}
```

Código-fonte 3.5: *Logs do Transmission sobre uma requisição de scrape e a respectiva resposta, com o conteúdo binário truncado*

Convenção de scrape de trackers

O endereço de scrape pode ser obtido a partir do endereço de announce, seguindo-se a seguinte convenção:

1. comece a partir do endereço de announce
2. encontre o último caractere `/`
3. se o texto que segue a última `/` não for `announce`, é um sinal que o tracker não segue a convenção
4. caso contrário, substituir `announce` por `scrape`

Alguns exemplos:

- suportam `scrape`
  1. `http://example.com/announce` → `http://example.com/scrape`
  2. `http://example.com/announce.php` → `http://example.com/scrape.php`
  3. `http://example.com/x/announce` → `http://example.com/x/scrape`
  4. `http://example.com/announce?x2%0644` → `http://example.com/scrape?x2%0644`
- não suportam `scrape`
  1. `http://example.com/a`
  2. `http://example.com/announce?x=2/4`
  3. `http://example.com/x%064announce`

Resultados de announce e scrape

Com a requisição do announce, o Transmission recebe a sua resposta e prepara esses dados para utilização, carregando-os em memória.

./libtransmission/announcer-http.c:288

```
1 void tr_tracker_http_announce(tr_session * session, const tr_announce_request * request,
2         tr_announce_response_func response_func, void * response_func_user_data) {
3     char * url = announce_url_new(session, request);
4     struct announce_data * d = tr_new0(struct announce_data, 1);
5     (...) // preenchimento dos campos da variável 'd'
6
7     tr_webRun(session, url, on_announce_done, d);
8     (...)
9 }
```

./libtransmission/announcer-http.c:189

```
static void on_announce_done(tr_session * session, bool did_connect, bool did_timeout,
    long response_code, const void * msg, size_t msglen, void * vdata) {
    tr_announce_response * response;
    struct announce_data * data = vdata;
    response = &data->response;
    (...)

    // lê os dados da resposta do tracker
    if (variant_loaded && tr_variantIsDict(&benc)) {
        int64_t i; size_t len; tr_variant * tmp; // variáveis ...
        const char * str; const uint8_t * raw;   // ... temporárias

        if (tr_variantDictFindStr(&benc, TR_KEY_failure_reason, &str, &len))
            response->errmsg = tr_strndup(str, len);

        if (tr_variantDictFindStr(&benc, TR_KEY_tracker_id, &str, &len))
            response->tracker_id_str = tr_strndup(str, len);

        if (tr_variantDictFindInt(&benc, TR_KEY_complete, &i))
            response->seeders = i;

        (...)

        if (tr_variantDictFindRaw(&benc, TR_KEY_peers, &raw, &len)) {
            response->pex = tr_peerMgrCompactToPex(raw, len,
            NULL, 0, &response->pex_count);
        }
        else if (tr_variantDictFindList(&benc, TR_KEY_peers, &tmp)) {
            response->pex = listToPex(tmp, &response->pex_count);
        }
    }

    tr_runInEventThread(session, on_announce_done_eventthread, data);
}
```

Após essa preparação dos dados da resposta do announce, cuja informação principal é a lista de peers, o Transmission gerencia o tempo para a próxima requisição periódica de announce ou scrape. Além disso, sinaliza para a thread principal do programa que recebeu a resposta do tracker.

./libtransmission/announcer.c:1012

```
static void on_announce_done(const tr_announce_response * response, void * vdata) {
    struct announce_data * data = vdata;
    (...)
    tr_tier * tier = getTier(announcer, response->info_hash, data->tierId);
    const time_t now = tr_time();

    (...)

    if (tier != NULL) {
        tr_tracker * tracker;
        tier->lastAnnounceTime = now;
        tier->lastAnnounceTimedOut = response->did_timeout;
        tier->lastAnnounceSucceeded = tier->isAnnouncing = false;
        tier->manualAnnounceAllowedAt = now + tier->announceMinIntervalSec;

        (...)
        int i = scrape_fields = seeders = leechers = 0; const char * str;

        (...)

        if (response->seeders >= 0) {
            tracker->seederCount = seeders = response->seeders;
            (...)
        }
        if (response->leechers >= 0) {
            tracker->leecherCount = leechers = response->leechers;
            (...)
        }

        (...)

        if ((i = response->min_interval)) tier->announceMinIntervalSec = i;
        if ((i = response->interval)) tier->announceIntervalSec = i;

        if (response->pex_count > 0)
            publishPeersPex(tier, seeders, leechers, response->pex, response->pex_count);

        (...)
        tier->lastAnnounceSucceeded = true;
        (...)
    }
    (...)
}
```

./libtransmission/announcer.c:523

```
static void publishPeersPex(tr_tier * tier, int seeds, int leechers, const tr_pex * pex,
    int n) {
    if (tier->tor->tiers->callback) {
        tr_tracker_event e = TRACKER_EVENT_INIT;
        e.pex = pex; e.pexCount = n; e.messageType = TR_TRACKER_PEERS;
        (...)
        tier->tor->tiers->callback(tier->tor, &e, NULL);
    }
}
```

A thread principal percebe que foi sinalizado o recebimento de uma resposta de tracker e começa a utilizar a lista de peers adquirida, adicionando-os ao poço de recursos (*pool*) do objeto em memória que representa o swarm.

./libtransmission/torrent.c:552

```
static void onTrackerResponse(tr_torrent * tor, const tr_tracker_event * event,
    void * unused UNUSED) {
    switch (event->messageType) {
        case TR_TRACKER_PEERS: {
            size_t i;
            (...)
            for (i = 0; i < event->pexCount; ++i)
                tr_peerMgrAddPex(tor, TR_PEER_FROM_TRACKER, &event->pex[i], seedProbability);
            break;
        }
        (...)
    }
}
```

./libtransmission/peer-mgr.c:2113

```
void tr_peerMgrAddPex(tr_torrent * tor, uint8_t from, const tr_pex * pex,
    int8_t seedProbability) {
    if (tr_isPex(pex)) { /* safeguard against corrupt data */
        tr_swarm * s = tor->swarm;
        managerLock(s->manager);

        if (!tr_sessionIsAddressBlocked(s->manager->session, &pex->addr))
            if (tr_address_is_valid_for_peers(&pex->addr, pex->port))
                ensureAtomExists(s, &pex->addr, pex->port, pex->flags, seedProbability, from);

        managerUnlock(s->manager);
    }
}
```

Essa inclusão é condicionada ao fato de que o endereço de um peer ainda não está contido na lista de peers.

./libtransmission/peer-mgr.c:1852

```
static void ensureAtomExists(tr_swarm * s, const tr_address * addr, const tr_port port,
    const uint8_t flags, const int8_t seedProbability, const uint8_t from) {
    struct peer_atom * a = getExistingAtom(s, addr);

    (...)

    if (a == NULL) { // Se ainda não está na lista...
        const int jitter = tr_cryptoWeakRandInt(60 * 10);
        a = tr_new0(struct peer_atom, 1);
        a->addr = *addr;
        a->port = port;
        (...)

        // ...adiciona peer na lista de peers, de forma ordenada.
        tr_ptrArrayInsertSorted(&s->pool, a, compareAtomsByAddress);
    }
    (...)
}
```

Assim, a lista de peers do Transmission está pronta para ser utilizada nos pedidos das partes do torrent.

3.2 Tabelas Hash Distribuídas e o Kademlia

Os DHTs surgiram quando buscas em redes peer-to-peer não eram eficientes, fosse pelos problemas da supercentralização dos sistemas ou pela falta dela, tentando criar uma maneira híbrida de buscar e localizar recursos nessas redes.

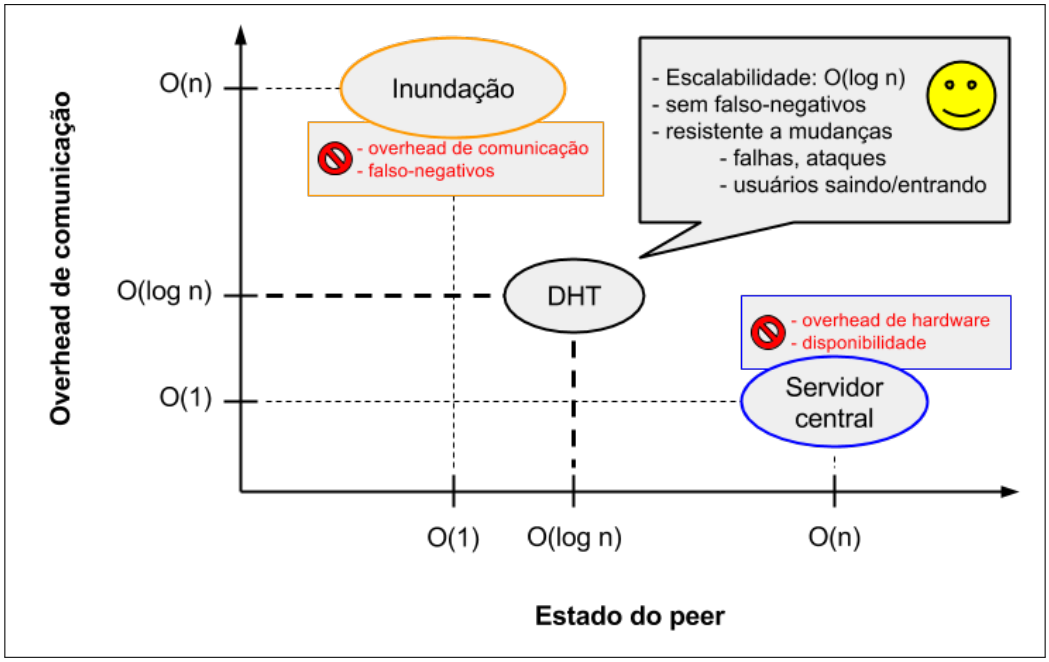
Existem duas formas de se encontrar objetos: busca e endereçamento. A primeira se baseia no casamento de palavras-chave com as das descrições dos objetos, sendo mais amigável para um usuário pois não necessita de formas complexas de identificação. Porém, é mais difícil de se tornar eficiente, além de precisar conferir também os objetos para saber se são o mesmo. As redes peer-to-peer descentralizadas utilizavam desta abordagem.

A segunda, no entanto, utiliza endereços que possam identificá-los de forma única. Dessa forma, um objeto é exclusivamente identificável, sendo possível encontrá-lo eficientemente. Contudo, é necessário algum procedimento de se conhecer seu nome único, além de ser requerido manter uma estrutura organizada por endereços. Historicamente, as redes peer-to-peer de estrutura centralizada se baseavam neste método.

Uma outra maneira de organizar esses objetos através da rede é de organizar as conexões entre os diversos peers, que acabam formando redes superpostas (*overlay networks*), que são redes virtuais que figuram sobre as tradicionais redes IP.

Anteriormente ao BitTorrent, nas redes peer-to-peer de estrutura centralizada, o servidor central tentava conhecer a situação de cada um dos peers da rede, enviando mensagens diretamente a eles. Gerenciar muitos peers ao mesmo tempo sobrecarregava o *hardware* do servidor, ou forçava o seu desligamento. Por outro lado, quando a rede tinha estrutura descentralizada, ocorria o fenômeno chamado de inundação de mensagens, no qual estas eram repassadas entre peers intermediários até o peer de destino. Esse repasse excessivo acarretava sobrecarga de processamento (*overhead*) dos intermediários, além de gerar mensagens falso-negativas.

Assim, as redes peer-to-peer necessitam de funcionalidades eficientes para que um peer possa entrar e sair do *overlay*, assim como armazenar objetos e encontrá-los (usando endereçamento). Tal eficiência é conseguida através de DHTs, onde todas essas funcionalidades dependem da troca de mensagens eficiente entre peers.



A funcionalidade do DHT passou anos sendo utilizado extraoficialmente à especificação do protocolo, quando, enfim, foi adicionada em 2008 [25].

Kademlia

O Kademlia é um DHT criado em 2002 [26] com o objetivo de melhorar os métodos de busca atuais (Napster e Gnutella), que eram ineficientes. Assim como os outros algoritmos de DHT, ele se baseou na estrutura informalmente conhecida como "rede de Plaxton" (*Plaxton mesh*), nome que remete a um dos seus autores [32]. Por ter mostrado bons resultados, foi usado na implementação da busca de arquivos no programa cliente eMule.

A sua modelagem computacional monta um mapa no formato de tabela hash onde IDs de peers ou de torrents são chaves para listas de outros peers.

Estrutura de dados

O algoritmo implementa uma rede *overlay* cuja estrutura e comunicação se baseiam na procura de seus nós. Cada um destes nós é identificado por um identificador único (ID), que serve tanto para a identificação quanto para a localização de valores na tabela hash.

Essa tabela hash é na forma de uma árvore binária, cujas folhas são os nós da rede. Cada

folha tem suas posições estabelecidas pelo menor prefixo comum de seus IDs, organizando-os de forma que, para um dado nó x, a árvore é dividida em várias subárvores menores que não o contém. Assim, a maior subárvore consiste de metade da árvore que não contém x, a subárvore seguinte é feita da metade da árvore restante onde x também não está contido, etc. O Kademlia garante ainda que todo nó conhece um outro que está em cada uma das subárvores, se estas contiverem algum nó.

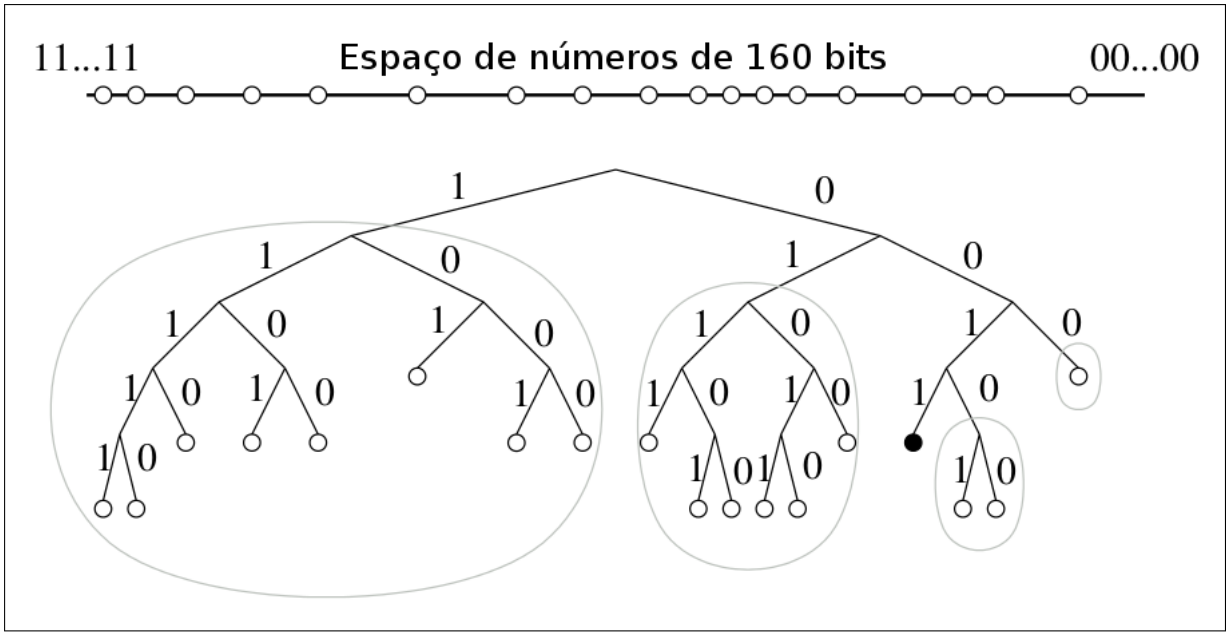


Figura 3.3: *Árvore binária do Kademlia. O nó preto é a posição do ID 0011...; os ovais cinzas são as subárvores onde o nó preto deve possuir nós conhecidos. Fonte: [26]*

No Kademlia, objetos e nós possuem IDs únicos de 160 bits: enquanto o primeiro utiliza o valor hash de 20 bytes SHA-1 da chave `info_hash` do arquivo .torrent, o segundo é um valor aleatório escolhido pelo próprio programa.

Durante uma busca por peers de um torrent, o processo deve conhecer a chave associado ao objeto, ou seja, o ID, e explora a rede em passos. A cada passo, encontra nós mais próximos da chave, até chegar ou ao valor buscado ou não nós existirem mais próximos que o atual. Dessa forma, para uma rede com n nós, o algoritmo visita apenas $O(\log n)$ nós.

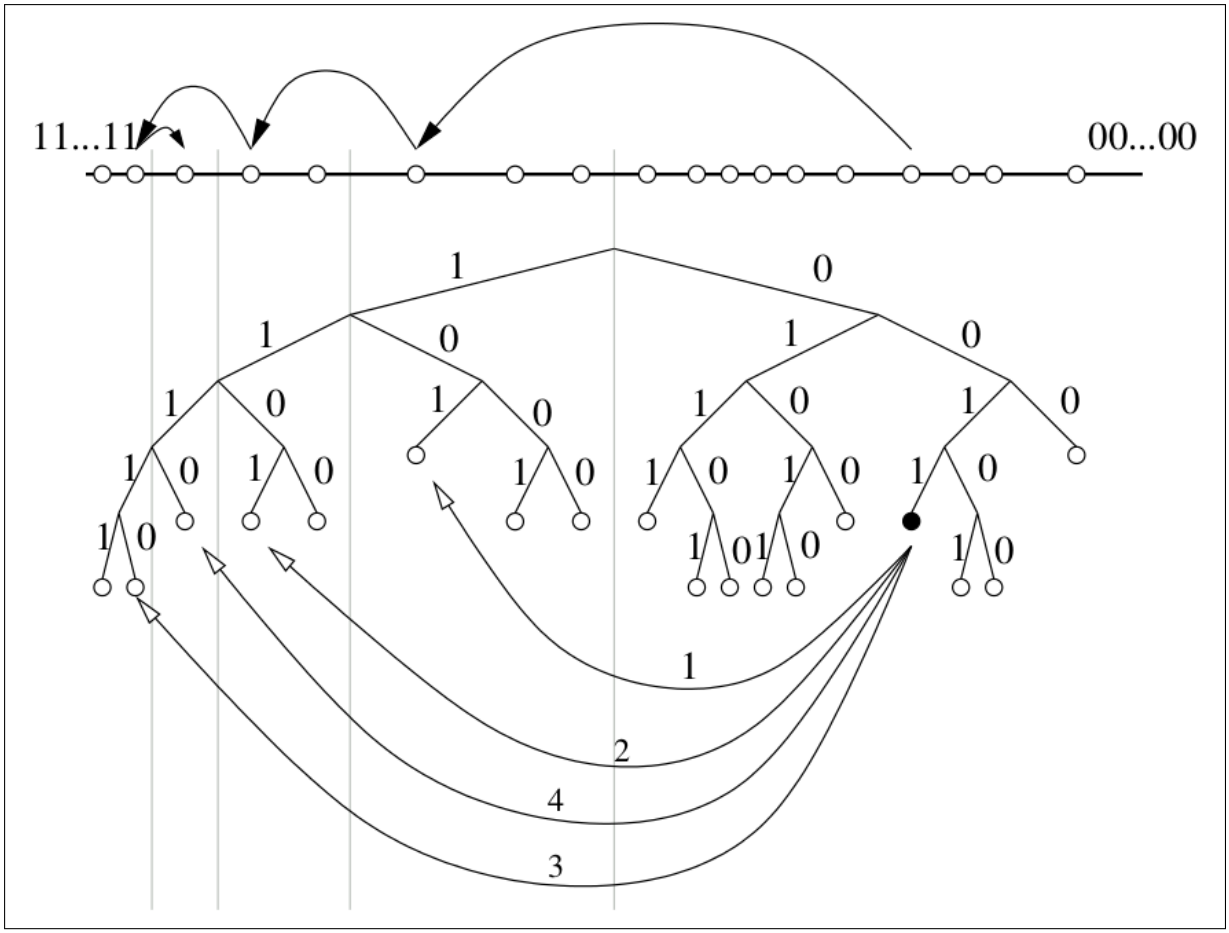


Figura 3.4: *Exemplo de uma busca na árvore de nós do Kademlia usando-se um ID. O nó preto, de prefixo 0011, encontra o nó de prefixo 1110 através de sucessivas buscas (setas numeradas inferiores). As setas superiores mostram a convergência da busca durante a execução. Fonte: [26]*

Para o conceito de proximidade, as distâncias são calculadas usando-se a função de distância

$$d(x, y) = x \oplus y \tag{3.1}$$

que possui algumas propriedades em comum com a equação de distância euclidiana usual:

- $d(x, x) = 0$
- $x \neq y$, $d(x, y) > 0$
- simetria: $\forall x, y$, $d(x, y) = d(y, x)$
- desigualdade triangular: $d(x, y) + d(y, z) \geq d(x, z)$.
  Isto vem do fato de $d(x, z) = d(x, y) \oplus d(y, z)$ e que $\forall a \geq 0, \forall b \geq 0 : a + b \geq a \oplus b$
- unidirecionalidade: para um dado ponto x e uma distância $\Delta > 0$, existe exatamente um ponto y tal que $d(x, y) = \Delta$. Isso garante que todas as procuras por uma mesma chave convirjam para um mesmo percurso, independente do ponto de partida.

Tabela de roteamento

Cada nó do Kademlia armazena informações sobre outros nós para rotear mensagens de pesquisa. Para cada bit i dos IDs (cada ID tem 160 bits) é mantido um k-balde (k-*bucket*), que contém os nós cuja distância até ele está entre 2^i e 2^{i+1}. Esses k-buckets são listas de endereço IP, porta de comunicação UDP e ID de nós, ordenadas pelo horário da última notícia destes. Para distâncias pequenas, essas listas geralmente serão vazias, enquanto para distâncias maiores poderão ser de tamanho k. Este valor, que é o de replicação do sistema, é escolhido de tal forma que esses k nós possuam grande probabilidade de não falharem na próxima hora.

Quando um nó **A** recebe uma mensagem de outro nó **B**, o balde (*bucket*) correspondente ao ID do remetente (nó **B**) é atualizado. Disto, podem ocorrer as seguintes situações:

- **B** já existe no bucket: passa a ser o primeiro da lista, pois existiu mensagem recente.
- **B** não existe no bucket:
  - bucket não está cheio: **B** é adicionado no começo da lista.
  - bucket cheio: é enviado um *ping* para o nó do final da lista (nó **C**), contatado há mais tempo
    - **C** não responde ao *ping*: **C** é retirado da lista e **B** é inserido no início
    - **C** responde ao *ping*: **C** é movido para o início da lista e **B** é ignorado

Por conta disso, ocorre que nós mais antigos e funcionais são preferidos, pois quanto mais tempo um nó está conectado, mais provável ele se manterá conectado por mais 1 hora [35]. Outra vantagem disso é a resistência a alguns ataques de negação de serviço, pois mesmo que ocorra uma inundação de novos nós, estes só seriam inseridos nos k-buckets se os antigos fossem excluídos.

Protocolo

O protocolo de mensagens DHT utiliza o formato KRPC, que é um mecanismo de chamada de procedimento remoto (RPC) que envia dicionários bencode através de UDP, uma única vez por chamada (um pacote para a requisição, outro para a resposta), sem novas tentativas.

Existem 3 tipos de mensagem: consulta (*query*), resposta (*response*) e erro (*error*). Para o protocolo DHT, são 4 comandos *query*: `ping`, `find_node`, `get_peers` e `announce_peer`. Em todos, o nó sempre enviará seu ID como valor da chave `id`.

Uma mensagem KRPC é um dicionário com 2 chaves comuns a todos os 4 comandos: `y`, que especifica o tipo da mensagem, e `t`, que corresponde ao ID da transação. Este é um número binário convertido para string, geralmente formada por 2 caracteres, possuindo valor até 2^{16}, e

devolvida nas respostas. Isso permite que estas se relacionem a múltiplas consultas a um nó. Esse ID é chamado de *magic cookie* (cookie mágico) [54].

Cada tipo de mensagem possui formatos diferentes entre si, permitindo parâmetros adicionais para cada chamada, possuindo as seguintes chaves e seus respectivos valores:

- *query*
  - `y`: caractere `q`
  - `q`: string do comando desejado (`ping`, `find_node`, `get_peers`, `announce_peer`)
  - `a`: dicionário contendo parâmetros adicionais, dependendo do comando passado na chave `q`
- *response*
  - `y`: caractere `r`
  - `r`: dicionário contendo valores da resposta, dependendo do comando passado na chave `q`
- *error*
  - `y`: caractere `e`
  - `e`: lista contendo 2 elementos: código (número inteiro) e mensagem para o erro (string). Os erros podem ser:
    - 201 (Generic Error): erros genéricos
    - 202 (Server Error): erros de servidor
    - 203 (Protocol Error): para pacote mal formado, argumento inválido ou token incorreto
    - 204 (Method Unknown): comando não conhecido
  - exemplo:
    `d1:eli201e23:A Generic Error Ocurrede1:t2:aa1:y1:ee` (bencode)
    `{"t":"aa", "y":"e", "e":[201, "A Generic Error Ocurred"]}` (string)

As informações retornadas podem ser sobre ou nós DHT: enquanto o primeiro é a "informação compacta de endereço IP/porta" - string de 6 bytes (4 bytes iniciais para o endereço IP e 2 bytes finais para a porta de comunicação usada) -, o segundo é a "informação compacta de nó" - string de 26 bytes (20 bytes iniciais para o ID do nó e 6 bytes finais para a respectiva informação compacta de endereço IP/porta).

Os 4 comandos de *query* do DHT (`ping`, `find_node`, `get_peers` e `announce_peer`) estão definidos da seguinte forma.

ping

É o comando mais simples, que verifica se o nó está online. Possui um único argumento, que é uma chave `id`, que é o ID do nó consultante (na requisição) ou do nó consultado (na resposta).

- formato da requisição
  `d1:ad2:id20:abcdefghij0123456789e1:q4:ping1:t2:aa1:y1:qe` (bencode)
  `{"t":"aa", "y":"q", "q":"ping", "a":{"id":"abcdefghij0123456789"}}` (string)
- formato da resposta
  `d1:rd2:id20:mnopqrstuvwxyz123456e1:t2:aa1:y1:re` (bencode)
  `{"t":"aa", "y":"r", "r":{"id":"mnopqrstuvwxyz123456"}}` (string)

./third-party/dht/dht.c:2291

```
/* We could use a proper bencoding printer and parser, but the format of DHT messages
is fairly stylised, so this seemed simpler. */
#define CHECK(offset, delta, size)                      \
    if(delta < 0 || offset + delta > size) goto fail

#define INC(offset, delta, size)                        \
    CHECK(offset, delta, size);                         \
    offset += delta

#define COPY(buf, offset, src, delta, size)             \
    CHECK(offset, delta, size);                         \
    memcpy(buf + offset, src, delta);                   \
    offset += delta;

#define ADD_V(buf, offset, size)                        \
    if(have_v) {                                        \
        COPY(buf, offset, my_v, sizeof(my_v), size);    \
    }
```

./third-party/dht/dht.c:2341

```
int send_ping(const struct sockaddr *sa, int salen, const unsigned char *tid, int tid_len) {
    int i = 0, rc; char buf[512]; // variáveis temporárias

    rc = snprintf(buf + i, 512 - i, "d1:ad2:id20:"); INC(i, rc, 512);
    COPY(buf, i, myid, 20, 512); // cola o ID do peer, que está em "myid"

    rc = snprintf(buf + i, 512 - i, "e1:q4:ping1:t%d:", tid_len); INC(i, rc, 512);
    COPY(buf, i, tid, tid_len, 512); // cola o ID do Transmission de 4 bytes para a transação
    ADD_V(buf, i, 512);

    rc = snprintf(buf + i, 512 - i, "1:y1:qe"); INC(i, rc, 512);
    return dht_send(buf, i, 0, sa, salen); // envia a requisição de ping
    (...)
}

int send_pong(const struct sockaddr *sa, int salen, const unsigned char *tid, int tid_len) {
    int i = 0, rc; char buf[512]; // variáveis temporárias

    rc = snprintf(buf + i, 512 - i, "d1:rd2:id20:"); INC(i, rc, 512);
    COPY(buf, i, myid, 20, 512); // cola o ID do peer, que está em "myid"

    rc = snprintf(buf + i, 512 - i, "e1:t%d:", tid_len); INC(i, rc, 512);
    COPY(buf, i, tid, tid_len, 512); // cola o ID do Transmission de 4 bytes para a transação
    ADD_V(buf, i, 512);

    rc = snprintf(buf + i, 512 - i, "1:y1:re"); INC(i, rc, 512);
    return dht_send(buf, i, 0, sa, salen); // envia a resposta a um ping
    (...)
}
```

find_node

Este comando, que equivale à mensagem de `FIND\_NODE` do artigo do Kademlia [26], é usado para encontrar as informações do nó dado seu ID. Necessita enviar 2 argumentos: a chave `id` e o ID do nó consultante, e a chave `target` e o ID do nó cujas informações o consultante está procurando (ou nó alvo).

- formato dos argumentos da requisição
  `{"id":\"<IDs dos nós consultantes>", "target":\"<ID do nó alvo>"}`
- formato da resposta
  `{"id":\"<IDs dos nós consultados>", "nodes":\"<info compacta do(s) nó(s)>"}`

O nó consultado deve responder com a chave `nodes` contendo uma string com a informação compacta (6 bytes) do nó alvo ou dos k nós bons (que fizeram contato recentemente) mais próximos que estão contidos em sua tabela de roteamento, de 1 ou mais k-buckets. O funcionamento do algoritmo da busca é explicado no trabalho:

> O procedimento mais importante que um participante do Kademlia deve realizar é encontrar os k nós próximos a um dado ID de nó. Nós chamamos esse procedimento de "*lookup* de nós". Kademlia utiliza de um algoritmo recursivo nas buscas por nós. O disparador das buscas começa escolhendo α nós do bucket não- vazio mais próximo (ou, se esse bucket tiver menos que α entradas, utiliza desses α nós mais próximos que conhece). Então, o disparador envia chamadas RPC assíncronas paralelas de comandos **find_node** para esses α nós escolhidos. α é um parâmetro de concorrência geral ao sistema, assumindo valor como 3.
>
> No passo recursivo, o disparador reenvia chamadas a **find_node** para os nós que conheceu das chamadas RPC passadas. (Esta recursão pode começar antes que todos os α nós anteriores tenham respondido). Dos k nós que o disparador concluiu serem mais próximos ao alvo, ele pega α que ainda não foram consultados e envia chamadas RPC **find_node**. Nós que falharem em responder rapidamente são desconsiderados até que respondam. Se uma rodada de comandos **find_node** não retornar algum nó mais próximo do que os nós já conhecidos, o disparador reenvia comandos **find_node** para todos os k nós mais próximos que ainda não foram consultados. O *lookup* termina quando o disparador tiver consultado e obtido respostas de todos os k nós mais próximos conhecidos.

Porém, o Transmission implementa essa busca de forma mais flexível e simples. De início, busca o bucket no qual o ID procurado está ou que contém nós mais próximos.

./third-party/dht/dht.c:2536

```c
int send_closest_nodes(const struct sockaddr *sa, int salen, const unsigned char *tid,
    int tid_len, const unsigned char *id, int want, int af, struct storage *st,
    const unsigned char *token, int token_len) {
    unsigned char nodes[8 * 26]; unsigned char nodes6[8 * 38];  // variáveis...
    int numnodes = 0, numnodes6 = 0; struct bucket *b;          // ... temporárias

    (...)
    b = find_bucket(id, AF_INET); // Busca o bucket provável.
    if (b) {
        // Procura por nós no bucket encontrado e nos vizinhos anterior e/ou posterior,
        // se possuir.
        numnodes = buffer_closest_nodes(nodes, numnodes, id, b);
        if (b->next) numnodes = buffer_closest_nodes(nodes, numnodes, id, b->next);
        b = previous_bucket(b);
        if (b) numnodes = buffer_closest_nodes(nodes, numnodes, id, b);
    }
    (...)

    // Envia os nós encontrados.
    return send_nodes_peers(sa, salen, tid, tid_len, nodes, numnodes * 26, nodes6,
        numnodes6 * 38, af, st, token, token_len);
}
```

A busca do bucket itera sobre a lista ligada de buckets.

./third-party/dht/dht.c:129

```c
struct node {
    // elemento de lista ligada de nós DHT
    (...)
    unsigned char id[20]; // ID do nó
    time_t time;          // horário da última mensagem recebida
    time_t reply_time;    // horário da última resposta recebida corretamente
    time_t pinged_time;   // horário da última requisição
    int pinged;           // quantidade de requisições feitas desde a última resposta
    struct node *next;    // ponteiro para o próximo elemento da lista ligada
};

struct bucket {
    // elemento de lista ligada de baldes
    (...)
    unsigned char first[20];        // ID do primeiro nó do bucket
    int count;                      // quantidade de nós no bucket
    int time;                       // horário da última resposta neste bucket
    struct sockaddr_storage cached; // endereços de possíveis candidatos
    int cachedlen;                  // tamanho da lista de candidatos
    struct node *nodes;             // ponteiro para lista ligada de nós
    struct bucket *next;            // ponteiro para o próximo elemento da lista ligada
};
```

./third-party/dht/dht.c:464

```c
static int id_cmp(const unsigned char *restrict id1, const unsigned char *restrict id2) {
    /* Memcmp is guaranteed to perform an unsigned comparison. */
    return memcmp(id1, id2, 20);
}

static struct bucket * find_bucket(unsigned const char *id, int af) {
    struct bucket *b = af == AF_INET ? buckets : buckets6; // seletor de buckets IPv4/IPv6

    if (b == NULL) return NULL;
    while (1) {
        if (b->next == NULL) return b;
        if (id_cmp(id, b->next->first) < 0) // id tem valor binário menor que b->next->first,
            return b;                       // logo deve ficar neste bucket
        b = b->next;
    }
}
```

Caso retorne o bucket mais provável, efetua buscas internas nele. Se ele possuir elementos vizinhos anteriores ou posteriores, também busca por nós neles.

./third-party/dht/dht.c:2523

```c
static int buffer_closest_nodes(unsigned char *nodes, int numnodes, const unsigned char *id,
    struct bucket *b) {
    struct node *n = b->nodes;
    while (n) {              // Itera sobre os nós...
        if (node_good(n))    // ... e, caso ele tenha feito contato recentemente, ...
            numnodes = insert_closest_node(nodes, numnodes, id, n); // ...o usa.
        n = n->next;
    }
    return numnodes; // Retorna a quantidade de nós encontrados.
}
```

./third-party/dht/dht.c:2476

```c
static int insert_closest_node(unsigned char *nodes, int numnodes, const unsigned char *id,
    struct node *n) {
    int i, size; // variáveis temporárias

    (...)

    // Itera sobre os k (neste caso, 8) nós mais próximos atuais (nodes) e verifica
    // se algum é o nó procurado ou se é mais próximo que algum nó conhecido (n).
    for (i = 0; i < numnodes; i++) {
        if (id_cmp(n->id, nodes + size * i) == 0) return numnodes;
        if (xorcmp(n->id, nodes + size * i, id) < 0) break;
    }

    // Se tiver iterado por todos os nós próximos atuais, eles são os mais próximos; ...
    if (i == 8) return numnodes;

    if (numnodes < 8) numnodes++; // ... caso contrário, o nó atual (n) entra para a lista.

    if (i < numnodes - 1) // Abre espaço para o nó atual (n) na lista.
        memmove(nodes + size * (i + 1), nodes + size * i, size * (numnodes - i - 1));

    (...)
    // Adiciona o nó atual (n) para a lista, convertendo os dados de endereço IP e porta.
    struct sockaddr_in *sin = (struct sockaddr_in*) &n->ss;
    memcpy(nodes + size * i, n->id, 20);
    memcpy(nodes + size * i + 20, &sin->sin_addr, 4);
    memcpy(nodes + size * i + 24, &sin->sin_port, 2);
    (...)

    return numnodes;
}
```

Ao fim da busca, envia a lista de nós que encontrou como resposta ao comando de `find_node` recebida. O Transmission também utiliza a função para enviar os nós encontrados pelo comando `get_peers` (pág. 36).

./third-party/dht/dht.c:2409

```c
int send_nodes_peers(const struct sockaddr *sa, int salen, const unsigned char *tid,
    int tid_len, const unsigned char *nodes, int nodes_len, const unsigned char *nodes6,
    int nodes6_len, int af, struct storage *st, const unsigned char *token, int token_len) {
    char buf[2048]; int i = 0, rc, j0, j, k, len; // variáveis temporárias

    rc = snprintf(buf + i, 2048 - i, "d1:rd2:id20:");
    INC(i, rc, 2048);
    COPY(buf, i, myid, 20, 2048);
    if (nodes_len > 0) {
        // Monta a string bencode dos nós a serem enviados.
        rc = snprintf(buf + i, 2048 - i, "5:nodes%d:", nodes_len);
        INC(i, rc, 2048);
        COPY(buf, i, nodes, nodes_len, 2048);
    }

    (...)

    // find_node: token_len = 0
    // get_peers: token_len > 0
    if (token_len > 0) {
        // Adiciona o token na resposta da chamada.
        rc = snprintf(buf + i, 2048 - i, "5:token%d:", token_len);
        INC(i, rc, 2048);
        COPY(buf, i, token, token_len, 2048);
    }

    // find_node: st = NULL
    // get_peers: st != NULL
    if (st && st->numpeers > 0) {
        /* We treat the storage as a circular list, and serve a randomly
         chosen slice.  In order to make sure we fit within 1024 octets,
         we limit ourselves to 50 peers. */

        len = af == AF_INET ? 4 : 16;
        j0 = random() % st->numpeers;
        j = j0;
        k = 0;

        rc = snprintf(buf + i, 2048 - i, "6:valuesl");
        INC(i, rc, 2048);
        do {
            if (st->peers[j].len == len) {
                unsigned short swapped;
                swapped = htons(st->peers[j].port);
                rc = snprintf(buf + i, 2048 - i, "%d:", len + 2);
                INC(i, rc, 2048);
                COPY(buf, i, st->peers[j].ip, len, 2048);
                COPY(buf, i, &swapped, 2, 2048);
                k++;
            }
            j = (j + 1) % st->numpeers;
        } while (j != j0 && k < 50);
        rc = snprintf(buf + i, 2048 - i, "e");
        INC(i, rc, 2048);
    }

    rc = snprintf(buf + i, 2048 - i, "e1:t%d:", tid_len);
    INC(i, rc, 2048);
    COPY(buf, i, tid, tid_len, 2048);
    ADD_V(buf, i, 2048);
    rc = snprintf(buf + i, 2048 - i, "1:y1:re");
    INC(i, rc, 2048);

    return dht_send(buf, i, 0, sa, salen);
    (...)
}
```

Um exemplo de requisição e resposta para este comando é

- exemplo de requisição
  `d1:ad2:id20:abcdefghij01234567896:target20:mnopqrstuvwxyz123456e1:q 9:find_node1:t2:aa1:y1:qe` (bencode)
  `{"t":"aa", "y":"q", "q":"find_node", "a":{"id":"abcdefghij0123456789", "target":"mnopqrstuvwxyz123456"}}` (string)

- exemplo de resposta
  `d1:rd2:id20:0123456789abcdefghij5:nodes9:def456...e1:t2:aa1:y1:re` (bencode)
  `{"t":"aa", "y":"r", "r":{"id":"0123456789abcdefghij", "nodes": "def456..."}}` (string)

get_peers

É o comando RPC da mensagem `FIND_VALUE`, serve para buscar peers para um dado um valor hash identificador de arquivo .torrent, enviado como valor da chave `info_hash`, além do ID do nó consultante como valor da chave `id`.

O funcionamento é equivalente ao comando `find_node`, com um detalhe extra: se o nó que recebeu a mensagem possuir peers para o valor hash dado, eles são informados imediatamente na forma compacta (6 bytes para cada peer) numa lista bencode de strings, devolvida como valor da chave `values`. Por outro lado, caso o receptor da mensagem não conhecer nós para o valor hash especificado, a resposta conterá a chave `nodes` com os k nós mais próximos desse valor hash. O nó original consulta outros nós próximos ao arquivo .torrent iterativamente. Ao fim da busca, o programa cliente insere o contato para si mesmo na lista de nós próximos ao arquivo .torrent.

Em ambos os casos, uma chave `token` é informada na resposta, cujo valor é uma string binária curta, que deverá ser utilizada em futuras mensagens de `announce_peer`.

referencio o código do comando anterior?

- formato dos argumentos da requisição
  `{"id":\"<IDs dos nós consultantes>",`
  `"info_hash":\"<hash de 20 bytes do torrent buscado>"}`

- formato da resposta
  `{"id":\"<IDs dos nós consultados>", "token":\"<token>",`
  `"values":[\"<info peer 1>", \"<info peer 2>", ...]}`
  ou
  `{"id":\"<IDs dos nós consultados>", "token":\"<token>",`
  `"nodes":\"<info compacta do(s) nó(s)>"}`

./third-party/dht/dht.c:1866

```c
int dht_periodic(const void *buf, size_t buflen, const struct sockaddr *from, int fromlen,
    time_t *tosleep, dht_callback *callback, void *closure) {
    (...)
    // variáveis temporárias
    int message;
    unsigned char tid[16], id[20], info_hash[20], target[20];
    unsigned char nodes[256], nodes6[1024], token[128];
    int tid_len = 16, token_len = 128;
    int nodes_len = 256, nodes6_len = 1024;
    unsigned short port;
    unsigned char values[2048], values6[2048];
    int values_len = 2048, values6_len = 2048;
    int want;

    (...)
    // Processa a mensagem recebida e identifica seu tipo.
    message = parse_message(buf, buflen, tid, &tid_len, id, info_hash, target,
            &port, token, &token_len, nodes, &nodes_len, nodes6, &nodes6_len,
            values, &values_len, values6, &values6_len, &want);
    (...)

    // Realiza os procedimentos conforme o tipo da mensagem recebida.
    switch (message) {
        (...)
        case FIND_NODE:
            // find_node: só envia os nós mais próximos ao 'target'.
            debugf("Find node!\n");
            new_node(id, from, fromlen, 1);
            debugf("Sending closest nodes (%d).\n", want);
            send_closest_nodes(from, fromlen, tid, tid_len, target, want, 0, NULL, NULL, 0);
            break;
        case GET_PEERS:
            debugf("Get_peers!\n");
            new_node(id, from, fromlen, 1);
            if (id_cmp(info_hash, zeroes) == 0) {
                (...)
            }
            else {
                struct storage *st = find_storage(info_hash);
                unsigned char token[TOKEN_SIZE];
                make_token(from, 0, token);
                // Se conhecer nós para o hash de torrent dado, os indica na resposta...
                if (st && st->numpeers > 0) {
                    debugf("Sending found%s peers.\n",
                            from->sa_family == AF_INET6 ? " IPv6" : "");
                    send_closest_nodes(from, fromlen, tid, tid_len, info_hash, want,
                            from->sa_family, st, token, TOKEN_SIZE);
                }
                else {
                    // ..., senão procura nós próximos, como no comando find_node.
                    debugf("Sending nodes for get_peers.\n");
                    send_closest_nodes(from, fromlen, tid, tid_len, info_hash,
                        want, 0, NULL, token, TOKEN_SIZE);
                }
            }
            break;
        (...)
    }
    (...)
}
```

- exemplo de requisição
  `d1:ad2:id20:abcdefghij01234567899:info_hash20:mnopqrstuvwxyz123456e`
  `1:q9:get_peers1:t2:aa1:y1:qe` (bencode)
  ```
  {"t":"aa", "y":"q", "q":"get_peers",
   "a":{"id":"abcdefghij0123456789", "info_hash":"mnopqrstuvwxyz123456"}}
  ```
  (string)

- exemplo de resposta

  – com peers
  `d1:rd2:id20:abcdefghij01234567895:token8:aoeusnth`
  `6:valuesl6:axje.u6:idhtnmee1:t2:aa1:y1:re` (bencode)
  ```
  {"t":"aa", "y":"r", "r":{"id":"abcdefghij0123456789",
   "token":"aoeusnth", "values":["axje.u", "idhtnm"]}}
  ```
  (string)

  – com nós próximos
  `d1:rd2:id20:abcdefghij01234567895:nodes9:def456...5:token`
  `8:aoeusnthe1:t2:aa1:y1:re` (bencode)
  ```
  {"t":"aa", "y":"r", "r":{"id":"abcdefghij0123456789",
   "token":"aoeusnth", "nodes":"def456..."}}
  ```
  (string)

announce_peer

O último comando é o da mensagem `STORE`, pelo qual um nó avisa outros que está baixando um arquivo .torrent, passando 4 argumentos: o ID do nó consultante como valor da chave `id`, o valor hash identificador do arquivo .torrent na chave `info_hash`, `port` contém um número inteiro de porta, e o `token` recebido como resposta de uma mensagem `get_peers` anterior. O nó consultado deve verificar que esse token foi enviado anteriormente para o mesmo endereço IP que o nó consultante, e então o nó consultado armazena usando o `info_hash` do torrent como chave e a informação compacta de endereço IP/porta do nó como valor.

Existe ainda mais um argumento, opcional, que é o `implied_port`, cujo valor pode ser 0 ou 1. Se este for 1, o argumento da porta deve ser ignorado e então a fonte de pacotes UDP deve ser usada como a porta do peer. Isso é útil para peers que estão em redes internas a gateway NAT, que podem não saber quais são suas portas externas, e que suportam uTP, aceitando conexões na mesma porta que o DHT.

O token tem papel fundamental para a segurança neste comando, pois serve para prevenir que um peer malicioso registre outros peers para um . No BitTorrent, esse token é a string do valor hash SHA-1 do endereço IP concatenado a uma chave secreta, criada pelo programa cliente, que varia a cada 5 minutos e são aceitos até 10 minutos depois de serem criados.

- formato dos argumentos da requisição

```
{\"id": \"<IDs dos nós consultantes>",
  \"implied_port": <0 ou 1>,
  \"info_hash": \"<hash de 20 bytes do torrent>",
  \"port": <número da porta>,
  \"token": \"<token>"}
```

- formato da resposta

```
{"id": \"<IDs dos nós consultados>"}
```

./third-party/dht/dht.c:1277

```c
static int storage_store(const unsigned char *id, const struct sockaddr *sa,
    unsigned short port) {
    int i, len; struct storage *st; unsigned char *ip;

    (...)
    st = find_storage(id); // Encontra o banco para o ID (hash do torrent) dado.

    if (st == NULL) {    // Se não existe banco, cria um para o torrent e adiciona na...
        (...)            // ... lista de bancos.
        st = calloc(1, sizeof(struct storage));
        (...)
        memcpy(st->id, id, 20);
        st->next = storage;
        storage = st;
        numstorage++;
    }

    // Procura pelo peer passado.
    for (i = 0; i < st->numpeers; i++) {
        if (st->peers[i].port == port &&
            st->peers[i].len == len &&
            memcmp(st->peers[i].ip, ip, len) == 0) break;
    }

    // Se o peer já existir, só atualiza o momento da última notificação.
    if (i < st->numpeers) {
        /* Already there, only need to refresh */
        st->peers[i].time = now.tv_sec;
        return 0;
    }
    else {
        struct peer *p;
        // Se não tiver espaço para o peer novo, expande o banco.
        if (i >= st->maxpeers) {
            /* Need to expand the array. */
            struct peer *new_peers; int n;

            // Não tem mais espaço.
            if (st->maxpeers >= DHT_MAX_PEERS) return 0;

            // Expande o banco criando 2 espaços ou dobrando de tamanho, limitando em...
            // ...DHT_MAX_PEERS espaços;
            n = st->maxpeers == 0 ? 2 : 2 * st->maxpeers;
            n = MIN(n, DHT_MAX_PEERS);
            new_peers = realloc(st->peers, n * sizeof(struct peer));
            if (new_peers == NULL) return -1;
            st->peers = new_peers; st->maxpeers = n;
        }
        // Adiciona o novo peer no banco.
        p = &st->peers[st->numpeers++];
        p->time = now.tv_sec; p->len = len;
        p->port = port; memcpy(p->ip, ip, len);
        return 1;
    }
}
```

- exemplo de requisição
  `d1:ad2:id20:abcdefghij01234567899:info_hash20:mnopqrstuvwxyz123456`
  `4:porti6881e5:token8:aoeusnthe1:q13:announce_peer1:t2:aa1:y1:qe`
  (bencode)
  `{"t":"aa", \"y":"q", \"q":"announce_peer",`
  `"a":{"id":"abcdefghij0123456789", \"implied_port": 1,`
  `"info_hash":"mnopqrstuvwxyz123456", \"port": 6881, \"token": \"aoeusnth"}}`
  (string)

- exemplo de resposta
  `d1:rd2:id20:mnopqrstuvwxyz123456e1:t2:aa1:y1:re`
  (bencode)
  `{"t":"aa", \"y":"r", \"r": {"id":"mnopqrstuvwxyz123456"}}` (string)

Funcionamento

Localizando nós

Localizando recursos

Entrando na rede

Saindo da rede

3.3 Peer Exchange

Funciona somente no swarm!

Explicar a funcionalidade de PEX

3.4 Jogo da troca de arquivos

Explicarei o algoritmo tit-for-tat padrão do protocolo BitTorrent, que vem da Teoria dos Jogos, e como o Transmission o implementa.

Capítulo 4

Conceitos de Computação no BitTorrent

Fazer pequena introdução aqui.

Aqui mostrarei detalhes técnicos sobre as partes coadjuvantes do BitTorrent e do Transmission.

4.1 Estruturas de dados, listas ligadas e árvores

Aqui vou falar de tipos de estruturas de dados utilizadas no programa, como *structs* e a sua utilização na implementação de listas ligadas e árvores e em como estes são usados na implementação de filas.

4.2 Funções de hash

Aqui vou explicar como funciona o algoritmo da função de hash SHA-1 e mostrar como e para que é usado na identificação de torrents e na verificação de integridade de partes.

4.3 Criptografia

Aqui vou explicar como este algoritmo de chave simétrica funciona e como é utilizado pelo Transmission para criptografar pacotes de dados.

4.4 Bitfields

Apesar de ser um simples array de bits usado no gerenciamento de partes que o programa já baixou ou não, foi percebido que o seu uso de forma não-convencional, chamado de *lazy bitfield*, pode ajudar a evitar o controle de banda (chamado de modelagem de tráfego, ou *traffic shaping* em inglês), feito por ISPs.

4.5 Protocolos de redes

Aqui vou explicar o que são os protocolos de rede TCP e UDP, apontar suas diferenças e mostrar os motivos pelos quais o UDP é preferido ao TCP no uso de endereços de announce de trackers.

4.6 Multicast

Apesar de não ser utilizado pelo protocolo BitTorrent, o multicast, que é uma forma de entregar dados a um grupo de computadores simultaneamente numa só transmissão, é usado pelo Transmission para tentar descobrir peers que estão na mesma rede local, otimizando as conexões.

4.7 Roteamento de pacotes

Em alguns roteadores que são ponte entre a Internet e a rede que ele gerencia, existe uma função de se configurar portas de comunicação de rede automaticamente usando-se o *Network Address Translation Port Mapping Protocol*. Assim, não é necessário realizar uma configuração específica somente pare esse fim.

4.8 Retomada de downloads

No Transmission e em algums outros softwares que realizam downloads, existe a função de se pausar a transferência do arquivo para que seja retomado em outro momento. Nesta seção, mostrarei qual a idéiapor trás desse mecanismo e a forma como foi implementado no Transmission.

4.9 Conexão com a Internet

Aqui mostrarei a parte técnica da programação em linguagem C para utilização de transmissão de dados por rede.

4.10 IPv6

O IPv6 é a versão mais recente do protocolo de Internet (IP), que foi criado para substituir o IPv4, que atualmente é mais o usado porém sofre de exaustão de endereços. Nesta seção, falarei sobre o novo protocolo e quais as implicações na programação de softwares com comunicação de redes.

4.11 Threads

Dentro de um contexto de programas que utilizam a Internet e funcionalidades que chegam próximas ao tempo real, processamentos pesados devem ser tratados com cautela a fim de se manter a instantaneidade do processo. Aqui explicarei como o Transmission usa o conceito de *threads* para paralelizar esses processamentos e, com isso, conseguir utilizar as informações de rápida mudança antes que seja necessário obtê-las novamente.

4.12 Engenharia de Software

O Transmission é um programa extenso e complexo, desenvolvido por vários programadores que estão espalhados pelo globo. Nesta seção, abordarei alguns pontos utilizados pelos desenvolvedores na manutenção do código aberto de qualidade e em funcionamento.

Capítulo 5

Comentários Finais

Capítulo 6

Glossary

announce
endereço URL do tracker para troca de informações, onde este recebe de peers informações sobre as respectivas situações com relação a um torrent específico naquele momento, as processa e então responde informando sobre a situação geral daquele torrent e uma lista de outros peers conectados naquele momento. 15--22, 42, 48

anycast
método de endereçamento e roteamento de rede onde os datagramas de um único remetente são roteados para um membro de um grupo de receptores potenciais que estão definidos pelo mesmo intervalo no endereço de destino. Geralmente é usado para serviços que demandem alta disponibilidade. 4

arquivo .torrent
arquivo que contém metadados, como a lista dos nomes dos arquivos a serem baixados e seus tamanhos, checksums das partes desses arquivos, endereços de um ou mais trackers, etc, formando um pacote chamado torrent. 8, 10, 11, 13--16, 19, 27, 36, 38, 49

Audiogalaxy
Rede P2P de compartilhamento de músicas MP3 criado em 1998. 2, 3

bencode
‘‘codificação B’’, pronunciado *bê encode*; formato de codificação compacta de arquivos torrent para transmissão de metadados. 11, 16--19, 29--31, 36, 38, 40

beta tester
usuários de uma versão beta de um software. 6

bucket
. 29, 32--34 escrever

case insensitive
em português, insensível ao tamanho das letras (maiúsculas ou minúsculas); uma string case insensitive não é diferenciada com letras iguais em caixas alta ou baixa. Assim, `FOO`, `Foo` e `foo` seriam strings dadas como iguais. 49

checksum
em português, soma de verificação; bloco de dados de tamanho fixo gerado por algum algoritmo de soma para verificação, usado no certificado de integridade contra problemas durante a transmissão ou de armazenamento (leitura ou escrita). 45

cookie
. 30 escrever

DHT
do inglês *distributed hash table*; tabela hash distribuída, ou seja, é um serviço de busca similar a uma tabela hash, mas descentralizada e na forma de sistema distribuído. 5, 25, 26, 29, 30, 38, 47

eDonkey
lançado em 6 de setembro de 2000, o protocolo foi inaugurado juntamente com o software que o utilizava, o eDonkey2000, mas inúmeros softwares cliente para diferentes plataformas surgiram nos dias seguintes ao lançamento. 4, 5

função de hash
é uma função ou algoritmo matemático que mapeia um dado de comprimento variável em outro de comprimento fixo. 13, 16, 41, 49

gateway NAT
do inglês *Network Address Translation*; tradução de endereço de rede, é uma funcionalidade empregada por roteadores de rede, que por associarem um endereço de Internet (o fornecido pelo ISP) para vários dispositivos dentro dessa rede, precisa manter uma tabela de tradução de endereços para que saiba rotear os pacotes de dados transmitidos entre as redes interna e externa. 17, 38

Gnutella
software de compartilhamento P2P desenvolvido por 3 programadores da empresa Nullsoft, recém adquirida da AOL Inc., lançado em 2000 sob a licença GPL. No dia seguinte ao lançamento, a AOL ordenou indisponibilizar o software, alegando problemas legais e proibindo a continuação do desenvolvimento. Alguns dias depois, o protocolo já tinha sido alvo de engenharia reversa e já havia softwares que o implementavam. 4, 5, 7, 26

HTTP GET
método de requisição do protocolo HTTP que permite uso de query strings para troca de informações. 15

ISP
do inglês *Internet Service Provider*; fornecedores de acesso à Internet, que são empresas que vendem serviço e equipamento que permitem o acesso de um computador pessoal à Internet. 2, 42, 46

Kademlia
DHT usado em P2P que especifica a estrutura da rede e a troca de informações através de buscas de nós, guardando as localizações de recursos que estão na rede. 5

k-bucket
. 29, 32 escrever

leecher
em português, sugador; nome dado ao peer que ainda não terminou um download de um torrent. 8, 18, 19

magnet link
em português, link magnético; padrão aberto, definido por convenção, de esquema de URI utilizado para localizar recursos de rede BitTorrent para download. 13, 14

metadado
dados sobre outros dados; informação sobre outra informação. 8, 13, 45

MP3
do inglês *MPEG-1/2 Audio Layer 3*; formato patenteado de compressão de dados de áudio digital, que usa um método de compressão de dados com perdas. 2, 3, 45

P2P
do inglês *peer-to-peer*; redes de arquitetura descentralizada e distribuída, onde cada nó (peer) fornece e consome recursos. 2--4, 6--8, 25, 45--47

peer
em português, significa par, colega; nome que se dá a cada nó da rede, ou seja, a um computador conectado. 4, 5, 7, 8, 10, 13, 15--19, 22--27, 36, 38, 42, 45, 47--49

pool
em português, poço de recursos; conjunto de recursos alocados em memória que são pré-computados, a fim de estarem prontos para serem utilizados. 23

proxy
servidor que funciona como intermediário de acessos, repassando requisições de um computador cliente para o servidor de destino. 17

query string
em português, string de busca; parte de uma URL que possui um dicionário de dados a serem transmitidos para alguma aplicação de Internet, determinado por um caractere `?`. Por exemplo, em `http://um/endereco/qualquer/?key1=value1&key2=value2`. 13, 46

RIAA
do inglês *Recording Industry Association of America*; Associação da Indústria de Gravação da América, organização que representa as gravadoras musicais e distribuidores, e tem sido autora de ações judiciais devido a quebra de direitos autorais causada por compartilhamento indevido de música. 3

RPC
. 29, 32, 36 escrever

scrape
do inglês *screen scraping*; sondagem de tela, que era um processo automatizado para recolhimento de informações sobre torrents que acessavam páginas web dos trackers. Por questões práticas e econômicas, foi estabelecido que neste endereço URL do tracker, ao contrário do announce, o tracker somente informa as quantidades de peers que estão participando de uma lista de torrents naquele momento. 19, 20, 22

seeder
em português, semeador; nome dado ao peer que já terminou um download de um torrent e que, por ainda estar conectado à rede, fornece partes a possíveis interessados. 8, 16, 18, 19

string
sequência de caracteres. 11--13, 16--18, 29--32, 36, 38, 40, 46, 47, 49

swarm
em português, enxame; grupo de peers que estão compartilhando dados de um mesmo torrent num determinado momento. 8--10, 13, 15, 19, 23, 40, V

swarming
também chamado de transmissão de arquivos por segmentação ou de múltiplas fontes, é a transmissão em paralelo de um arquivo, a partir de um ou vários locais onde estiver disponível, para um único destino. Cabe ao software do destinatário juntar as partes recebidas. 5

tabela hash
ou *mapa de hash*, é uma estrutura de dados que cria uma lista de correspondência chave-valor, onde os dados são guardados como valores e indexados por seus respectivos *valores hash*. 5, 26, 46

TCP
do inglês *Transmission Control Protocol*; protocolo de controle de transmissão, é um dos protocolos principais de Internet (IP). Pertencente à camada de transporte de dados de rede, provê conexões cujos pacotes de dados são pesados, mas com entregas confiáveis, ordenadas e com verificação de erros e de congestionamento. 16, 19

thread
. 22, 23 escrever

torrent
: conjunto de um ou mais arquivos definidos por um arquivo .torrent. 8--10, 13--16, 19, 24, 26, 27, 45

tracker
: em português, rastreador; servidor que funciona como um ponto de encontro de peer. 6, 8, 13, 15--19, 21--23, 42, 45, 48

UDP
: do inglês *Transmission Control Protocol*; protocolo de controle de transmissão, é um dos protocolos principais de Internet (IP). Pertencente à camada de transporte de dados de rede, provê conexões cujos pacotes de dados são leves, mas com entregas não confiáveis, desordenadas e sem verificação de erros (feita na camada de aplicação) e de congestionamento. 16, 19, 29, 38

URI
: do inglês *Uniform Resource Identifier*; Identificador Uniforme de Recursos, é uma string usada para identificar algum recurso, especificando algum protocolo e um caminho. Por exemplo, o URI `file:///arquivo.txt` indica um arquivo computador local (nome de esquema `file`), enquanto `http://pagina.com` se refere a uma página de Internet (nome de esquema `http`). 13, 47, 49

URL
: do inglês *Uniform Resource Locator*; Localizador Uniforme de Recursos, é uma string usada para identificar algum recurso na Internet, especificando algum protocolo de comunicação. 45, 47--49

URL encode
: em português, codificação de URL; por possuir caracteres especiais, URLs devem converter esses caracteres quando se desejar transmití-los. Letras (A-Z e a-z), números (0-9), caracteres (`.~_-`); espaços são convertidos para `+` ou `%20`; o restante é convertido para o respectivo valor em hexadecimal do caractere convertido para UTF-8. 13, 16

URN
: do inglês *Uniform Resource Name*; Nome Uniforme de Recursos, é o nome histórico dado a uma URI que usa o esquema `urn`. Sua sintaxe é `urn:<NID>:<NSS>`, onde `urn` é o prefixo case insensitive, `<NID>` é o identificador de espaço de nomes, que determina a interpretação sintática de `<NSS>`, que é a string específica do espaço de nomes usado. 13

valor hash
: ou *hash*; valores gerados por uma função de hash. 5, 13--16, 19, 27, 36, 38

Capítulo 7

Bibliografia

[1] *Amazon Simple Storage Service (Amazon S3) --- Amazon S3 Functionality*. [Online; acessado em 7-outubro-2013]. URL: http://aws.amazon.com/s3/#functionality.

[2] CCP Aporia. *All quiet on the EVE Launcher front?* [Online; acessado em 5-outubro-2013]. 11 de mar. de 2013. URL: http://community.eveonline.com/news/dev-blogs/74573.

[3] Kennon Ballou. *R.I.P. Audiogalaxy*. [Online; acessado em 30-setembro-2013]. 21 de jun. de 2002. URL: http://www.kuro5hin.org/story/2002/6/21/171321/675.

[4] Kenneth P. Birman. *Reliable Distributed Systems: Technologies, Web Services, and Applications*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005, pp. 532–534. ISBN: 0387215093.

[5] *BitTorrent*. URL: http://www.bittorrent.com/.

[6] *Blizzard Downloader --- Wowpedia*. [Online; acessado em 5-outubro-2013]. 2013. URL: http://wowpedia.org/index.php?title=Blizzard_Downloader&oldid=3198520.

[7] *CBC to BitTorrent Canada's Next Great Prime Minister*. URL: http://archive.is/VYmFD.

[8] *Codecon 2002 --- Schedule*. 2002. URL: http://web.archive.org/web/20021012072819/http://codecon.org/2002/program.html#bittorrent.

[9] M. Denters. *Download California Dreaming*. [Online; acessado em 7-outubro-2013]. 8 de nov. de 2010. URL: http://tegenlicht.vpro.nl/nieuws/2010/november/creative-commons.html.

[10] *DGM Live --- FAQ*. [Online; acessado em 7-outubro-2013]. URL: http://www.dgmlive.com/help.htm#whatisbittorrent.

[11] *Dr. Dre Raps Napster*. [Online; acessado em 30-setembro-2013]. 18 de abr. de 2000. URL: http://www.wired.com/techbiz/media/news/2000/04/35749.

[12] Ernesto. *BitTorrent And MPAA join forces*. 23 de nov. de 2005. URL: http://torrentfreak.com/BitTorrent-and-mpaa-join-forces/.

[13] Ernesto. *BitTorrent Makes Twitter's Server Deployment 75x Faster*. 16 de jul. de 2013. URL: http://torrentfreak.com/bittorrent-makes-twitters-server-deployment-75-faster-100716/.

[14] Ernesto. *Twitter Uses BitTorrent For Server Deployment*. 10 de fev. de 2013. URL: http://torrentfreak.com/twitter-uses-bittorrent-for-server-deployment-100210/.

[15] Ernesto. *UK Government Uses BitTorrent to Share Public Spending Data*. 4 de jun. de 2010. URL: http://torrentfreak.com/uk-government-uses-bittorrent-to-share-public-spending-data-100604/.

[16] Roy T. Fielding et al. *RFC 2616 --- Hypertext Transfer Protocol -- HTTP/1.1*. Jun. de 1999. URL: http://tools.ietf.org/html/rfc2616.

[17] Elle Cayabyab Gitlin. *BitTorrent gets US$8.75 million in VC money*. 29 de set. de 2005. URL: http://arstechnica.com/uncategorized/2005/09/5363-2/.

[18] Daniela Hernandez. *April 13, 2000: Seek and Destroy – Metallica Sues Napster*. [Online; acessado em 30-setembro-2013]. 13 de abr. de 2012. URL: http://www.wired.com/thisdayintech/2012/04/april-13-2000-seek-and-destroy-metallica-sues-napster/.

[19] *HPC Data Repository*. [Online; acessado em 7-outubro-2013]. URL: http://www.hpc.fsu.edu/index.php?option=com_wrapper&view=wrapper&Itemid=80.

[20] Sandvine Inc. *Global Internet Phenomena Report --- 1H 2013*. Online; retirado de http://macaubas.com/wp-content/uploads/2013/05/Sandvine_Global_Internet_Phenomena_Report_1H_2013.pdf. 2013. URL: https://www.sandvine.com/downloads/general/global-internet-phenomena/2013/sandvine-global-internet-phenomena-report-1h-2013.pdf.

[21] Christopher Jones. *Metallica Rips Napster*. [Online; acessado em 30-setembro-2013]. 13 de abr. de 2000. URL: http://www.wired.com/politics/law/news/2000/04/35670.

[22] David Kravets. *Dec. 7, 1999: RIAA Sues Napster*. [Online; acessado em 30-setembro-2013]. 7 de dez. de 2009. URL: http://www.wired.com/thisdayintech/2009/12/1207riaa-sues-napster/.

[23] J.F. Kurose e K.W. Ross. *Computer Networking: A Top-Down Approach*. Pearson Education, Limited, 2010. ISBN: 9780131365483. URL: http://books.google.com.br/books?id=2hv3PgAACAAJ.

[24] M. Lehmann et al. "Swarming: como BitTorrent revolucionou a Internet". Em: *Atualizações em Informática*. Ed. por PUC-Rio. Vol. 1. [Online; accessed 23-outubro-2013]. Rio de Janeiro, 2011. Cap. 6, pp. 209–258. URL: http://www.lbd.dcc.ufmg.br/colecoes/jai/2012/006.pdf.

[25] Andrew Loewenstern e Arvid Norberg. *BitTorrent Enhancement Proposals --- DHT Protocol*. 29 de fev. de 2008. URL: http://www.bittorrent.org/beps/bep_0005.html.

[26] Petar Maymounkov e David Mazières. "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric". Em: (). [Online; acessado em 30-December-2013], p. 6. URL: http://pdos.csail.mit.edu/~petar/papers/maymounkov-kademlia.pdf.

[27] Petar Maymounkov e David Mazières. "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric". Em: *Revised Papers from the First International Workshop on Peer-to-Peer Systems*. IPTPS '01. London, UK, UK: Springer-Verlag, 2002, pp. 53–65. ISBN: 3-540-44179-4. URL: http://dl.acm.org/citation.cfm?id=646334.687801.

[28] *Miro*. [Online; acessado em 7-outubro-2013]. URL: http://getmiro.com.

[29] *Moving Image Archive - Night of the Living Dead (1968)*. [Online; acessado em 16-outubro-2013]. URL: http://archive.org/details/night_of_the_living_dead.

[30] *Napster: 20 million users*. [Online; acessado em 30-setembro-2013]. 19 de jul. de 2000. URL: http://cnnfn.cnn.com/2000/07/19/technology/napster/index.htm.

[31] *NRKbeta*. [Online; acessado em 7-outubro-2013]. URL: http://nrkbeta.no/bittorrent/.

[32] C. Greg Plaxton, Rajmohan Rajaraman e Andréa W. Richa. "Accessing Nearby Copies of Replicated Objects in a Distributed Environment". Em: *Proceedings of the Ninth Annual ACM Symposium on Parallel Algorithms and Architectures*. SPAA '97. Newport, Rhode Island, New York, NY, USA: ACM, 1997, pp. 311–320. ISBN: 0-89791-890-8. DOI: 10.1145/258492.258523. URL: http://doi.acm.org/10.1145/258492.258523.

[33] *Press Release: Global Napster usage plummets, but new file-sharing alternatives gaining ground, reports Jupiter Media Metrix*. [Online; acessado em 8-outubro-2013]. Out. de 2013. URL: http://www.lse.ac.uk/media@lse/documents/MPP/LSE-MPP-Policy-Brief-9-Copyright-and-Creation.pdf.

[34] *Rhapsody.com*. [Online; acessado em 30-setembro-2013]. URL: http://www.rhapsody.com.

[35] Stefan Saroiu, P. Krishna Gummadi e Steven D. Gribble. "A Measurement Study of Peer-to-Peer File Sharing Systems". Em: 2002.

[36] Clive Thompson. *The BitTorrent Effect*. Jan. de 2005. URL: http://www.wired.com/wired/archive/13.01/bittorrent.html.

[37] "Um pouco de história: redes P2P de compartilhamento de arquivos". Em: *Revista PnP* 10 (out. de 2008). [Online; retirado de http://www.thecnica.com/artigos/PnP_10_02.pdf], p. 12. URL: http://www.revistapnp.com.br/pnp_10.php.

[38] *VODO --- About*. [Online; acessado em 7-outubro-2013]. URL: http://www.dgmlive.com/help.htm#whatisbittorrent.

[39] Michael Welzl. *Peer to Peer Systems --- Structured P2P file sharing systems*. [Online; accessed 17-December-2013]. University of Innsbruck, Austria. URL: http://heim.ifi.uio.no/michawe/teaching/p2p-ws08/p2p-4-6.pdf.

[40] Wikibooks. *The World of Peer-to-Peer (P2P) --- Wikibooks, The Free Textbook Project*. [Online; acessado em 3-outubro-2013]. 2012. URL: http://en.wikibooks.org/w/index.php?title=The_World_of_Peer-to-Peer_(P2P)&oldid=2316492.

[41] Wikipedia. *AM Records, Inc. v. Napster, Inc. --- Wikipedia, The Free Encyclopedia*. [Online; acessado em 2-dezembro-2013]. 2013. URL: http://en.wikipedia.org/w/index.php?title=A%26M_Records,_Inc._v._Napster,_Inc.&oldid=579733684.

[42] Wikipedia. *Anycast --- Wikipedia, The Free Encyclopedia*. [Online; acessado em 2-outubro-2013]. 2013. URL: http://en.wikipedia.org/w/index.php?title=Anycast&oldid=574680440.

[43] Wikipedia. *Audiogalaxy --- Wikipedia, The Free Encyclopedia*. [Online; acessado em 29-setembro-2013]. 2013. URL: http://en.wikipedia.org/w/index.php?title=Audiogalaxy&oldid=560950036.

[44] Wikipedia. *BitTorrent --- Wikipedia, The Free Encyclopedia*. [Online; accessed 28-October-2013]. 2013. URL: http://en.wikipedia.org/w/index.php?title=BitTorrent&oldid=578877679.

[45] Wikipedia. *Bram Cohen --- Wikipedia, The Free Encyclopedia*. [Online; acessado em 7-outubro-2013]. 2013. URL: http://en.wikipedia.org/w/index.php?title=Bram_Cohen&oldid=574084830.

[46] Wikipedia. *EDonkey network --- Wikipedia, The Free Encyclopedia*. [Online; acessado em 3-outubro-2013]. 2013. URL: http://en.wikipedia.org/w/index.php?title=EDonkey_network&oldid=568576016.

[47] Wikipedia. *File sharing --- Wikipedia, The Free Encyclopedia*. [Online; acessado em maio de 2013]. 2013. URL: http://en.wikipedia.org/w/index.php?title=File_sharing&oldid=556034682.

[48] Wikipedia. *Gnutella2 --- Wikipedia, The Free Encyclopedia*. [Online; acessado em 2-outubro-2013]. 2013. URL: http://en.wikipedia.org/w/index.php?title=Gnutella2&oldid=556794729.

[49] Wikipedia. *Gnutella --- Wikipedia, The Free Encyclopedia*. [Online; acessado em 2-outubro-2013]. 2013. URL: http://en.wikipedia.org/w/index.php?title=Gnutella&oldid=574304390.

[50] Wikipedia. *Hash function --- Wikipedia, The Free Encyclopedia*. [Online; acessado em 5-outubro-2013]. 2013. URL: http://en.wikipedia.org/w/index.php?title=Hash_function&oldid=574871670.

[51] Wikipedia. *Hash table --- Wikipedia, The Free Encyclopedia*. [Online; acessado em 5-outubro-2013]. 2013. URL: http://en.wikipedia.org/w/index.php?title=Hash_table&oldid=575499828.

[52] Wikipedia. *Internet service provider --- Wikipedia, The Free Encyclopedia*. [Online; acessado em 29-setembro-2013]. 2013. URL: http://en.wikipedia.org/w/index.php?title=Internet_service_provider&oldid=573549991.

[53] Wikipedia. *Kademlia --- Wikipedia, The Free Encyclopedia*. [Online; acessado em 3-outubro-2013]. 2013. URL: http://en.wikipedia.org/w/index.php?title=Kademlia&oldid=575742258.

[54] Wikipedia. *Magic cookie --- Wikipedia, The Free Encyclopedia*. [Online; accessed 31-December-2013]. 2013. URL: http://en.wikipedia.org/w/index.php?title=Magic_cookie&oldid=568707666.

[55] Wikipedia. *Motion Picture Association of America --- Wikipedia, The Free Encyclopedia*. [Online; acessado em 7-outubro-2013]. 2013. URL: http://en.wikipedia.org/w/index.php?title=Motion_Picture_Association_of_America&oldid=575124240.

[56] Wikipedia. *MP3.com --- Wikipedia, The Free Encyclopedia*. [Online; acessado em 29-setembro-2013]. 2013. URL: http://en.wikipedia.org/w/index.php?title=MP3.com&oldid=571025541.

[57] Wikipedia. *MP3 --- Wikipedia, The Free Encyclopedia*. [Online; acessado em 29-setembro-2013]. 2013. URL: http://en.wikipedia.org/w/index.php?title=MP3&oldid=574123988.

[58] Wikipedia. *Napster --- Wikipedia, The Free Encyclopedia*. [Online; acessado em 30-setembro-2013]. 2013. URL: http://en.wikipedia.org/w/index.php?title=Napster&oldid=573999770.

[59] Wikipedia. *Percent-encoding --- Wikipedia, The Free Encyclopedia*. [Online; accessed 27-October-2013]. 2013. URL: http://en.wikipedia.org/w/index.php?title=Percent-encoding&oldid=573113056.

[60] Wikipedia. *Recording Industry Association of America --- Wikipedia, The Free Encyclopedia*. [Online; acessado em 30-setembro-2013]. 2013. URL: http://en.wikipedia.org/w/index.php?title=Recording_Industry_Association_of_America&oldid=574192660.

[61] Wikipedia. *Segmented file transfer --- Wikipedia, The Free Encyclopedia*. [Online; acessado em 5-outubro-2013]. 2013. URL: http://en.wikipedia.org/w/index.php?title=Segmented_file_transfer&oldid=533100656.

[62] Wikipedia. *Timeline of file sharing --- Wikipedia, The Free Encyclopedia*. [Online; acessado em 28 de setembro de 2013]. 2013. URL: http://en.wikipedia.org/w/index.php?title=Timeline_of_file_sharing&oldid=571061187.

[63] Wikipédia. *BitTorrent --- Wikipédia, a enciclopédia livre*. [Online; accessed 22-outubro-2013]. 2013. URL: http://pt.wikipedia.org/w/index.php?title=BitTorrent&oldid=36953538.

[64] Theory.org Wiki. *BitTorrentSpecification --- Theory.org Wiki,* [Online; accessed 17-October-2013]. 2013. URL: https://wiki.theory.org/index.php?title=BitTorrentSpecification&oldid=2527#Bencoding.

[65] Theory.org Wiki. *BitTorrentSpecification --- Theory.org Wiki,* [Online; accessed 17-October-2013]. 2013. URL: https://wiki.theory.org/index.php?title=BitTorrentSpecification&oldid=2527#Tracker_Response.

Capítulo 8

Visão Pessoal