Instituto Politécnico de Castelo Branco
Escola Superior de Tecnologia

# Sistema Flexível de Distribuição de Imagens de Disco Rígido

Pedro Gonçalves

Trabalho de Mestrado
Desenvolvimento de Software e Sistemas Interactivos

Trabalho Efectuado sob a orientação do
Professor Doutor Osvaldo Arede dos Santos

Junho 2012

# DECLARAÇÃO

Nome: Pedro Gonçalves

E-mail: pedro.goncalv3s@gmail.com

Bilhete de Identidade: 11586167

Título do trabalho: Sistema Flexível de Distribuição de Imagens de Disco Rígido

Orientador: Doutor Osvaldo Arede dos Santos

Ano de conclusão: 2012

Designação do Mestrado: Desenvolvimento de Software e Sistemas Interactivos



Instituto Politécnico de Castelo Branco, ____/____/_____

Assinatura: _____

**Agradecimentos**

A todos os que me apoiaram e ajudaram durante a realização deste projecto.

Um agradecimento especial à Escola Superior de Tecnologia de Castelo Branco pelo acesso facilitado (muitas vezes fora de horas) às instalações e equipamentos onde os testes foram realizados.

**Palavras chave**

distribuição de imagens de disco rígido, bittorrent, LAN

**Resumo**

A distribuição de imagens de disco rígido num cenário de salas de aula é um desafio para os recursos de rede. Neste trabalho é proposto desenvolver um *software* que faça uma gestão inteligente da forma mais eficiente de distribuição das imagens. Resultados experimentais demonstraram que a utilização de um típico cliente de BitTorrent com uma estratégia de transferência por blocos, é a forma mais eficiente em todos os casos excepto quando a transferência é apenas para um único computador. Esta estratégia de transferência foi desenvolvida pois não existe armazenamento temporário para o ficheiro de imagem de disco completo, no ambiente *GNU/Linux live distribution* usado na instalação das imagens.

**Keywords**

hard disk image distribution, bittorrent, LAN

**Abstract**

A classroom hard disk image deployment scenario is quite challenging on network resources. In this work it is proposed to develop a software to inteligently manage the most eficient way to deploy those images. Experimental results showed that the usage of a tipical BitTorrent client with a download by blocks strategy, is the most eficient way in all cases except when the transfer is for only one computer. This download strategy had to be developed because there is no temporary storage for the whole hard disk image file in the GNU/Linux live distribution environment used for the image deployment.

# Índice geral

# Índice de Figuras

# Índice de Tabelas

# Lista de abreviaturas

CD – Compact Disc
DHCP – Dynamic Host Configuration Protocol
DHT – Distributed Hash Table
ESTCB – Escola Superior de Tecnologia de Castelo Branco
FEC – Forward Error Correction
FTP – File Transfer Protocol
HTTP – HyperText Transfer Protocol
IP – Internet Protocol
JSON-RPC – JavaScript Object Notation - Remote Procedure Call
LAN – Local Area Network
LPD – Local Peer Discovery
MBR – Master Boot Record
NTFS – New Technology File System
PEX – Peer EXchange
PXE – Preboot eXecution Environment
RAM – Random Access Memory
TCP – Transmission Control Protocol
TFTP – Trivial File Transfer Protocol
TI – Tecnologias de Informação
UDP – User Datagram Protocol
USB – Universal Serial Bus
WDS – Windows Deployment Service
XML-RPC – Extensible Markup Language - Remote Procedure Call

# 1. Introdução

## 1.1 Contexto

Em redes informáticas com um grande número de computadores semelhantes é comum o uso de imagens de disco rígido (de agora em diante denominadas apenas "imagens") para reverter cada computador a uma configuração básica inicial quando necessário. Esta configuração tipicamente inclui não só o sistema operativo mas também algumas aplicações necessárias para o processo de negócio da organização. Esta técnica também se adequa a escolas onde tipicamente as salas de computadores são constituídas por cerca de 20 equipamentos idênticos. A situação pode ser ainda mais específica existindo diferenças no *software* instalado e licenciado de sala para sala, como por exemplo a sala de edição de vídeo, a sala de multimédia, a sala de programação, etc.

A utilização de imagens para instalação do sistema operativo já com *software* adicional num grande número de computadores similares é uma técnica bem estabelecida e como tal os produtores de sistemas operativos e grandes empresas de *software* disponibilizam ferramentas comerciais para o efeito (Symantec Corporation, 2012; Microsoft Corporation, 2012c; International Business Machines Corp., 2012; Acronis Inc., 2012a).

Tendo em conta a natureza dos dados que serão transferidos pela rede (sistema operativo e *software* adicional a multiplicar por 20 computadores) facilmente se percebe que este tipo de aplicações consegue esgotar a capacidade de transferência de uma rede local (LAN) tornando a distribuição das imagens pelos computadores um processo relativamente lento. A solução mais óbvia para minorar este problema é a aquisição de equipamento de rede com maior capacidade de transferência, se bem que, adquirir equipamento dispendioso com base numa necessidade que apenas ocorre esporadicamente pode não ser justificável.

Tipicamente este tipo de aplicações trabalham sobre o *Internet Protocol* (IP), usando ligações ponto-a-ponto em *Transmission Control Protocol* (TCP) ou ponto-multiponto em *User Datagram Protocol* (UDP) mais concretamente *multicast* (Symantec Corporation, 2012; Microsoft Corporation, 2012d; Acronis Inc., 2012b). Ambas as soluções têm algumas desvantagens. Aquando do uso do TCP, o problema surge quando o número de clientes a transferir dados simultaneamente do mesmo servidor aumenta, o que faz com que a largura de banda disponível para cada um rapidamente decresça pois tem de ser partilhada por um número crescente de clientes. Também é conhecido que os algoritmos de gestão de congestionamento (*congestion avoidance algorithms*) têm pior desempenho após atingida a *bottleneck bandwidth* (Wang, Pau and Yamada, 2004). O protocolo UDP/*multicast* não tem este problema mas tem a desvantagem de não garantir a entrega dos dados (Postel, 1980). Até técnicas de *Forward Error Correction* (FEC) aplicadas ao UDP (Carle, 1997) não ajudam neste caso pois os testes preliminares no

ambiente real revelaram um desempenho extremamente fraco.

O protocolo BitTorrent (Cohen, 2008) surgiu como uma alternativa que permite aliviar o servidor principal da transferência total dos dados, distribuindo essa responsabilidade pelos clientes que já tenham transferido previamente os dados. Neste protocolo, a parte da transferência dos dados funciona sobre TCP (Cohen, 2008) e possui mecanismos por forma a garantir que os dados não estão corrompidos (Cohen, 2008, 2003).

Neste trabalho propõe-se uma arquitetura que use o protocolo BitTorrent para disseminação de imagens sobre uma rede local, adaptando-o às limitações de um ambiente *GNU/Linux live distribution* sem armazenamento temporário para o ficheiro da imagem de disco.

## 1.2 Motivação

A Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco (ESTCB), como forte utilizadora deste tipo de aplicação para distribuir o sistema operativo e restante *software* para as suas salas de aula, depara-se frequentemente com este problema da incapacidade da rede local dar resposta à transferência da grande quantidade de dados que as imagens representam para os computadores das salas de aulas.

Devido à grande ocupação das salas de aulas durante o período letivo, as alterações ao *software* instalado são executadas durante a noite devido ao tempo necessário para o processo, no entanto isto levanta problemas quando ocorrem erros pois no dia seguinte os computadores não estarão operacionais.

Melhorar a eficiência deste processo reduzindo o tempo necessário para distribuir as imagens de disco pelos computadores, facilitaria a gestão das salas de aula pelos seus responsáveis.

## 1.3 Objetivos

Estudar, implementar e testar um sistema inteligente de distribuição de imagens de disco rígido que permita:

- Criar imagens de um computador e guardá-las num servidor;
- Distribuir imagens a partir do servidor para um ou vários clientes;
- Baseando-se no estudo efetuado sobre TCP/UDP/BitTorrent, o sistema deve inteligentemente escolher a melhor forma de distribuição das imagens tendo em conta parâmetros como o tamanho da imagem, velocidade da rede, número de clientes em simultâneo, etc.;
- Gerir imagens de disco e outros parâmetros da aplicação no servidor através de uma interface WEB.

O sistema desenvolvido apenas se irá debruçar sobre a distribuição eficiente das imagens de disco rígido. O processo de criação, compressão, descompressão estará a cargo de aplicações já existentes que poderão ter as suas próprias limitações mas que não serão abordadas aqui.

## 1.4 Estrutura deste relatório

No primeiro capítulo, que corresponde à introdução, é apresentado o contexto, motivação e objectivos do projecto.

No segundo capítulo, serão apresentadas ferramentas populares para manipulação de imagens disco rígido bem como os passos necessários para criação das mesmas.

No terceiro capítulo, é feita uma análise de requisitos baseando-se nas limitações e particularidades da rede local da ESTCB.

No quarto capítulo, é realizado um estudo comparativo de várias tecnologias de transferência possíveis.

No quinto capítulo, é abordado o projecto do desenvolvimento do *software* propriamente dito.

No sexto capítulo é apresentada a conclusão.

# 2. Imagens de disco rígido

No início deste capítulo serão abordadas algumas ferramentas populares de manipulação de imagens de disco rígido, seguindo-se uma explicação genérica dos passos necessários para criar uma imagem.

## 2.1 Ferramentas mais populares

A instalação de sistemas é uma tarefa bastante comum para os departamentos de tecnologias de informação (TI). Consiste em preparar um computador com todo o *software* que necessita (sistema operativo e conjunto de aplicações base) (Stanford_University, 2010). Estas podem ser feitas essencialmente de duas formas: instalação automatizada do sistema operativo ou clonagem de imagens capturadas.

A primeira solução usa ferramentas diferentes consoante o sistema operativo em questão: *Windows Deployment Service* (WDS) para o Windows (Microsoft Corporation, 2012c), *Fully Automated Installation* para Debian e Ubuntu (Lange, 2012) e *Kickstart* para RedHat e CentOS (Red Hat, n.d.). Neste caso também é necessário automatizar a instalação de todo o *software* adicional que venha a ser necessário, tendo de ser preparado caso a caso. A vantagem principal desta técnica é que funciona independentemente do *hardware* (físico ou virtual) pois é semelhante a uma instalação de raiz.

A segunda solução não tem o tempo inicial de preparação (além da normal instalação do *software* num computador) e podem ser usados métodos universais, excetuando a ferramenta que lê os dados das partições do disco rígido, que depende do formato da partição em questão. A desvantagem é que é necessário criar uma imagem diferente para cada tipo de computador/*hardware* devido aos drivers que já vêm instalados nela. Alguns *softwares* procuram atenuar este ponto permitindo a instalação de *drivers* adicionais na imagem como por exemplo o DeployAnywhere da Symantec (Symantec Corporation, 2012), o Universal Deploy da Acronis (Acronis Inc., 2012a) e o Driver Injection do IBM Tivoli (International Business Machines Corp., 2012).

Não é fácil encontrar informação sobre os protocolos usados por estes produtos para a transferência dos dados embora alguns deles mencionem a possibilidade de utilização de *multicast* (Symantec Corporation, 2012; Microsoft Corporation, 2012d; Acronis Inc., 2012b).

## 2.2 Como criar uma imagem

A criação de uma imagem completa envolve vários passos devido à forma como os dados num disco rígido estão estruturados. Um disco rígido encontra-se dividido em múltiplas unidades lógicas denominadas "partições". Estas serão apresentadas pelo sistema operativo como discos independentes. A informação destas unidades lógicas é armazenada numa estrutura denominada "tabela de partições". Estas podem ter vários formatos nomeadamente: "*Apple partition map*", "*BSD disklabel*", "*GUID partition table*" e o mais comum (sem nome) usado de forma predefinida nos *IBM compatible PC* pelos sistemas operativos Microsoft Windows e GNU/Linux que se encontra integrado no *master boot record* (MBR). O MBR contém uma tabela de partições que permite referenciar até quatro partições bem como o código de arranque do sistema operativo.

Assim sendo, para efetuar uma imagem de um disco rígido é necessário, no mínimo, guardar os dados do MBR e de seguida os dados de cada uma das partições que este referencia. O MBR é uma estrutura fixa de 512 bytes, no entanto as partições podem ter vários tamanhos e formatações pelo que será necessário usar técnicas/ferramentas distintas consoante o tipo destas. Por exemplo, para guardar uma partição no formato *New Technology File System* (NTFS) do Microsoft Windows tem de se usar uma técnica/ferramenta diferente da usada para guardar uma partição no formato *ext4* disponível em GNU/Linux pois possuem estruturas completamente diferentes.

A forma mais segura de efetuar a guarda/restauro destes dados é garantindo que os ficheiros não estão a ser usados no momento. Isto implica que o computador tenha de executar temporariamente um sistema operativo alternativo por forma a que o sistema operativo nativo que se encontra instalado (e que naturalmente usa os ficheiros da sua instalação) não esteja em funcionamento. Ferramentas mais recentes como o "Microsoft Windows 7 Backup and Restore" (Microsoft Corporation, 2012a) misturam as funcionalidades de cópia de segurança e imagem, e como tal já permitem criar uma imagem com o sistema operativo em execução (*online*). O restauro de ficheiros pode ser *online* para ficheiros de dados, no entanto, para recuperar uma imagem de sistema a ferramenta de restauro funciona em *offline* a partir dos modos de recuperação do Microsoft Windows (Microsoft Corporation, 2012b).

# 3. Análise de requisitos

Nesta secção irá ser analisado o cenário atual existente na ESTCB, com vista a identificar os problemas e perceber as respostas que o sistema a desenvolver terá de encontrar. É importante ter presente que toda a problemática se coloca na fase de restauro da imagem e não propriamente na fase de guardar a imagem. Como o guardar da imagem no servidor é feito a partir de um só computador, pode ser usado um protocolo simples de transferência de ficheiros como o *File Transfer Protocol* (FTP) (Postel and Reynolds, 1985) pois a largura de banda disponível será suficiente. O problema coloca-se na fase de distribuição dessa imagem simultaneamente para os computadores de uma sala de aula (cerca de 20). Toda a análise das limitações/requisitos se vai centrar neste aspeto.

Este cenário tem por base o atual sistema (básico) de distribuição de imagens de disco rígido da ESTCB que funciona no modelo cliente-servidor (Tanenbaum, 2002) baseado em *GNU/Linux* sobre o protocolo *Hypertext Transfer Protocol* (HTTP) (Fielding et al., 1999). Os clientes executam uma *GNU/Linux live distribution*, por forma a ser possível reescrever o disco rígido inteiro, a partir da qual é feita a transferência dos dados por HTTP, a sua descompressão e escrita no disco rígido.

## 3.1 Esquema de rede atual

De uma forma simplista podemos descrever o esquema de rede atual como sendo constituído por um servidor que possui uma ligação a 1Gbit/s à rede local, que mantém esta ligação até ao *switch* do edifício, onde depois é dividida em 100Mbit/s para cada sala de aula. Dentro desta, os computadores vão partilhar estes 100Mbit/s entre eles, pelo que aquele é o ponto de estrangulamento (*bottleneck*) do sistema tal como apresentado no diagrama seguinte.

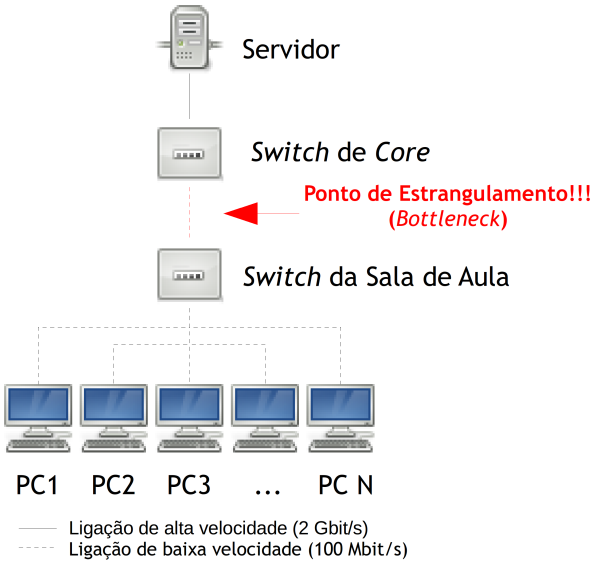



**Figura 1** - Ponto de estrangulamento no cenário de sala de aula

## 3.2 Limitações da infraestrutura existente

Considerando que os dados se encontram num servidor e serão transferidos pela rede para os clientes onde estes fazem a descompressão dos dados, podem encontrar-se limitações no desempenho da rede, do disco rígido dos clientes e/ou do servidor e da CPU dos clientes. A velocidade de escrita no disco foi excluída como limitadora pois nos equipamentos atuais facilmente se atingem os 70MB/s (Bestofmedia Group, 2012) o que é claramente superior às velocidades de transferência espetáveis para múltiplos clientes em simultâneo. Neste sentido foram realizados testes aos restantes fatores em vários cenários.

O servidor usado para os testes consiste de um Intel Pentium 4 a 2.8GHz com 3GB de memória RAM e 500GB de disco rígido. Este possui o sistema operativo GNU/Linux CentOS 5.7 com Apache 2.2.3. Os clientes correm a *GNU/Linux live distribution* GRML (Grml Live Linux, 2012) versão 2010.12.

### 3.2.1 Velocidade máxima de transferência da rede da ESTCB

Por forma o obter um valor real da velocidade máxima de transferência de rede da EST foram realizados testes em *TCP* com base no seguinte esquema de ligação:

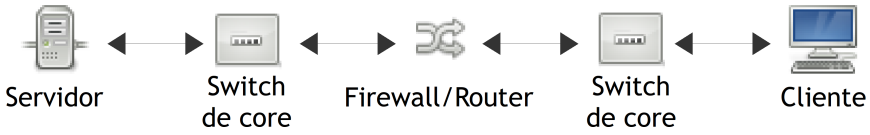



**Figura 2** - Esquema de testes da velocidade máxima de transferência da rede

De salientar que o servidor se encontra num segmento de rede independente da rede local e protegido por uma *firewall*.

Foram feitas várias transferências através do protocolo HTTP usando o comando *wget* em *GNU/Linux* que no final apresenta o tempo usado e a velocidade média de transferência em *bytes* por segundo.

O comando usado para o efeito foi:

```
wget -O /dev/null http://servidor.est.ipcb.pt/ficheiro_com_X_GB
```

Do qual os resultados obtidos foram:

| Velocidade da porta do switch onde liga o cliente | Tamanho do ficheiro de teste | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1GB | | 5GB | | 10GB | | 20GB | |
| | seg | MB/s | seg | MB/s | seg | MB/s | seg | MB/s |
| 100MBit/s | 91,0 | 11,2 | 456,0 | 11,2 | 913,0 | 11,2 | 1826,0 | 11,2 |
| | 91,0 | 11,2 | 456,0 | 11,2 | 913,0 | 11,2 | 1827,0 | 11,2 |
| | 91,0 | 11,2 | 456,0 | 11,2 | 913,0 | 11,2 | 1839,0 | 11,2 |
| | 91,0 | 11,2 | 456,0 | 11,2 | 913,0 | 11,2 | 1826,0 | 11,2 |
| | 91,0 | 11,2 | 456,0 | 11,2 | 913,0 | 11,2 | 1826,0 | 11,2 |
| | | | | | | | | |
| 1GBit/s | 26,0 | 40,9 | 92,0 | 56,3 | 206,0 | 48,1 | 511,0 | 41,8 |
| | 9,5 | 108,0 | 92,0 | 56,3 | 206,0 | 48,4 | 511,0 | 41,8 |
| | 9,7 | 100,0 | 92,0 | 56,5 | 207,0 | 48,2 | 512,0 | 41,4 |
| | 9,6 | 105,0 | 92,0 | 56,2 | 206,0 | 48,3 | 511,0 | 41,7 |
| | 9,5 | 110,0 | 92,0 | 56,3 | 206,0 | 48,4 | 511,0 | 41,7 |

**Tabela 1** - Resultados experimentalmente obtidos da velocidade máxima de transferência da rede (em segundos e MB/s)

Por forma a aumentar a fiabilidade das medições, foi retirado o melhor e o pior resultado das anteriores e calculada a média das 3 restantes obtendo-se:

| Velocidade da porta do switch onde liga o cliente | Tamanho do ficheiro de teste | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1GB | | 5GB | | 10GB | | 20GB | |
| | seg | MB/s | seg | MB/s | seg | MB/s | seg | MB/s |
| 100MBit/s | 91,0 | 11,2 | 456,0 | 11,2 | 913,0 | 11,2 | 1826,3 | 11,2 |
| 1GBit/s | 9,6 | 104,3 | 92,0 | 56,3 | 206,0 | 48,3 | 511,0 | 41,7 |

**Tabela 2** - Médias dos resultados experimentalmente obtidos da velocidade máxima de transferência da rede (em segundos e MB/s)

Os resultados obtidos são bastante consistentes, sendo que na velocidade relevante para este estudo (100Mbit/s) se consegue uma taxa de transferência constante de 11.2MB/s (89.6Mbit/s). Esta será a velocidade máxima de transferência que os clientes da rede da ESTCB terão acesso.

Relativamente à comunicação a 1Gbit/s a taxa máxima de transferência é de 104.3MB/s (834.4Mbit/s). Numa primeira leitura este valor pode não parecer o correto, pois nota-se uma grande diferença na transferência a 1Gbit/s entre o ficheiros de 1GB e os restantes. A conclusão obtida é que o limite de transferência é estabelecido pela velocidade de leitura do disco rígido do servidor. A suportar esta teoria temos que o primeiro resultado obtido na transferência do ficheiro de 1GB também é muito inferior. Como o servidor possui 3GB de RAM, que normalmente

estão livres, o ficheiro ficou na *cache* (Tanenbaum, 2001) em memória RAM após a primeira transferência permitindo que as restantes ocorressem à velocidade máxima que a rede suporta.

Convém salientar mais uma vez que estes valores representam o melhor cenário possível. As redes têm flutuações de tráfego que podem degradar consideravelmente o desempenho, mas que são impossíveis de prever.

### 3.2.2 Compressão máxima das ferramentas GNU/Linux

Por forma a reduzir o espaço de armazenamento das imagens e também a quantidade de dados a transferir pela rede, as imagens devem ser comprimidas. Neste sentido é necessário comparar as tecnologias de compressão existentes. Devido à escolha de sistema operativo, serão comparadas as ferramentas de compressão tipicamente disponibilizadas em *GNU/Linux*: gzip, bzip2, lzma e xz.

Com vista a realizar os testes com dados o mais reais possível, foi gerada uma imagem típica do sistema operativo Microsoft Windows XP da qual foi retirado 1GB de dados. Para tal foi usado o comando:

```
dd if=/dev/particao_do_windows of=dados.img bs=1M count=1024
```

Ao aplicar as várias ferramentas de compressão, obtiveram-se os seguintes resultados:

| | | Ferramenta usada na compressão | | | | |
|---|---|---|---|---|---|---|
| | | Original | gzip | bzip2 | lzma | xz |
| Tamanho em ... | Bytes | 1073741824 | 576188601 | 580370549 | 458146910 | 457145060 |
| | Mbytes | 1024 | 549,50 | 553,48 | 436,92 | 435,97 |
| | % | 100% | 53,66% | 54,05% | 42,67% | 42,57% |

**Tabela 3** - Comparação da compressão máxima das ferramentas GNU/Linux

Os resultados mostram dois claros vencedores, sendo que a ferramenta "xz" é ligeiramente superior.

### 3.2.3 Velocidade máxima de descompressão nos clientes

Outro facto importante é a velocidade com que os clientes efetuam descompressão dos dados. Mesmo que a rede consiga transferir dados muito rapidamente, se a descompressão for lenta, o resultado final será limitado pela velocidade de descompressão.

Considerando que na ESTCB existem diversos tipos de clientes, e que a descompressão é um processo intensivo a nível de processador, esta medição foi realizada em computadores com

diferentes tipos de processadores. De salientar que estas ferramentas não estão preparadas para aproveitar as vantagens de sistemas multi-processador (Tanenbaum, 2001).

A velocidade de descompressão foi medida da seguinte forma:

```
cat ficheiro_comprimido | pv | descompressor > /dev/null
```

A ferramenta "pv" permite monitorizar a velocidade a que os dados estão a passar num *pipe* (Wood, 2012). No final apresenta a velocidade média a que os dados transitaram.

Os resultados obtidos foram:

| | | | | Ferramenta de compressão | |
|---|---|---|---|---|---|
| | CPU | Cores | RAM | lzma | xz |
| Características do computador | Intel(R) Pentium(R) 4 CPU 2.80GHz | 1 (+HT) | 3GB | 4,05 | 5,24 |
| | Intel(R) Core(TM)2 CPU 6300 @ 1.86GHz | 2 | 3GB | 7,18 | 8,31 |
| | Intel(R) Pentium(R) 4 CPU 2.80GHz | 1 | 512MB | 4,06 | 4,76 |
| | Intel(R) Core(TM)2 Duo CPU E7500 @ 2.93GHz | 2 | 2GB | 9,65 | 10,7 |

**HT = hyper-threading*

**Tabela 4** - Comparação da velocidade máxima de descompressão em computadores diferenciados (em MB/s)

## 3.3 Conclusões preliminares

Tanto a ferramenta "lzma" como a "xz" apresentam taxas de compressão semelhantes sendo que o "xz" é ligeiramente melhor. Relativamente à velocidade de descompressão o "xz" também leva vantagem em todos os processadores testados.

A taxa máxima de descompressão é de 10.7 MB/s, apenas disponível num processador recente, mas que mesmo assim não esgota os 11.2 MB/s que a rede da ESTCB disponibiliza (no melhor caso possível) a 100 Mbit/s. Se a descompressão não esgota os limites da rede a 100Mbit/s então muito menos o fará em ligações a 1 Gbit/s, pelo que não se justifica esta ligação a nível dos clientes para os computadores atualmente existentes.

O cenário desejável será então aquele em que se consiga ter uma utilização plena dos recursos de uma rede a 100Mbit/s em cada um dos clientes que recebe a imagem.

De notar que conseguir este tipo de utilização também está muito dependente do equipamento de rede (*switch*) onde estão ligados os clientes. Equipamentos com custo mais baixo pertencentes ao segmento *edge* tipicamente não possuem capacidade para garantir a velocidade máxima simultaneamente em todas as suas portas.

# 4. Investigação das tecnologias de transferência

Tendo em conta o problema identificado anteriormente levanta-se a questão: qual a melhor estratégia para transferência de imagens no cenário já referido (figura 1) que representa a realidade da rede local da ESTCB?

Com vista a responder a esta questão será feito um estudo comparativo entre três tecnologias de transferência: HTTP (TCP), *multicast* (UDP) e BitTorrent (TCP+UDP).

Pretende-se dar oportunidade aos protocolos da camada OSI de transporte (Tanenbaum, 2002) *Transmission Control Protocol* (TCP) e *User Datagram Protocol* (UDP) de demonstrarem os seus pontos fortes. Por um lado o "TCP providencia um fluxo confiável de dados entre dois sistemas" ao "garantir que o outro ponto acuse a recepção dos pacotes enviados" (Stevens, 1994), ao passo que o UDP "providencia um serviço muito mais simples", "mas não garante que os pacotes alcancem o outro ponto" (Stevens, 1994).

Por forma a medir o desempenho destas tecnologias foram estabelecidas três variáveis:

- Quantidade de dados a transferir;
- Número de computadores para onde os dados são transferidos;
- Tempo total do processo (transferência + descompressão + escrita no disco).

Por forma a facilitar a comparação entre tempos e quantidades diferentes de dados as comparações serão realizadas na forma: *quantidade de dados a transferir / tempo total do processo* que será representada na unidade *MBytes/segundo*.

A inclusão da descompressão e da escrita no disco no tempo total do processo são importantes pois o processo a implementar também irá incluir a descompressão e a escrita no disco rígido. Um sistema que consiga paralelizar estas três fases terá à partida maior vantagem do que um que o faça de forma sequencial.

Para por à prova estas tecnologias foi concebido um teste que consiste do seguinte esquema lógico:

**`transferência => descompressão => hash`**

Em que o *hash* substitui o processo de escrita no disco rígido, e serve para validar se os dados no final do processo correspondem aos dados iniciais. Anteriormente já tinha sido estabelecido que o tempo do processo de escrita no disco era negligênciável e o mesmo se passa com o tempo deste processo de *hash*.

Os testes foram repetidos para ficheiros de 1, 5, 10 e 20GB e para 1, 5, 10, 15 e 20 computadores clientes em simultâneo. Mediu-se o tempo total da operação para o cliente mais lento e dividiu-se pela quantidade de dados transmitidos obtendo-se assim a velocidade de transferência em MB/s que será usada para comparação. Foi usado um mecanismo de

sincronização que faz com que todos os clientes iniciem a transferência com uma diferença máxima de 1 segundo, valor este que pode ser desprezado face aos testes que à partida demorariam entre vários minutos até várias horas.

Por forma a melhorar a fiabilidade dos resultados deste teste, será usado o método já aplicado anteriormente em que são realizadas 5 medições das quais será retirado o melhor e o pior resultado e calculada a média dos restantes 3.

Um problema encontrado foi a geração dos ficheiros comprimidos. Como não é possível prever a compressão que determinados dados irão ter, não foi possível gerar ficheiros exatamente com o tamanho pretendido. Assim procurou-se gerar dados aleatórios, prevendo-se pouca ou nenhuma compressão, comprimindo-os de seguida. Segue-se o comando executado para a obtenção do ficheiro de 1GB:

```
dd if=/dev/urandom bs=1G count=1 | xz -9c > 1GB.xz
```

Repetindo este processo para os restantes ficheiros obtiveram-se ficheiros comprimidos ligeiramente superiores ao pretendido como apresentado na tabela seguinte:

| | | Tamanho | | |
|---|---|---|---|---|
| | | Pretendido | Gerado | % |
| Ficheiro de ... | 1GB | 1073741824 | 1073795012 | 100,005 |
| | 5GB | 5368709120 | 5368974804 | 100,005 |
| | 10GB | 10737418240 | 10737949544 | 100,005 |
| | 20GB | 21474836480 | 21475899024 | 100,005 |

**Tabela 5** - Tamanho real dos ficheiros comprimidos usados nos testes

Considerou-se que estas diferenças são negligenciáveis para as medições pretendidas.

## 4.1 Hypertext Transfer Protocol - HTTP (TCP)

O *Hypertext Transfer Protocol* (HTTP) é "um protocolo de nível aplicacional para sistemas de informação distribuídos, colaborativos e de hipermédia. É um protocolo genérico, sem estado (*stateless*) que pode ser usado para muitas tarefas para além do seu uso para hipertexto". "O HTTP é usado pela iniciativa global de informação *World Wide Web* desde 1990". "A comunicação HTTP habitualmente ocorre sobre ligações TCP/IP" (Fielding et al., 1999).

Como previamente mencionado, é a tecnologia usada no sistema atual. Quando combinada com um sistema de gestão justa de largura de banda tem um desempenho muito em linha com o desempenho teórico expectável e que neste caso, basicamente consiste na largura de banda disponível no *bottleneck* a dividir pelo número de clientes a efetuar a transferência.

Apesar de o protocolo TCP já possuir vários algoritmos de prevenção de congestionamento (*congestion avoidance algorithms*) (Stevens, 1994), à medida que o número de clientes aumenta

verifica-se que um ou mais clientes começam a receber menos dados e consequentemente progridem mais lentamente que os restantes. Isto ocorre porque os algoritmos de prevenção de congestionamento não são perfeitos e possuem limitações. Mais informações sobre este tópico podem ser encontradas em (Jacobson, 1988). Como neste caso particular os dados originam todos do servidor de imagens é possível, do lado deste, impedir que uma ou mais ligações monopolizem a largura de banda. Tal foi conseguido aplicando o algoritmo de *stochastic fairness queuing* (McKenney, 1991) que vem incluído nos sistemas operativos *GNU/Linux* através do comando "tc" (Hubert, 2012). Esta classe de algoritmos é "adequada para implementações de alta velocidade por *software*" sendo que o "custo é a perda de determinismo e de garantias absolutas de justiça" (McKenney, 1991). Existem outros algoritmos para o efeito, no entanto, considerou-se ser mais importante a velocidade do que a precisão do algoritmo.

O comando executado em concreto, foi o seguinte:

```
/sbin/tc qdisc add dev <device> root sfq perturb 10
```

Os resultados teóricos esperados deste teste são:

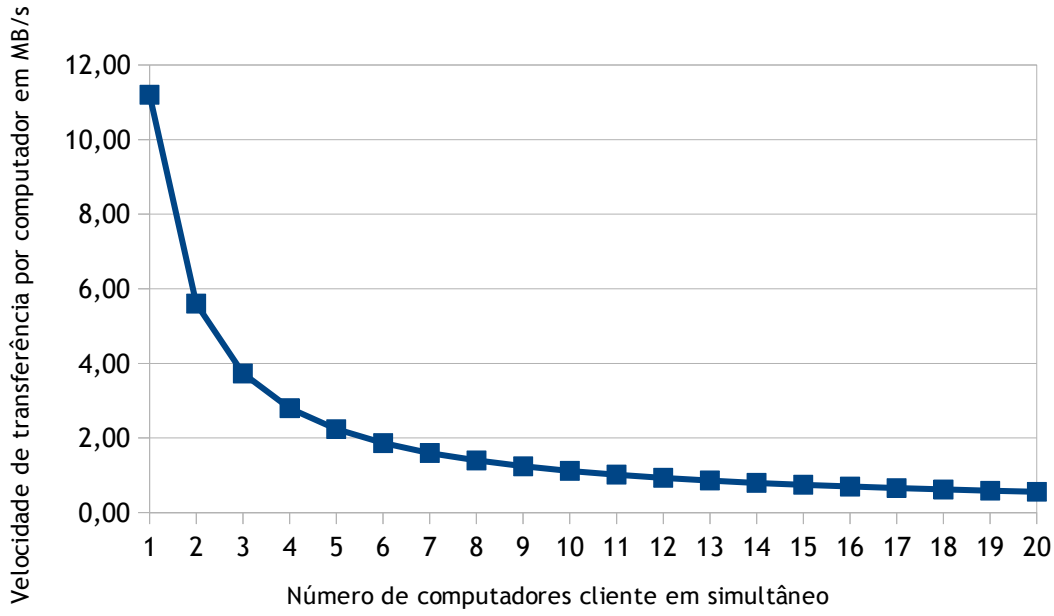



**Figura 3** - Resultados teóricamente esperados para a transferência por HTTP

| | Número de computadores cliente em simultâneo | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | (...) | 10 | (...) | 15 | (...) | 20 |
| Velocidade de transferência por computador em MB/s | 11,20 | 5,60 | 3,73 | 2,80 | 2,24 | | 1,12 | | 0,75 | | 0,56 |

**Tabela 6** - Resultados teóricamente esperados para a transferência por HTTP

De seguida apresentam-se os resultados obtidos experimentalmente. Como em testes anteriores, para a realização das transferências por HTTP foi usada a ferramenta *open-source* "wget" (Free Software Foundation Inc., 2011) que entrega os dados ao descompressor que por sua vez os entrega à ferramenta de *hashing*.

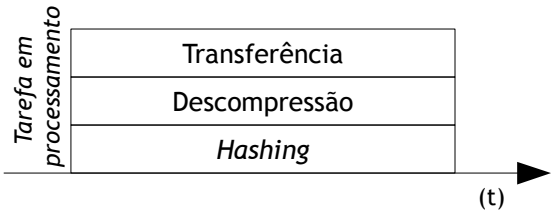



**Figura 4** - Paralelismo de execução de tarefas no teste de transferência por HTTP

Tal efeito foi conseguido através do comando:

```
wget -q -t 25 -O - "ficheiro_de_teste" | xz -dc | sha1sum
```

Do qual foram obtidos os seguintes resultados:

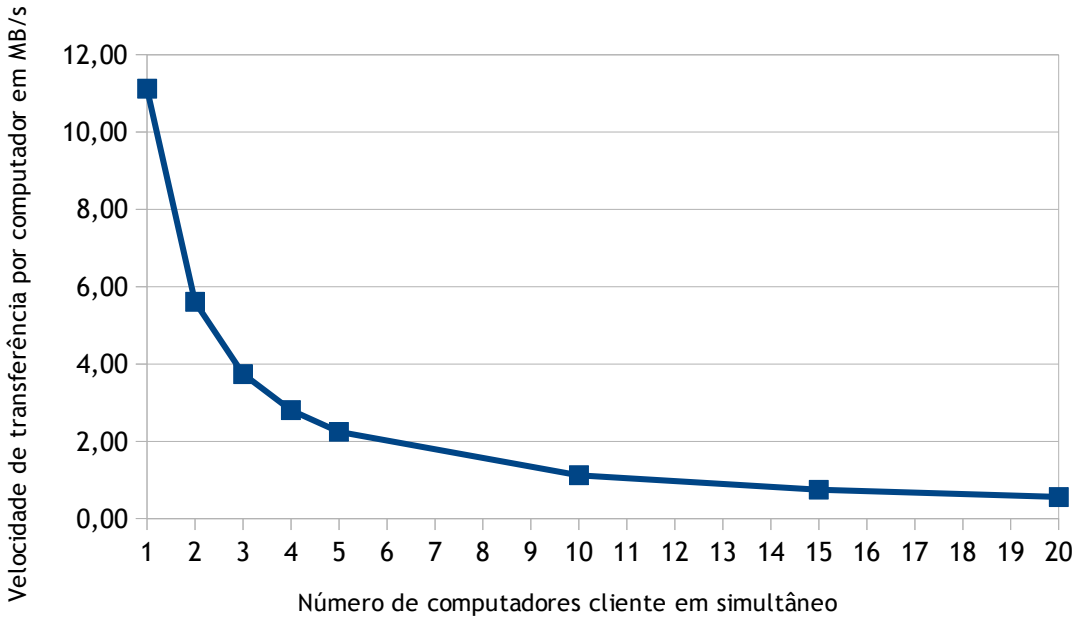



**Figura 5** - Resultados experimentalmente obtidos para a transferência por HTTP (valor médio)

| Número de computadores cliente em simultâneo | Tamanho do ficheiro de teste 1GB | 5GB | 10GB | 20GB | Média |
|---|---|---|---|---|---|
| 1 | 11,21 | 11,15 | 11,05 | 11,06 | **11,12** |
| 2 | 5,61 | 5,61 | 5,61 | 5,61 | **5,61** |
| 3 | 3,74 | 3,74 | 3,74 | 3,74 | **3,74** |
| 4 | 2,81 | 2,81 | 2,81 | 2,81 | **2,81** |
| 5 | 2,25 | 2,25 | 2,24 | 2,24 | **2,24** |
| 10 | 1,12 | 1,12 | 1,12 | 1,12 | **1,12** |
| 15 | 0,75 | 0,75 | 0,75 | 0,75 | **0,75** |
| 20 | 0,56 | 0,56 | 0,56 | 0,56 | **0,56** |

**Tabela 7** - Resultados experimentalmente obtidos para a transferência por HTTP (em MB/s)

Ao contrário da estratégia habitual de fazer 5 medições e efetuar a média dos 3 valores intermédios, as medições para 15 e 20 computadores foram realizadas uma só vez. Isto acontece porque os testes foram realizados nas salas de aulas durante a noite ou o fim de semana. Acontece que estas simulações demorariam mais de 48 horas pelo que nem o fim de semana seria suficiente para as realizar.

Tal como previsto, os dados teóricos e obtidos experimentalmente são idênticos. O problema principal desta tecnologia é que quando o número de clientes duplica, a velocidade de transmissão passa para metade o que por sua vez significa que o tempo de transferência duplica também. Esta relação pode ser representada por:

*tempo total = tamanho do ficheiro * número de clientes / largura de banda no bottleneck*

## 4.2 Multicast (UDP)

Sendo que os mesmos dados irão ser transmitidos para vários computadores ao mesmo tempo, faz sentido analisar a tecnologia *multicast* baseada em UDP que opera segundo a teoria de que os dados são enviados apenas uma vez do servidor e recebidos por vários clientes simultaneamente (Deering, 1989).

Para tal foi usada a ferramenta open-source "udpcast" (Knaff, 2011) que é constituída por um programa que envia os dados para a rede "udp-sender" e outro para os receber "udp-receiver". Estes comandos estão preparados para serem integrados em *pipes*.

Antes de se proceder aos testes formais foram realizados alguns testes com vista a compreender o funcionamento da ferramenta, no entanto rapidamente se percebeu que no contexto em questão o desempenho era muito fraco apenas conseguindo transmissões sem erros a cerca de 256KBytes/s (ou 2Mbit/s).

O comando usado para o envio do lado do servidor foi:

```
udp-sender --file 1GB.xz --full-duplex --async --max-bitrate 2m
```

O comando para receber do lado do cliente foi:

```
udp-receiver | xz -dc | sha1sum
```

Como previamente mencionado, o cálculo do *hash* no final serviria para comparar com o *hash* dos dados originais, isto apesar de a própria ferramenta de receção informar imediatamente quando deteta que existem pacotes em falta.

Apesar de não se ter aprofundado o motivo de tão fraco desempenho, a explicação mais plausível tem a ver com o ambiente heterogéneo de equipamentos multi-marca de baixo custo (que possivelmente não implementam corretamente a distribuição de pacotes em *multicast* usando ao invés disso o *broadcast*, que congestiona muito mais a rede) e com o domínio de colisão relativamente grande com muitos níveis.

A ferramenta até disponibiliza opções de correção de erros e retransmissão de pacotes não recebidos, mas com valores obtidos tão baixos e visto que o propósito é usar a rede atual e não reestruturar, optou-se por deixar esta solução definitivamente de fora.

## 4.3 BitTorrent (TCP + UDP)

Tal como indicado na especificação oficial o "BitTorrent é um protocolo para distribuição de ficheiros" e "a sua vantagem em relação ao HTTP simples é que quando múltiplas transferências do mesmo ficheiro acontecem concorrentemente, os clientes enviam os dados uns para os outros, tornando possível que a origem do ficheiro suporte um grande número de clientes com apenas um pequeno aumento na sua carga" (Cohen, 2008, 2003). Um estudo pela empresa "Ipoque" apresenta o protocolo BitTorrent como o mais usado na Internet na quase totalidade das regiões do mundo (Schulze and Mochalski, 2009).

O protocolo BitTorrent já foi usado para distribuição de imagens *GNU/Linux* (Finley et al., 2007) bem como de máquinas virtuais para salas de aula com resultados satisfatórios (O'Donnell, 2008). É usado pela "Blizzard Entertainment" para transferência de jogos e atualizações (Blizzard Entertainment Inc., 2012). Vários estudos também apontam o seu uso para sistemas de *Video on Demand* (VoD) (Cheng et al., 2008; Savolainen and Raatikainen, 2008; Aalto, Lassila and Raatikainen, 2010).

A teoria geral de funcionamento do protocolo BitTorrent é explicada em (Cohen, 2003). Os clientes irão partilhar os dados entre si e é este aspeto que dá a eficiência ao algoritmo. Para tal, o ficheiro (ou ficheiros) original é dividido em partes denominadas "pieces". Existe um ficheiro "*torrent*" (que tipicamente possui a extensão ".torrent") que descreve o ficheiro e contém o *hash* de cada uma das *pieces* para além de outras informações. Existe um *software* servidor denominado *tracker* que lista que clientes (*peers*) é que possuem o ficheiro parcial ou totalmente (*seeders*). O endereço do *tracker* a utilizar também está incluída no ficheiro

"*torrent*". Assim um novo cliente vai abrir o ficheiro "*torrent*" e a partir daí irá perguntar ao *tracker* que *peers* possuem o ficheiro. De seguida irá contactar diretamente os *peers* por forma a pedir-lhes o envio de *pieces*. A partir do ponto que possua *pieces*, este novo cliente também será contactado por outros *peers* que lhe irão pedir *pieces* que este possua. Esta é uma explicação básica, pois o protocolo possui ainda mecanismos que lhe permitem, por exemplo, eliminar a dependência de um *tracker*, garantir que quem recebe dados também os partilha, etc.

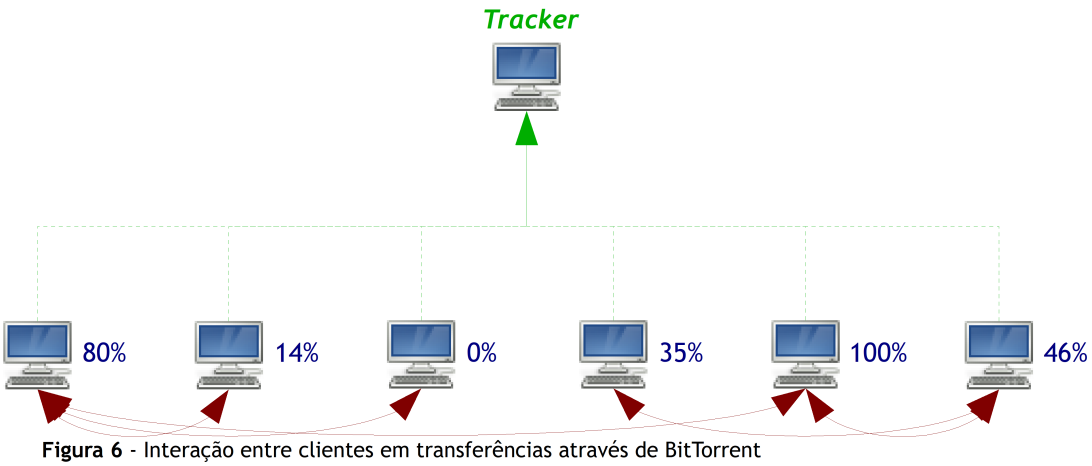



**Figura 6** - Interação entre clientes em transferências através de BitTorrent

Explicado o funcionamento do protocolo BitTorrent, esta é a altura para explicar porque se acredita que este protocolo poderá ter um bom desempenho na situação em estudo. Como explicado anteriormente (ver figura 1) o *bottleneck* da transferência dos dados encontra-se entre a sala de aula, onde se encontram todos os clientes, e o servidor. Sendo o BitTorrent um protocolo em que os dados são partilhados entre os clientes, significa que esta partilha não é afetada pelo *bottleneck* e que os dados poderão fluir livremente entre os clientes (dentro dos limites da capacidade do *hardware*). Por outro lado poderão existir entraves ao bom funcionamento do protocolo, como por exemplo, como será que se comporta devido ao facto de inicialmente apenas um cliente (o servidor) possuir os dados? Existem ainda problemas técnicos para os quais será necessário encontrar solução. Um aspeto importante e limitador deste protocolo é que a transferência dos dados não é sequencial e como tal é necessário reservar espaço no disco rígido para o ficheiro completo. O ficheiro irá sendo completado à medida que as *pieces* vão chegando. Esta situação é contrária ao ambiente encontrado num ambiente *GNU/Linux live distribution* que tipicamente é executado exclusivamente em memória onde o espaço de armazenamento temporário é escasso.

Relativamente à aplicação de BitTorrent a usar, considerando que neste projecto será necessária a integração de várias ferramentas independentes, o fator mais importante é a possibilidade de execução em segundo plano (sem interface gráfica). Neste sentido optou-se por usar o *software* "Transmission" (Transmission Project, 2012b) que já vem incluído nos

repositórios do GRML e possui uma biblioteca *python* denominada "python-transmissionrpc" (Svensson, 2011) que através de JSON-RPC permite gerir a transferência de *torrents*.

Para a função de *tracker* será usado o *software* "opentracker" na sua versão 1.233 que "pretende ter uma utilização mínima de recursos por forma a poder ser executado num *router* de cliente residencial" (Engling, n.d.). Deverá ser adequado para o pretendido até porque não se prevê muitos clientes em simultâneo. A função do *tracker* é bastante simples pelo que poderia ser utilizado qualquer outro *software* equivalente previsivelmente sem qualquer impacto no resultado final.

Os testes a realizar são mais complexos que os anteriores pelo que não é possível usar apenas um comando. Neste sentido foram implementados um conjunto de *scripts* em *bash shell* e *python* para coordenar a transferência dos dados e registar as medições.

Os testes iniciais do *software* "Transmission" revelaram alguma instabilidade sendo que por vezes deixava de transmitir dados apesar de aparentar estar a funcionar corretamente. Verificou-se que a versão usada não era a mais recente. Para usar a versão mais recente (2.22) do *software* foi necessário activar o repositório "experimental" do Debian. Após mais alguns testes o *software* já aparentava um funcionamento adequado e foi possível prosseguir.

Posteriormente também se verificou que há vários parâmetros que afetam o desempenho do protocolo BitTorrent e particularmente do "Transmission" usado neste caso. Como tal algumas das opções de configuração disponíveis foram ajustadas (Transmission Project, 2012a). Foram efetuadas várias tentativas até à obtenção de uma velocidade de transmissão considerada constante, mas que apenas foi verificada visualmente. Os parâmetros alterados foram:

- *Distributed Hash Table* (DHT) inativo – pois como indicado na especificação "cada *peer* converte-se num *tracker*" (Loewenstern, 2008). Este é um comportamento indesejado, pois pode permitir que os dados fiquem disponíveis fora do ambiente da rede local;
- *Local Peer Discovery* (LPD) ativo – pois esta extensão é "um método pelo qual o BitTorrent tenta descobrir novos *peers* locais relativamente à rede do computador" através do "uso de IP *multicast*" (BitTorrent Inc., 2012b). Neste cenário, devido às transferências se iniciarem todas ao mesmo tempo, o *tracker* irá apenas gradualmente ter conhecimento dos seus *peers*. Como o intervalo de refrescamento (*request interval*) (Cohen, 2008) no "opentracker" é de 1800 segundos significa que o primeiro *peer* a ligar-se ao *tracker* teria de esperar 30 minutos para ter conhecimento dos outros *peers*. O LPD irá mitigar este problema;
- *Peer Exchange* (PEX) ativo – pois esta extensão permite "trocar listas de *peers* com outros *peers*" (BitTorrent Inc., 2012c) ajudando a resolver o mesmo problema explicado no ponto anterior;
- 8 MB/s de máximo para *upload* (*apenas* nos clientes) – após alguns testes notou-se que a velocidade de *download* não era constante. Subia até à velocidade máxima e logo de seguida descia rápidamente para zero e depois novamente até ao máximo. Após alguma

pesquisa, pensou-se que poderia estar relacionado com um efeito chamado "*ACK starvation*" que é frequente quando se usa BitTorrent sobre ligações ADSL onde a largura de banda de *upload* é muito inferior à largura de banda de *download* (Lakshman, Madhow and Suter, 2000). Isto acontece porque existem tantos dados a ser enviados (*upload*) que não existe largura de banda para enviar o *ACK* dos dados recebidos (*download*) forçando retransmissões e penalizando a velocidade de *download* no geral. Para prevenir este problema o *upload* é limitado a cerca de 75% de 100Mbit/s como sugerido em (BitTorrent Inc., 2012d);

- 8 *upload slots per torrent* (apenas no servidor) – após alguns testes também se verificou que a maioria dos clientes tentava transferir os dados a partir do servidor. Este não é o comportamento desejado porque o tráfego para o servidor tem de passar pelo *bottleneck*. Então para prevenir os clientes de pedirem sempre os dados ao servidor e depois esperar que exista largura de banda disponível para o envio dos dados, é melhor o servidor indicar logo à partida que não irá aceitar mais pedidos e forçar os clientes a ir procurar os dados junto dos outros clientes.

A especificação do protocolo BitTorrent bem como as suas implementações continuam a evoluir pelo que estes valores apenas devem ser considerados válidos para o cliente BitTorrent em questão e na versão indicada.

### 4.3.1 Apenas um *inicial seeder*

O primeiro teste que foi realizado foi o teste com apenas um cliente inicial (*initial seeder*) que neste caso é o próprio servidor de imagens. Este teste irá permitir determinar se o protocolo BitTorrent tem um bom desempenho neste cenário partindo de condições que são particularmente adversas para ele.

Com vista a realizar este teste o ficheiro da imagem será transferido para o disco do computador e posteriormente será descomprimido, guardando o tempo total destas duas operações.

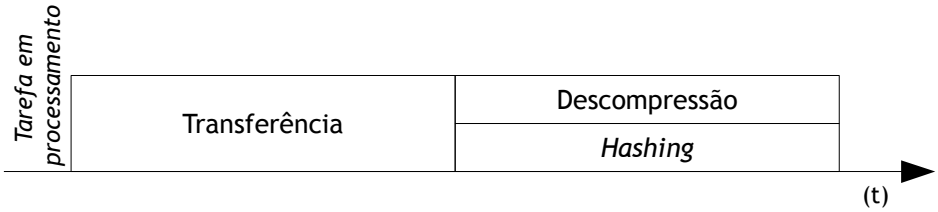



**Figura 7** - Paralelismo de execução de tarefas no teste de transferência por BitTorrent com apenas um initial seeder

Daqui se conclui que o tempo total da operação pode ser representado pela fórmula:

$$tempo\ total = tempo\ transferência + tempo\ descompressão$$

Ao realizar este teste detetou-se um problema em que, ao iniciar a segunda execução do teste, o cliente fica à espera dos dados mas não os recebe. Tal só acontece após muito tempo de espera. Apesar de não se ter investigado exaustivamente este problema, concluiu-se que provavelmente está relacionado com o facto de ao terminar a transferência, o cliente notificar o *tracker* de que agora possui o ficheiro e assim todos os outros clientes ficam a pensar que este cliente já possui o ficheiro não enviando novamente os dados mesmo que este os solicite. Isto levou novamente a uma reestruturação dos testes. Ao passo que inicialmente eram executados 5 transferências consecutivas do ficheiro de 1GB depois do de 5GB, etc, agora são executadas 5 transferências sequenciais dos ficheiros de 1, 5, 10 e 20GB. No final de cada execução todos os clientes de *torrent* são reiniciados por forma a perderem a sua *cache* e começarem sempre limpos.

Este teste originou os seguintes resultados:

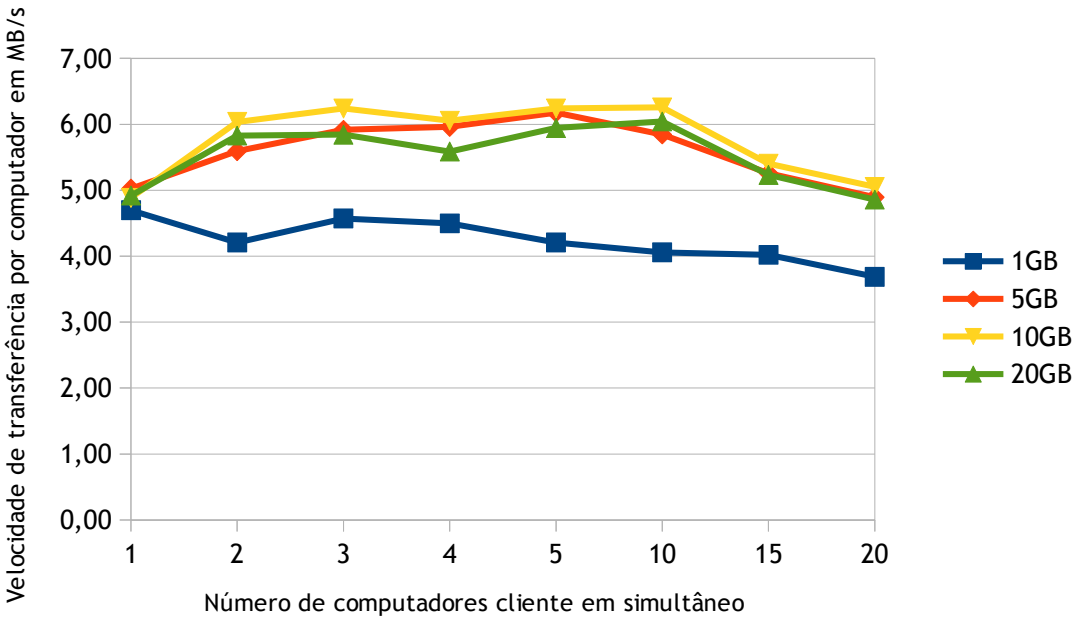



**Figura 8** - Resultados experimentalmente obtidos para a transferência por BitTorrent com apenas um initial seeder

| Número de computadores cliente em simultâneo | Tamanho do ficheiro de teste 1GB | 5GB | 10GB | 20GB |
|---|---|---|---|---|
| 1 | 4,70 | 5,02 | 4,88 | 4,92 |
| 2 | 4,21 | 5,59 | 6,04 | 5,83 |
| 3 | 4,57 | 5,92 | 6,24 | 5,84 |
| 4 | 4,50 | 5,96 | 6,05 | 5,59 |
| 5 | 4,21 | 6,18 | 6,24 | 5,94 |
| 10 | 4,06 | 5,84 | 6,26 | 6,04 |
| 15 | 4,02 | 5,26 | 5,40 | 5,23 |
| 20 | 3,69 | 4,89 | 5,05 | 4,86 |

**Tabela 8** - Resultados experimentalmente obtidos para a transferência por BitTorrent com apenas um initial seeder (em MB/s)

Para uma melhor comparação, apresenta-se agora o mesmo gráfico adicionando os resultados obtidos anteriormente para o HTTP.

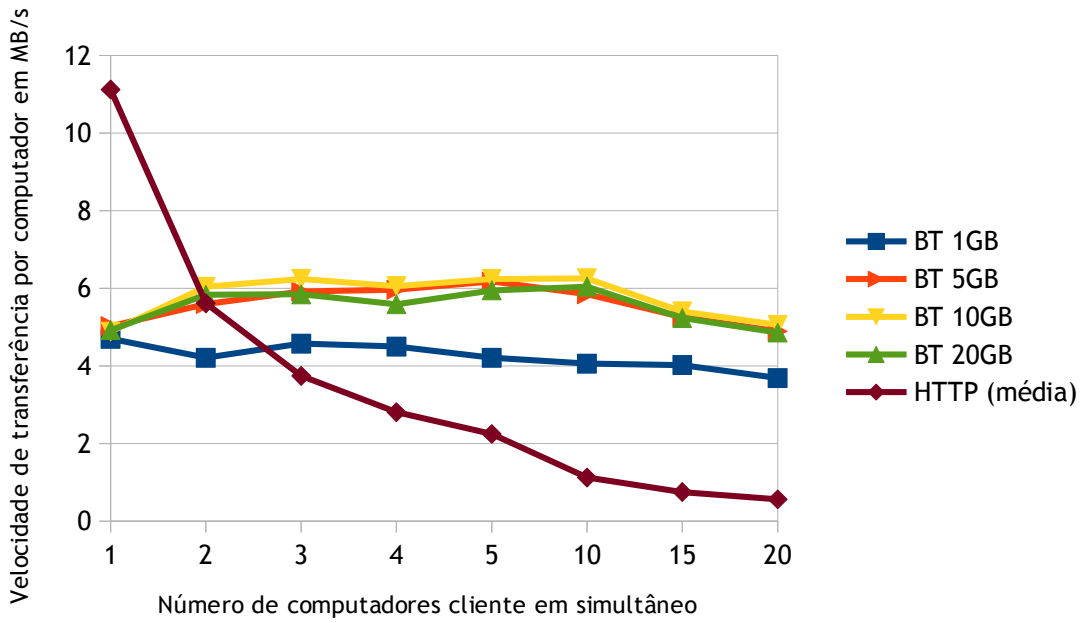



**Figura 9** - Comparação dos resultados obtidos para a transferências por BitTorrent e HTTP

Apesar desta técnica não poder ser usada na realidade devido à necessidade de guardar o ficheiro da imagem temporariamente no disco rígido, é possível ter uma noção clara das capacidades do protocolo neste cenário e apenas com um cliente inicial (*initial seeder*).

Também se nota que no caso do BitTorrent com um ficheiro de 1GB a transferência foi mais lenta, isto porque existe um tempo inicial de negociação que, devido ao ficheiro ser relativamente pequeno e o tempo de transmissão curto, representa uma percentagem elevada do tempo total.

O BitTorrent supera claramente o HTTP para três ou mais clientes, iguala o HTTP para 2 clientes e apenas perde no caso em que um único cliente transfere os dados.

Também se nota que o desempenho do BitTorrent começa a cair a partir de 10 computadores em simultâneo. Tal provavelmente deve-se ao *overhead* do protocolo ou ao facto de o *switch* da sala de aula atingir a sua capacidade máxima.

### 4.3.2 Ficheiro divido em blocos

Mesmo sem olhar para os resultados, o teste anterior deixou claramente espaço para uma melhoria pois a transferência e a descompressão ocorrem sequencialmente. Um ganho óbvio seria executar em paralelo estes passos fazendo com que a transferência e a descompressão ocorram simultaneamente (o que aliás já acontece com o atual sistema em HTTP). Tendo em conta que os atuais computadores das salas de aula são todos *multi-core* o impacto seria mínimo.

Para avançar para esta solução é necessário resolver um problema. Se a transferência dos dados em BitTorrent não é sequencial, então como fazer para os dados chegarem sequencialmente para serem alimentados ao descompressor? A solução encontrada para resolver este problema consiste em dividir o ficheiro de imagem em blocos (neste caso de 640MB) e efetuar um *download* sequencial destes blocos, ou seja ao invés de um ficheiro *torrent* teríamos vários ficheiros *torrent* um para cada bloco (não confundir estes blocos com as *pieces* do BitTorrent). Assim o *software* de BitTorrent poderia efetuar a transferência do bloco seguinte enquanto se descomprime o bloco atual, tal como apresentado no diagrama seguinte.

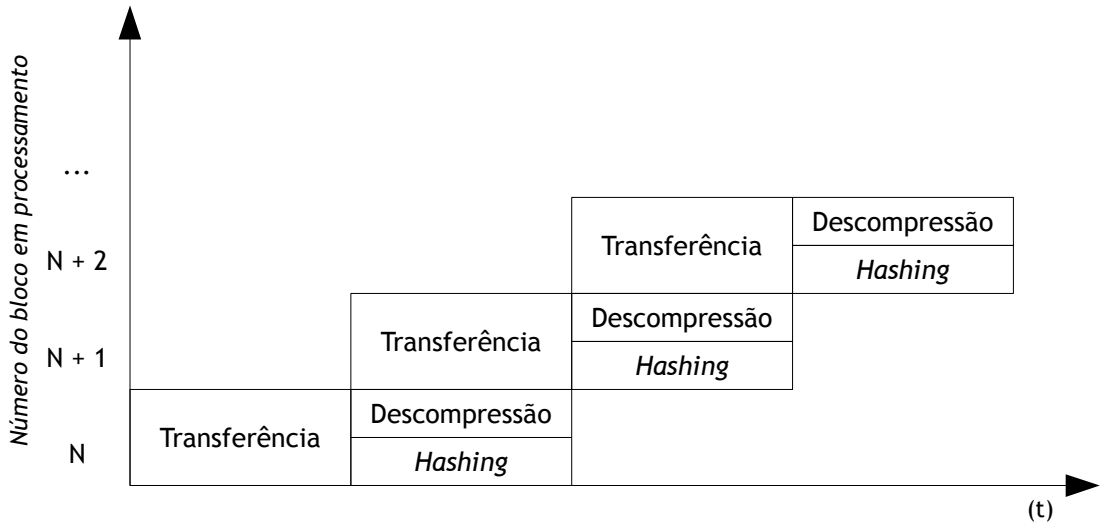



**Figura 10** - Paralelismo de execução de tarefas no teste de transferência por BitTorrent com ficheiro dividido em blocos

Neste caso o tempo total da operação poderia ser representada por:

*tempo total = tempo transferência de todos os blocos + tempo descompressão do último bloco*

Então coloca-se a questão de qual o tamanho de bloco mais adequado. Não deve ser demasiado pequeno pois a transferência de cada novo *torrent* tem custos de ligação tais como comunicar com o *tracker*, com os clientes, etc. Se for demasiado grande será necessário espaço temporário em disco para o armazenar e, como previamente indicado, o espaço em disco não poderá ser usado pois irá ser completamente reescrito. Então o mais sensato é escolher um tamanho de bloco tendo em conta a memória dos computadores atuais e considerando que no futuro a tendência será para que cada vez este valor seja superior. Assim, considerando um computador com 2GB de memória RAM (2048MB) teríamos:

*2048 (memória total do computador) - 768 (GRML + transmission + xz) = 1280 (livres)*
*1280 / 2 (blocos, um que está a ser transferido e outro a ser descomprimido) = 640 (cada bloco)*

De mencionar que inicialmente o tamanho previsto para o bloco era de 768MB, no entanto verificou-se que esporadicamente ocorriam erros de falta de memória pelo que foi necessário aumentar a memória disponível para o sistema operativo e aplicações.

Considerando que o teste anterior providenciou valores de referência com pouca variação à medida que o número de clientes aumenta e que o acesso ao ambiente de testes é limitado, optou-se por fazer os testes seguintes apenas para 15 computadores que é o mais próximo do valor médio de computadores das salas da ESTCB.

Este teste produziu os seguintes resultados:

| | Tamanho do ficheiro de teste | | | |
|---|---|---|---|---|
| | 1GB | 5GB | 10GB | 20GB |
| Velocidade de transferência por computador para 15 computadores em simultâneo em MB/s | 3,21 | 2,60 | 3,00 | 3,35 |

**Tabela 9** - Resultados experimentalmente obtidos para a transferência por BitTorrent com o ficheiro dividido em blocos

### 4.3.3 Ficheiro divido em blocos e *webseed*

Um aspeto interessante que foi verificado aquando da realização dos testes, foi que há um efeito de convergência, ou seja quando os clientes não são inicializados todos exatamente ao mesmo tempo, o que se verifica é que o último cliente a iniciar a transferência, ao fim de algum tempo, atinge o ponto de transferência em que os outros clientes estão. Este efeito é fácil de justificar tendo em conta o funcionamento do protocolo BitTorrent. Se uma determinada *piece* já existe em muitos clientes é mais fácil obtê-la do que quando ela apenas existe no servidor. Considerando que neste cenário todos os clientes que estão atrás do *bottleneck* têm muita

largura de banda disponível entre eles, esta partilha é rápida, ao passo que a ligação ao servidor é mais lenta.

Apesar de haver partilha entre os vários clientes, nos testes anteriores todos os clientes competiam pelos recursos do servidor, pelo que uma melhoria que se poderia fazer era tentar que um dos clientes obtivesse mais rapidamente os dados do servidor por forma a disponibilizá-los mais rapidamente atrás do *bottleneck*. Assim tirar-se-ia partido do efeito de convergência identificado.

Por forma a explorar esta solução foi usada uma extensão do protocolo BitTorrent denominada *webseed*. Esta permite a um servidor HTTP ser mais um *seeder* numa rede BitTorrent (BitTorrent Inc., 2012a). No ficheiro *torrent* é incluído um endereço HTTP onde se poderão descarregar os dados. Esta extensão é suportada nativamente pelo cliente "Transmission" usado nos testes anteriores pelo que as alterações são mínimas. Assim o primeiro cliente faria *download* por BitTorrent (normal) e *webseed* (HTTP) ao passo que os outros apenas fariam *download* por BitTorrent (normal).

Este teste produziu os seguintes resultados:

| | Tamanho do ficheiro de teste | | | |
|---|---|---|---|---|
| | 1GB | 5GB | 10GB | 20GB |
| **Velocidade de transferência por computador para 15 computadores em simultâneo em MB/s** | | | 3,89 | |

**Tabela 10** - Resultados experimentalmente obtidos para a transferência por BitTorrent com o ficheiro dividido em blocos e webseed

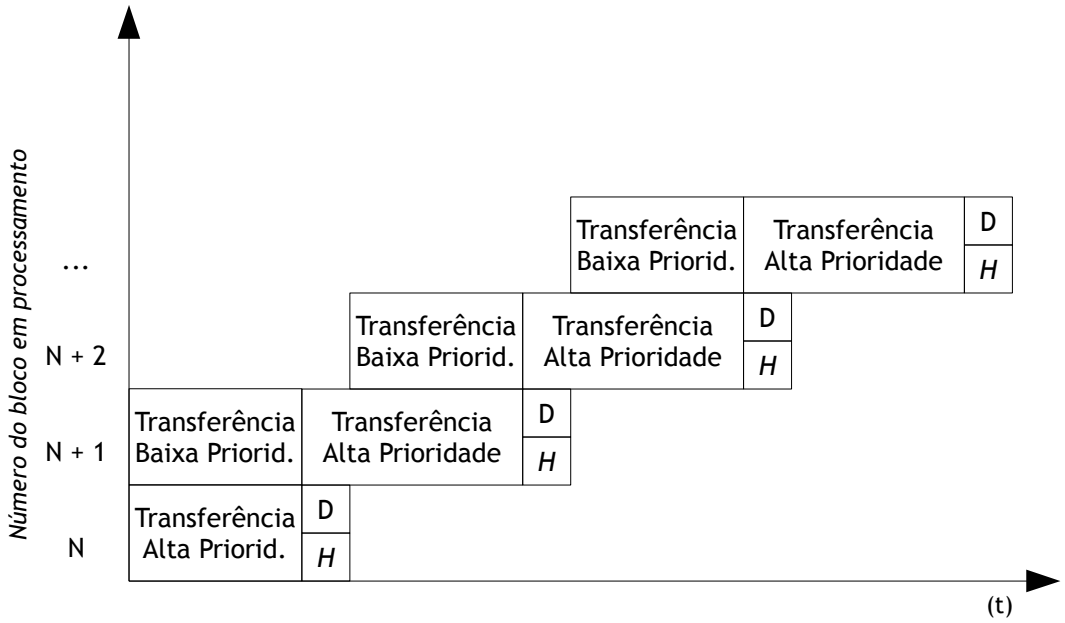
Verifica-se que o conceito do *webseed* funciona como previsto, no entanto imediatamente foi identificado um problema. A transferência por HTTP origina muitos pedidos pois o cliente faz um pedido parcial (*partial GET* (Fielding et al., 1999)) por cada *piece* do ficheiro. Isto origina uma grande carga no servidor e na rede.

### 4.3.4 Ficheiro divido em blocos com transferências simultâneas

Um problema que se identificou nos dois testes anteriores foi o tempo que decorre até ao início da transferência em BitTorrent. Existe um espaço de tempo de vários segundos em que o cliente de BitTorrent aparenta não estar a fazer nada, pois apesar de o ficheiro já estar na lista para transferência, não estão a ser transferidos dados. Na realidade, provavelmente o cliente está à procura dos outros clientes que possuem o ficheiro para depois poder pedir os dados. Quando a transferência é de apenas um ficheiro, este tempo é negligenciável, no entanto como neste sistema são transferidos vários ficheiros (os blocos) a soma destes tempos é significativa.

Para diluir este tempo de espera, a solução passa por paralelizar mais o processo. Enquanto o sistema está a descomprimir o bloco n, está a transferir o bloco n+1 à velocidade

máxima. Quando a descompressão termina, elimina da memória o bloco n e coloca já a transferir o bloco n+2 em baixa velocidade. Assim, enquanto o bloco n+1 está a descarregar à velocidade máxima, está a decorrer o tempo de espera e eventualmente começa a transferência do bloco n+2. Como esta transferência é com baixa prioridade não interfere na transferência do bloco n+1 por forma a manter coerência na ordem de chegada dos blocos. Esta explicação pode ser visualizada no diagrama seguinte.




* D – Descompressão, H – Hashing

**Figura 11** - Paralelismo de execução de tarefas no teste de transferência por BitTorrent com ficheiro dividido em blocos e transferências simultâneas

Neste caso o tempo total da operação poderia ser representado por:

*tempo total = tempo transferência de todos os blocos + tempo descompressão do último bloco*

Este teste teve os seguintes resultados:

| | Tamanho do ficheiro de teste | | | |
|---|---|---|---|---|
| | 1GB | 5GB | 10GB | 20GB |
| **Velocidade de transferência por computador para 15 computadores em simultâneo em MB/s** | 3,94 | 4,60 | 4,61 | 4,73 |

**Tabela 11** - Resultados experimentalmente obtidos para a transferência por BitTorrent com o ficheiro dividido em blocos e transferências simultâneas

## 4.4 Conclusões

Colocando os resultados (15 clientes x 10GB) das diversas tecnologias no mesmo gráfico comparativo, obtém-se:

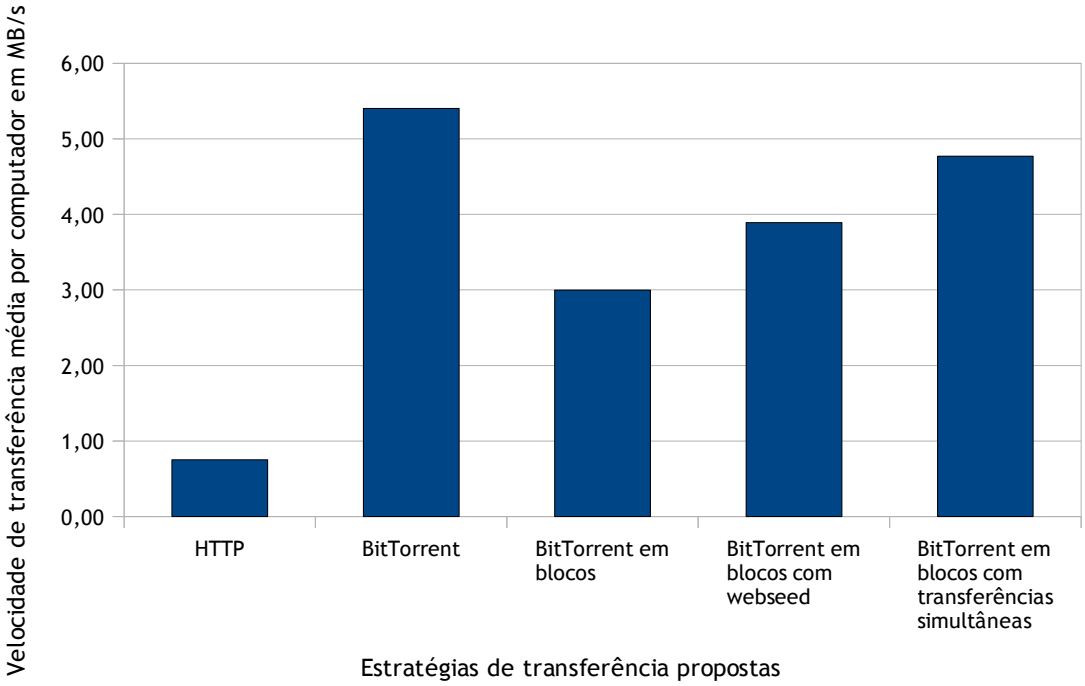



**Figura 12** - Comparação dos resultados obtidos para as diversas estratégias de transferência

Considerando estes resultados pode-se concluir que: o protocolo BitTorrent é largamente mais eficiente que o HTTP. Previsivelmente, a única exceção é quando apenas um cliente está a transferir os dados (ver a anterior figura 9). Como previsto, o processamento adicional da negociação da transferência dos blocos fez com que o BitTorrent perdesse alguma da sua eficiência, neste caso cerca de metade. Esta melhorou com a técnica baseada em *webseed* mas a técnica que revela um desempenho mais próximo do BitTorrent simples é a BitTorrent por blocos com transferências simultâneas.

Expandindo a solução final para todos os tamanhos e considerando o BitTorrent "simples" como valor de referência (100%) temos:

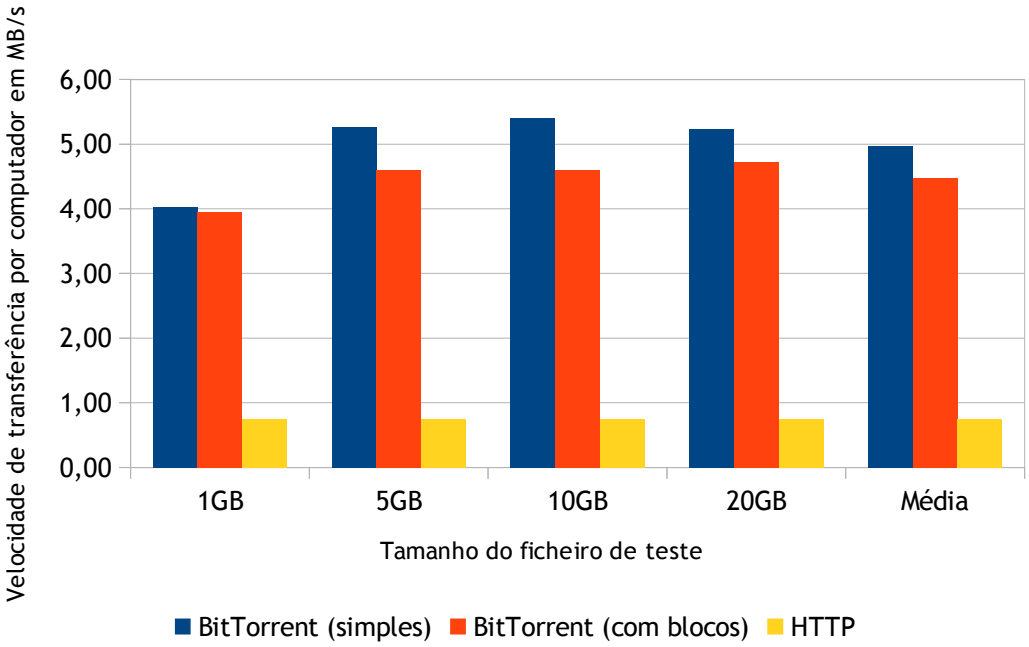



**Figura 13** - Comparação do desempenho da solução final

| | | Estratégias de transferência | | |
|---|---|---|---|---|
| | | BitTorrent (simples) | BitTorrent (com blocos) | HTTP |
| Tamanho do ficheiro de teste | 1GB | 100,00 | 98,07 | 18,60 |
| | 5GB | 100,00 | 87,53 | 14,22 |
| | 10GB | 100,00 | 85,28 | 13,85 |
| | 20GB | 100,00 | 90,38 | 14,30 |
| | *Média* | *100,00* | *90,32* | *15,24* |

**Tabela 12** - Comparação do desempenho da solução final (em %)

A solução BitTorrent por blocos com transferências simultâneas mantém a sua coerência com os restantes ficheiros de teste, atingindo um desempenho de cerca de 90% do BitTorrent simples e ficando largamente acima do HTTP que neste caso em concreto com 15 clientes se ficou pelos 15%.

Após os resultados dos testes anteriores, os requisitos da aplicação a conceber podem ser bastante simplificados. Visto que o UDP não pode de todo ser usado e que o HTTP não é vantajoso para transferências em simultâneo, a aplicação não tem de se preocupar a selecionar a melhor estratégia de transferência. Apenas existem duas opções: ou a imagem é apenas para um computador e é transferida por HTTP ou é para mais do que um e é transferida por BitTorrent.

# 5. Desenvolvimento

Neste capítulo será abordado o desenvolvimento do *software* própriamente dito. Considerando que nesta fase se possui toda a informação relevante sobre o problema e as suas soluções técnicas, optou-se pela utilização de um modelo de desenvolvimento de *software* em cascata (Pfleeger, 2004) passando pelas fases de definição de requisitos, especificação/projeto, implementação e testes.

## 5.1 Requisitos definitivos

Considerando que o público alvo desta aplicação são os Administradores de Sistemas com conhecimentos de redes e do sistema operativo *GNU/Linux*, e no seguimento dos resultados dos testes realizados, é agora possível definir mais concretamente quais os requisitos finais que o sistema deve ter.

- Permitir enviar imagens para o servidor;
- Preparar imagens para estarem acessíveis por BitTorrent (no servidor);
- Permitir gerir as imagens no servidor (as operações são demasiado simples para justificarem uma interface WEB);
- Permitir receber imagens do servidor (por HTTP e por BitTorrent);
- A partir do endereço IP do cliente detetar automaticamente a imagem a transferir;
- O *software* cliente deverá executar no ambiente de linha de comandos, recebendo parâmetros que permitem escolher a estratégia de receção da imagem;
- Criar os automatismos adicionais que permitem arrancar automaticamente o ambiente *GNU/Linux live distribution*, transferir a ferramenta de imagens e preparar todas as suas dependências.

## 5.2 Linguagens de programação

### 5.2.1 Python

"Python é uma poderosa linguagem programação dinâmica que é usada numa grande variedade de domínios aplicacionais. O Python é muitas vezes comparado ao Tcl, Perl, Ruby, Scheme ou Java." (Python Software Foundation, 2012).

Esta foi escolhida para este projeto pois, como é usada em vários aspetos base do sistema operativo *GNU/Linux*, já vem incluído de raiz no *GNU/Linux live distribution* que vamos usar e provavelmente em qualquer outro que se pretenda usar.

## 5.3 Tecnologias

### 5.3.1 HTTP

Como anteriormente explicado na secção 4.1 o HTTP é o protocolo usado na WWW. Ficou demonstrado que é um protocolo muito eficiente na transferência ponto a ponto ficando muito próximo dos resultados máximos teóricos possíveis.

Neste projeto ele é usado para transferir informação prévia à transferência da imagem e, caso o BitTorrent não seja usado, para transferir o próprio ficheiro ou ficheiros da imagem. Caso o sistema operativo esteja a ser executado a partir da rede também será usado para transferir os ficheiros necessários ao funcionamento deste.

Para receber os pedidos HTTP do lado do servidor é usado o conhecido *Apache HTTP Server* que, segundo o levantamento de Abril de 2012 da Netcraft, é o servidor web mais utilizado na Internet há já mais de 10 anos (Netcraft Ltd, 2012).

De salientar que neste projeto se fez questão de não usar linguagens dinâmicas (PHP, CGI, etc) do lado do servidor. Todos os ficheiros a aceder são previamente gerados e estáticos. Isto permite tornar o servidor mais seguro (The Apache Software Foundation, 2012a) e simples de manter.

Como também já mencionado na fase de testes, para que o protocolo HTTP garanta alguma justiça na atribuição de largura de banda aos vários clientes é usado um algoritmo de *stochastic fairness queuing* na interface de rede do servidor.

### 5.3.2 FTP

Ao contrário do restauro de imagens que é feito para vários computadores de uma só vez, a salvaguarda de imagens é feita de apenas um computador. Neste sentido o problema da concorrência de ligações não se coloca e como tal foi usado o bem estabelecido protocolo FTP (Postel and Reynolds, 1985).

A sua implementação do lado do servidor está a cargo do *vsftpd* (Evans, 2012). Este é descrito pelo autor como “provavelmente o servidor FTP mais seguro e rápido para sistemas tipo UNIX”. Este foi escolhido pela sua configuração simples e por permitir limitar o acesso de um utilizador apenas à sua pasta pessoal (*home*) escondendo toda a restante estrutura de pastas do servidor.

### 5.3.3 BitTorrent

Tal como já foi apurado na fase de testes, o protocolo BitTorrent será implementado através do *software* “Transmission” (Transmission Project, 2012b). No caso do servidor, é ainda usado o *tracker* “opentracker” (Engling, n.d.). No cliente a interligação do *python* com este

*software* é feita através da biblioteca "*python-transmissionrpc*" (Svensson, 2011).

## 5.4 Modulação UML

Considerando a simplicidade da aplicação a desenvolver, para a fase de modulação usou-se uma abordagem ICONIX simplificada. Esta foi escolhida pois a ideia principal do ICONIX é "como modular o menos possível, no mais curto período de tempo, de forma a concretizar um bom sistema. O ICONIX não privilegia explicitamente a utilização de vários diagramas UML, em particular não privilegia os diagramas de estado, de actividade, de arquitectura e mesmo os diagramas de colaboração. Tal posição parece-nos adequada num processo que se pretende prático e simples" (Silva and Videira, 2001).

### 5.4.1 Diagrama de casos de uso

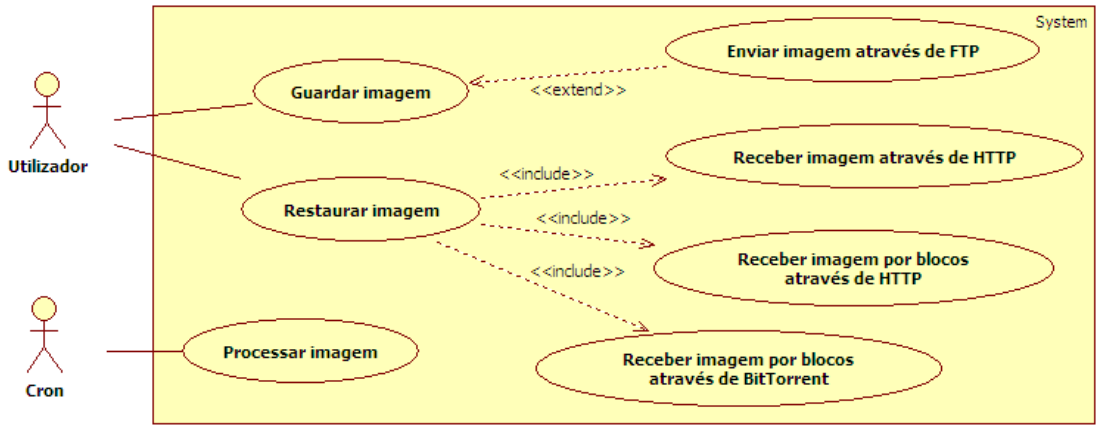



**Figura 14** - Diagrama de casos de uso

**Descrição dos atores**

- O ator "Utilizador" representa o utilizador que irá usar o sistema do lado do cliente.
- O ator "Cron" representa um automatismo do lado do servidor que deverá iniciar o caso de uso "Processar imagem" periodicamente (de hora a hora) de forma automática.

**Descrição dos casos de uso**

*Guardar imagem*

- Pré-requisitos: O "utilizador" indicou um disco ou partição válidos do sistema.
- Descrição: Este caso de uso irá criar uma imagem de um disco completo ou de uma partição. Os dados serão lidos do disco, comprimidos com um algoritmo de compressão rápida e enviados para o servidor através do protocolo FTP.
- Pós-requisitos: Existe uma imagem comprimida do disco/partição no servidor.

*Processar imagem*

- Pré-requisitos: Existem uma ou mais imagens de disco não processadas no servidor.
- Descrição: Este caso de uso irá recomprimir imagens recebidas dos clientes para uma compressão definitiva (o máximo possível) e gerar os blocos e ficheiros *torrent* necessários para que a imagem possa passar a ser transferida também por BitTorrent.
- Pós-requisitos: Existem imagens comprimidas ao máximo no servidor e que podem ser transferidas por HTTP e por BitTorrent.

*Restaurar imagem*

- Pré-requisitos: Existem uma ou mais imagens de disco adequadas a este computador cliente acessíveis no servidor.
- Descrição: Este caso de uso irá restaurar uma imagem existente no servidor para um disco rígido local do computador cliente. Os dados serão transferidos do servidor, descomprimidos e escritos no disco. A transferência dos dados poderá ser de uma de 3 formas dependendo de limitações do cliente ou servidor:
    - Caso os dados no servidor ainda não tenham sido processados (através do caso de uso "processar imagem") - a transferência será por HTTP.
    - Caso os dados no servidor já tenham sido processados (implica que os blocos estejam criados), se:
        - o cliente não tem o *software* de BitTorrent disponível (não está instalado, falta de memória, etc.) - a transferência será por HTTP mas bloco a bloco.
        - o cliente tem o *software* de BitTorrent disponível – a transferência será por BitTorrent.
- Pós-requisitos: O computador cliente possui a imagem restaurada no seu disco rígido.

*Enviar imagem através de FTP*

- Descrição: Este caso de uso irá auxiliar o caso de uso "guardar imagem" na parte do envio dos dados para o servidor através do protocolo FTP.

*Receber imagem através de HTTP*

- Descrição: Este caso de uso irá auxiliar o caso de uso "restaurar imagem" na parte da receção dos dados do servidor através do protocolo HTTP. Este é o método mais simples de transferência e é usado nos ficheiros pequenos ou caso o ficheiro ainda não tenha sido convertido em blocos.

*Receber imagem por blocos através de HTTP*

- Descrição: Este caso de uso irá auxiliar o caso de uso "restaurar imagem" na parte da

receção dos dados do servidor através do protocolo HTTP. Este método é usado caso a transferência seja para apenas um computador cliente, ou caso o ficheiro já tenha sido convertido em blocos mas não seja possível usar o protocolo BitTorrent do lado do cliente.

*Receber imagem por blocos através de BitTorrent*

- Descrição: Este caso de uso irá auxiliar o caso de uso "restaurar imagem" na parte da receção dos dados do servidor através do protocolo BitTorrent. Este método é o preferido sempre que esteja disponível e a transferência seja para mais do que um computador cliente.

### 5.4.2 Diagrama de componentes

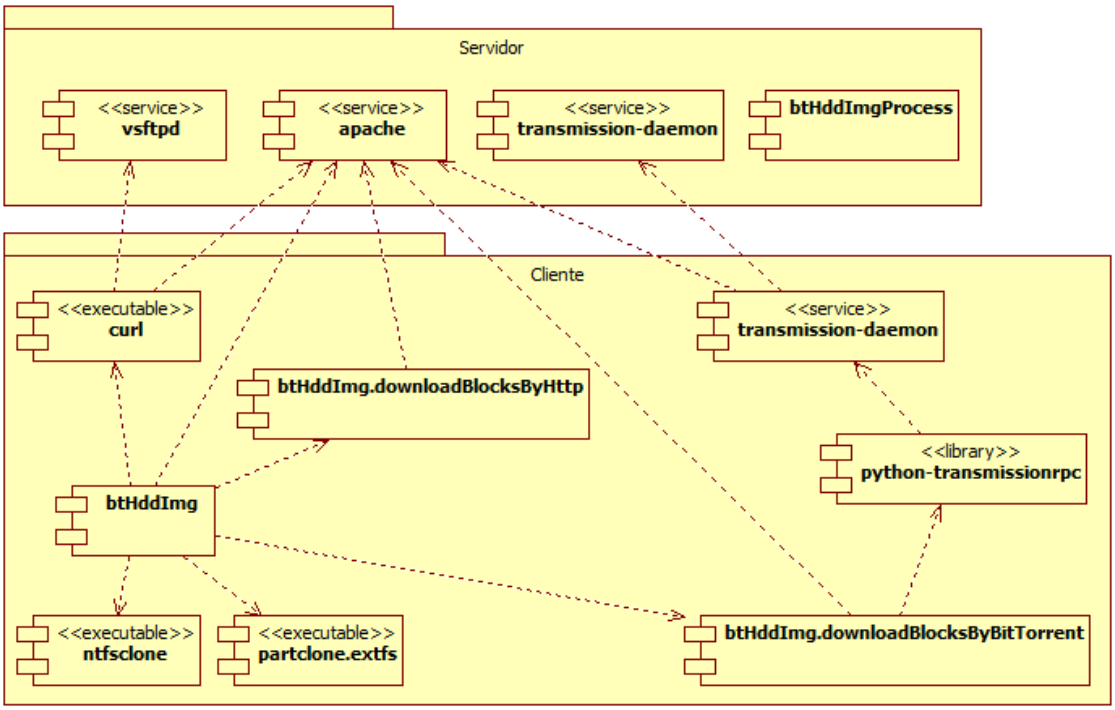



**Figura 15** - Diagrama de componentes

**Descrição dos componentes**

- *vsftpd* – Como mencionado anteriormente este será o *software* no servidor para comunicação através de FTP.
- *apache* – Como mencionado anteriormente este será o *software* que irá receber os pedidos HTTP no lado do servidor.
- *transmission-daemon* – Como mencionado anteriormente o *software* Transmission (Transmission Project, 2012b) será usado para a comunicação BitTorrent. O

"transmission-daemon" é o serviço que fica em execução e gere todo o processo de transferências.

- *btHddImgProcess* – A implementar (ver diagramas de classes)
- *curl* – segundo o sítio oficial "curl é uma ferramenta de linha de comandos para transferência de dados com uma sintaxe de URL, e que suporta DICT, FILE, FTP, FTPS, Gopher, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMTP, SMTPS, Telnet e TFTP" (Stenberg, 2012). Neste *software* é usado sempre que se pretende fazer uma transferência "normal" por HTTP, ou seja, não uma transferência por blocos. Este será integrado em *pipes* juntamente com outras ferramentas.
- *btHddImg* – A implementar (ver diagramas de classes)
- *ntfsclone* – é um utilitário pertencente ao pacote "ntfs-3g" que permite guardar e restaurar um sistema de ficheiros NTFS (Tuxera, 2012). É usado neste *software* para a guardar e restaurar os dados de partições do sistema operativo Microsoft Windows. Este será integrado em *pipes* juntamente com outras ferramentas.
- *partclone.extfs* – é um utilitário que permite guardar e restaurar os blocos usados de uma partição suportando vários formatos destas (Tsai, 2012). O seu componente com suporte a ext2, ext3 e ext4 é usado para guardar e restaurar os dados de partições do sistema operativo GNU/Linux. Este será integrado em *pipes* juntamente com outras ferramentas.
- *btHddImg.downloadBlocksByHttp* – A implementar (ver secção 5.4.6). Será integrado em *pipes* juntamente com outras ferramentas.
- *btHddImg.downloadBlocksByBitTorrent* – A implementar (ver secção 5.4.6). Será integrado em *pipes* juntamente com outras ferramentas.
- *python-transmissionrpc* – é um "módulo que permite usar o *python* para ligar ao serviço JSON-RPC do *software* Transmission" (Svensson, 2011). Será usado para gerir as transferencias dos blocos por BitTorrent.

### 5.4.3 Diagrama de instalação

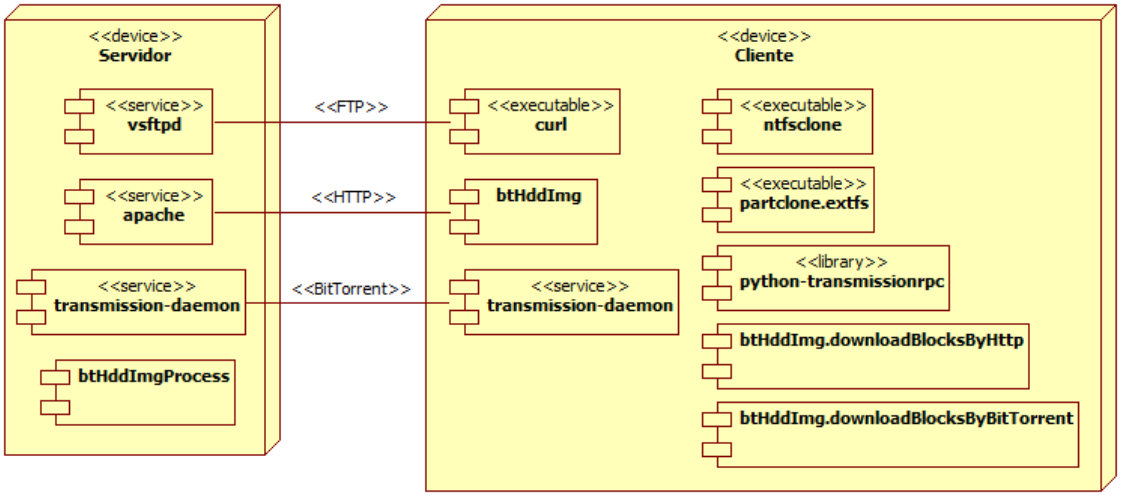



**Figura 16** - Diagrama de instalação

Este diagrama apresenta onde será instalado cada um dos componentes anteriormente especificados no diagrama de componentes.

### 5.4.4 Diagrama de classes

**Componente btHddImg**

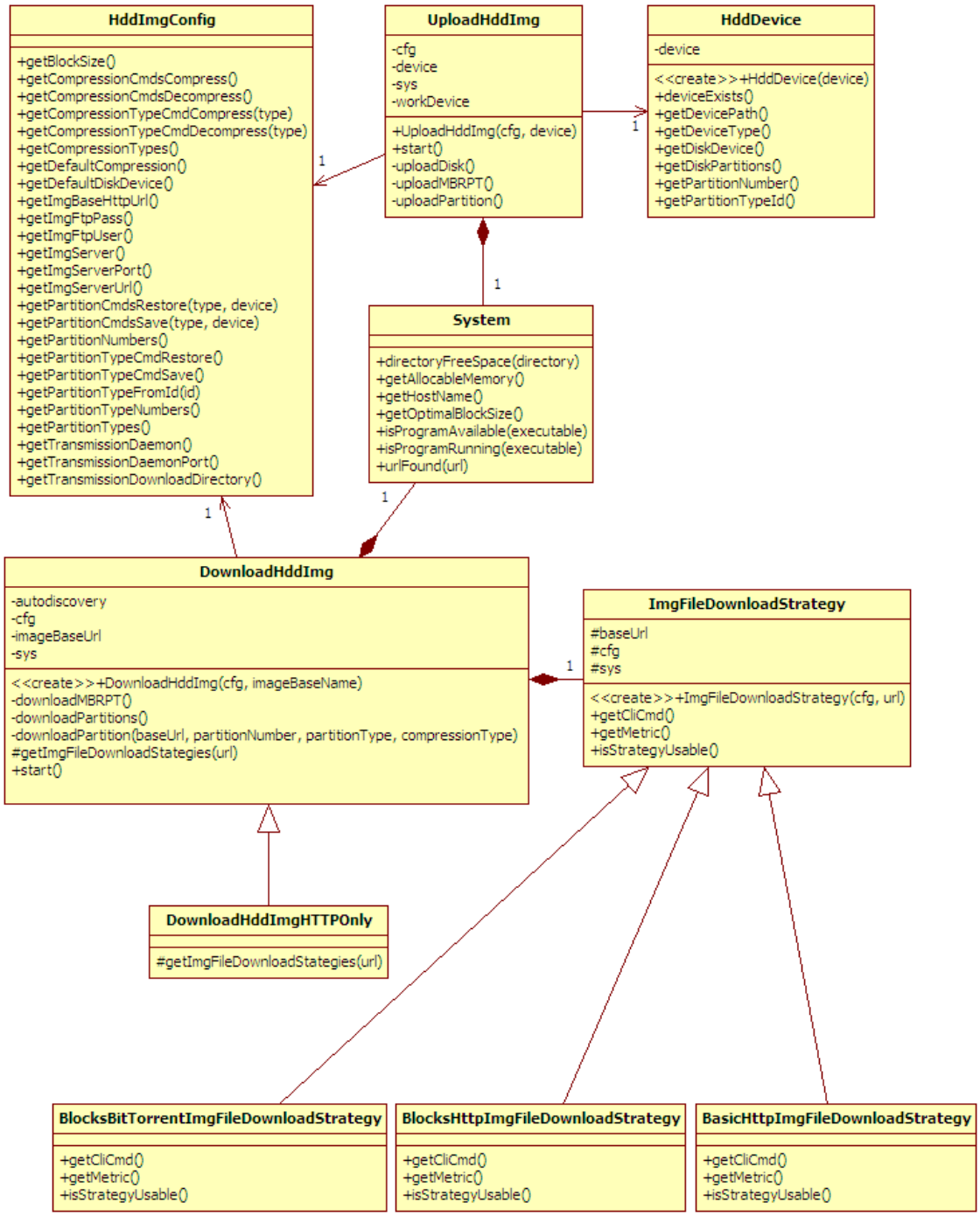



**Figura 17** - Diagrama de classes do componente btHddImg

**Descrição das classes**

- *HddImgConfig* – classe responsável por configurações do sistema tal como endereço do servidor, tipos de compressão, ferramentas de compressão, ferramentas de acesso a partições, etc. Estes dados devem ser configurados aquando da instalação do *software* para reflectir a realidade da organização.
- *System* – classe responsável por certas chamadas especificas ao sistema operativo. Serve como camada de abstração para uma futura migração da ferramenta para outros sistemas operativos.
- *HddDevice* – classe que representa um disco rígido ou partição de um disco rígido.
- *UploadHddImg* – classe responsável por gerir a execução das ferramentas que irão ler os dados do disco/partição e enviá-los para o servidor.
- *DownloadHddImg* – classe responsável por gerir a execução das ferramentas que irão transferir a imagem e restaurar os dados no disco/partição.
- *DownloadHddImgHttpOnly* – subclasse da anterior, idêntica mas que apenas irá usar o protocolo HTTP.
- *ImgFileDownloadStrategy* – classe que representa uma estratégia de *download*. Tem métodos para validar se uma estratégia pode ser usada e um sistema de métricas para indicar estratégias mais eficientes que outras. Pode ser extendida com estratégias adicionais.
- *BlocksBitTorrentImgFileDownloadStrategy* – subclasse responsável pela estratégia de *download* através de blocos com o protocolo BitTorrent.
- *BlocksHttpImgFileDownloadStrategy* – subclasse responsável pela estratégia de *download* através de blocos com o protocolo HTTP.
- *BasicHttpImgFileDownloadStrategy* – subclasse responsável pela estratégia de *download* simples através HTTP.

**Componente btHddImgProcess**

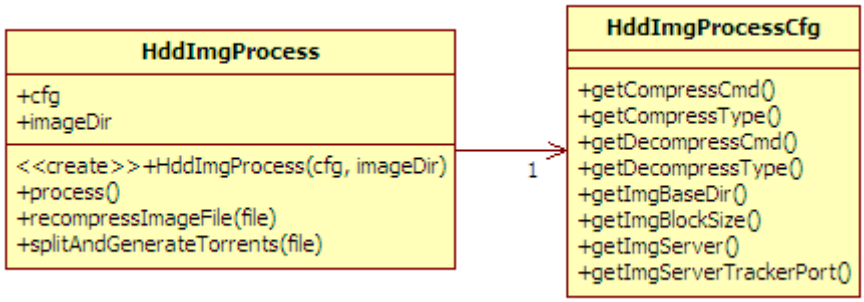



**Figura 18** - Diagrama de classes do componente btHddImgProcess

**Descrição das classes**

- *HddImgProcessCfg* – classe responsável por configurações do sistema tal como a localização das imagens, descompressor, compressor, etc. Estes dados devem ser configurados aquando da instalação do *software* para reflectir a realidade do servidor de imagens.
- *HddImgProcess* – classe responsável por gerir o processo de recomprimir uma imagem para o máximo possivel, e gerar os blocos e respectivos ficheiros *torrent* para que a imagem possa ser descarregada através deste protocolo.

### 5.4.5 Diagramas de sequência

**Guardar uma imagem no servidor (upload)**

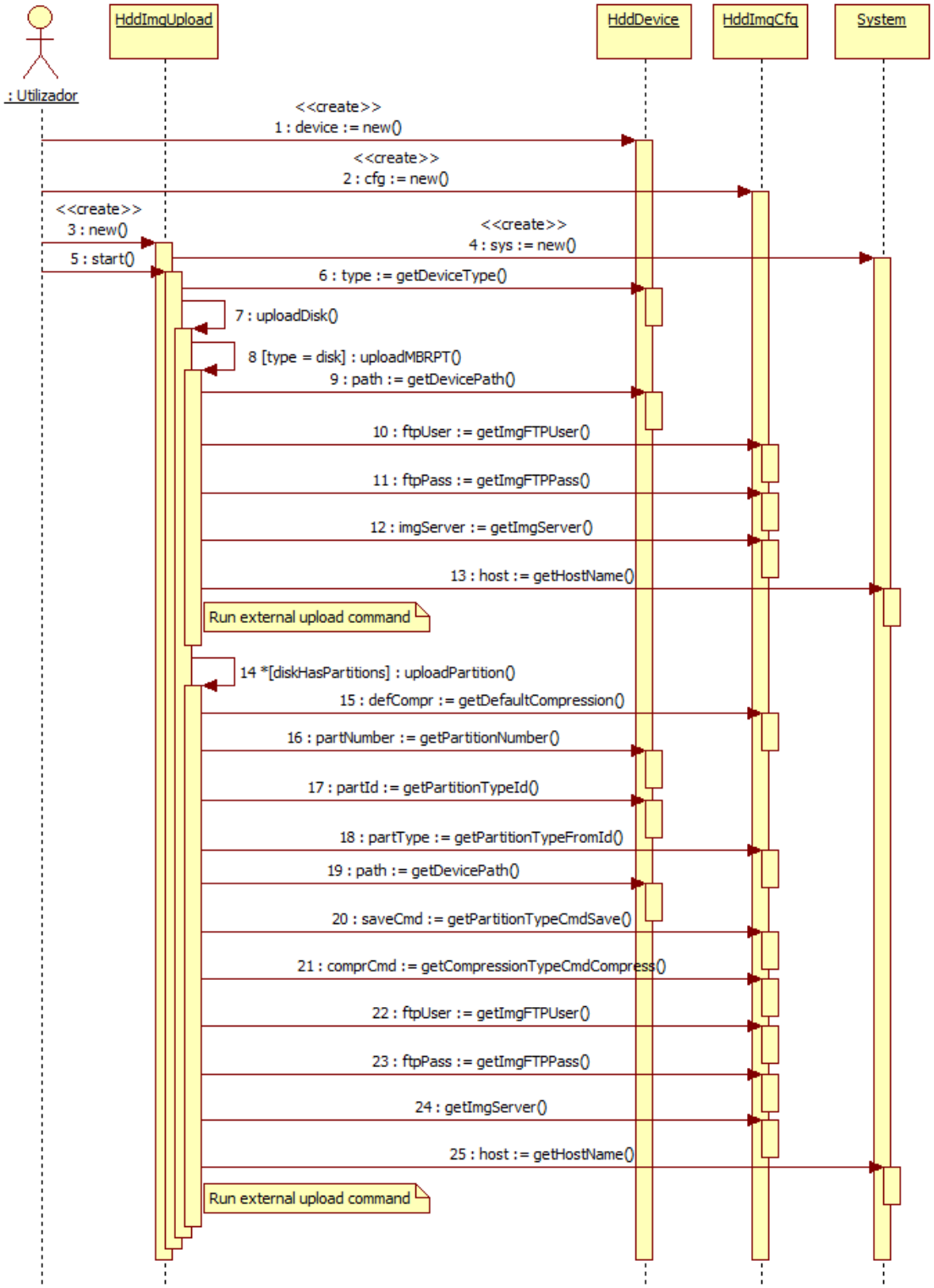



**Figura 19** - Diagrama de sequência - upload

**Restaurar uma imagem do servidor (*download*)**

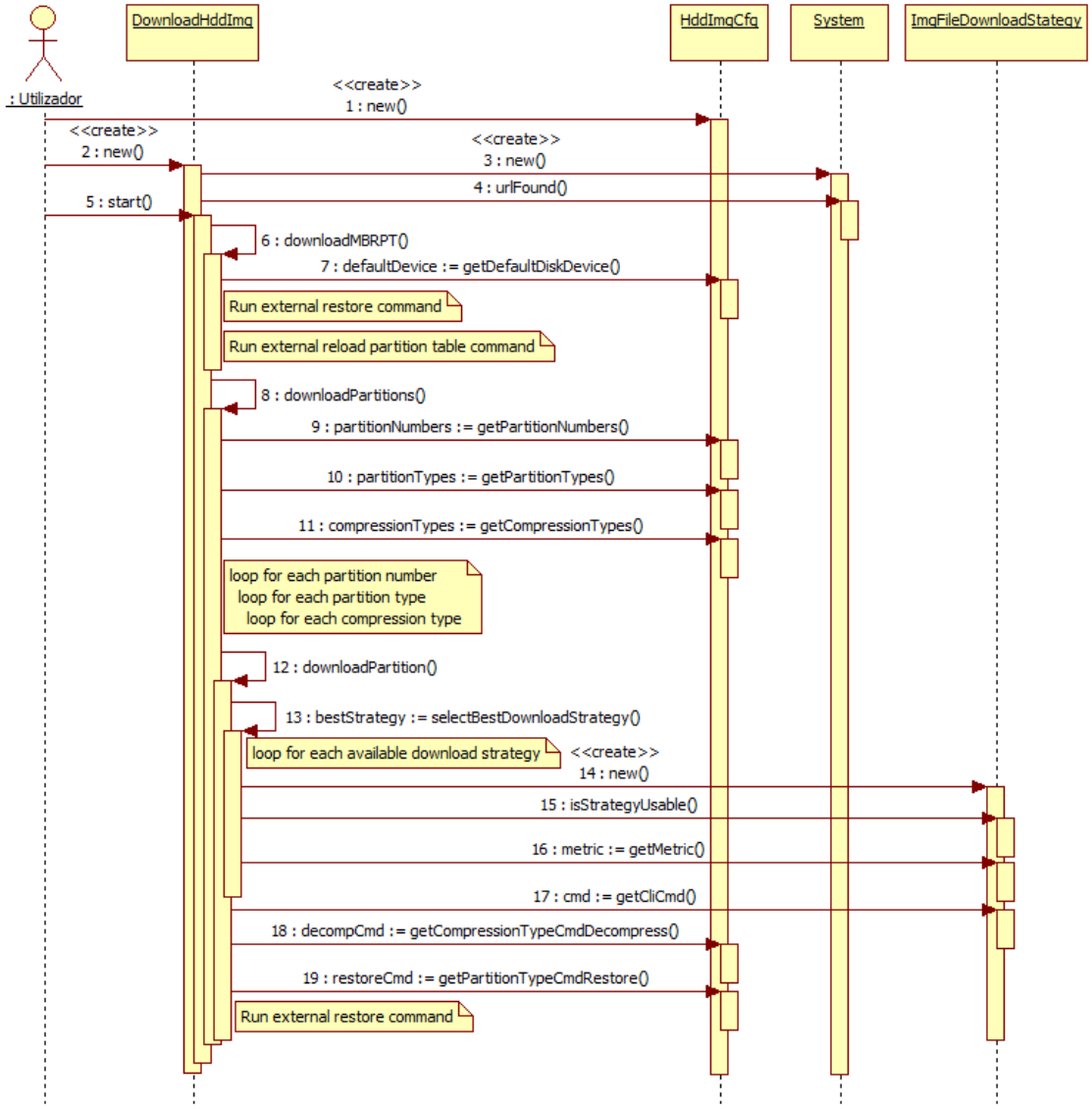



**Figura 20** - Diagrama de sequência - download

## 5.4.6 downloadBlocksByBitTorrent & downloadBlocksByHttp

Estes dois componentes devem ser implementados em executáveis autónomos para poderem ser integrados em *pipes* que serão lançados a partir da aplicação principal. O "downloadBlocksByHttp" é necessário pois o *download* por BitTorrent pode nem sempre estar disponível do lado do cliente, por exemplo, por limitações de memória.

De seguida apresenta-se o algoritmo destes dois componentes.

**downloadBlocksByBitTorrent**

- Verificar se no servidor existe o ficheiro *torrent* do primeiro bloco (blocoN.00.torrent)
- Enquanto não transferiu todos os blocos
    - Verificar se o ficheiro *torrent* do bloco N existe no servidor
    - Iniciar a transferência do bloco N em baixa prioridade
    - Se esta é a única transferência colocar a prioridade no máximo
    - Esperar o final da transferência do bloco N-1
    - Alterar a prioridade do bloco N para o máximo
    - Ler o bloco N-1 para o STDOUT
    - Remover o bloco N-1 da lista de torrents e apagar o ficheiro

**downloadBlocksByHttp**

- Verificar se no servidor existe o ficheiro *torrent* do primeiro bloco (blocoN.00)
- Enquanto existam blocos no servidor
    - Verificar se o ficheiro *torrent* do bloco N existe no servidor
    - Transferir o bloco N do servidor e colocar no STDOUT

## 5.5 Implementação

A fase de implementação do *software* foi relativamente simples pois os algoritmos de gestão da transferência dos dados já se encontravam implementados da fase de testes, bastando generalizar o código e integrar nas classes.

À medida que a implementação atingia pontos onde era possível a execução, foram feitos testes para eliminação preliminar de *bugs*.

## 5.6 Ferramentas adicionais GNU/Linux

Muitas interações com o sistema operativo podem ser feitas diretamente através do *python*, no entanto existem algumas para as quais é necessário usar ferramentas externas. De seguida descreve-se a função de várias ferramentas usadas explicitamente neste *software*.

- *dd* - é uma ferramenta que pertence às "GNU core utilities" e como tal vem tipicamente incluído em todas as distribuições GNU/Linux. Permite copiar dados "em bruto" de/para qualquer dispositivo de blocos (*block device*) (Free Software Foundation Inc., 2010a). É usado neste *software* para guardar ou restaurar o MBR e a tabela de partições do disco rígido em questão;
- *sfdisk* - é uma ferramenta que permite manipular tabelas de partições em Linux. Faz parte do pacote "util-linux" (Zak, 2012). É usado para obter uma listagem das partições

existentes no disco rígido;

- *blockdev* - permite obter/definir diversos parâmetros de dispositivos de blocos (*block devices*). Também faz parte do pacote "util-linux" (Brouwer, n.d.). É usada neste *software* para indicar ao sistema operativo que deve ler a nova tabela de partições (*partition table*) a partir do disco rígido, após esta ter sido restaurada a partir da imagem que se encontra no servidor;
- *df* - é uma ferramenta pertencente às "GNU core utilities" que permite obter a ocupação de um sistema de ficheiros (*file system*) (Free Software Foundation Inc., 2010b). É usada neste *software* para determinar se a pasta temporária onde são guardados os blocos descarregados por BitTorrent tem capacidade para dois blocos em simultâneo;
- *free* - permite obter a quantidade de memória ocupada e livre no sistema (Edmonds, n.d.). É usada neste *software* para determinar se o computador tem memória RAM suficiente para que se possa usar o método BitTorrent;
- *pidof* - devolve o *id* de um determinado processo que esteja em execução (Smoorenburg, n.d.). É usado neste *software* para determinar se o "transmission-daemon" está a ser executado e consequentemente se é possível usar o método BitTorrent.

## 5.7 Sistema operativo base

Como previamente indicado, todo o processo de guardar ou restaurar uma imagem não pode ser feito com o sistema operativo nativo em funcionamento. O facto de existirem ficheiros abertos faria com que a imagem não representasse o real estado do sistema.

Por este motivo todo o sistema é executado num ambiente *GNU/Linux live distribution* que permite que o sistema operativo seja executado totalmente em memória RAM sem necessidade de acessos ao disco.

A distribuição de *GNU/Linux* escolhida para este efeito foi o GRML que é definida pelos próprios autores como um *Live-CD* de arranque baseado no Debian e que inclui uma coleção de *software GNU/Linux* adequada para administradores de sistemas (Grml Live Linux, 2012). Este possui uma versão *small* que na sua versão 10.12 possui cerca de 90MB, e já inclui as principais ferramentas necessárias num ambiente de linha de comandos. Também é fácil instalar ferramentas adicionais caso necessário. Apesar desta escolha, à partida, qualquer outra *GNU/Linux live distribution* pode ser usada desde que possua, ou seja possível instalar, as ferramentas usadas neste projeto.

O GRML pode ser executado a partir de um CD, de um disco USB, ou ainda a partir da rede através de PXE (Intel Corporation, 1999).

Por uma questão de facilidade de utilização o sistema está configurado para arrancar por PXE bastando para isso tipicamente carregar no F12 (a tecla a usar pode mudar consoante o fabricante da BIOS) aquando do arranque do computador. Todo o sistema operativo será

descarregado para o computador através de TFTP e HTTP (Kildau, 2011) e quando o processo de arranque do sistema operativo estiver concluído, serão automaticamente instaladas as ferramentas e *scripts* adicionais necessários.

Caso a placa de rede do computador não suporte PXE ou não seja bem suportada pelo sistema de arranque, é sempre possível usar o método alternativo do CD ou mesmo do disco USB. A única diferença é que no final do arranque do sistema operativo é necessário executar manualmente o *script* que irá fazer a instalação das ferramentas e *scripts* adicionais.

Estas ferramentas e scripts adicionais são instalados com recurso a um *shell script* muito simples que os descarrega do servidor por HTTP e instala as ferramentas adicionais através do comando *apt-get*.

## 5.8 Integração final - restauro completamente automático de imagens

O processo de restauro de imagens foi pensado para ser completamente automatizado. A pensar nisto, na classe DownloadHddImg existe um atributo "autoDiscovery" que, caso no construtor não seja indicado um "imageBaseName", contém o nome que será usado para pedir a imagem automática ao servidor. Este processo de automatizar completamente o restauro de imagens envolve vários componentes do sistema de rede (DHCP, PXE, TFTP, iPXE, Apache + mod_rewrite) e como tal tem de ser parametrizado manualmente. Este processo apenas funciona com o sistema a arrancar por PXE. Os passos a seguir são os seguintes:

1. Todos os computadores pertencentes a uma determinada sala deverão estar registados manualmente no DHCP com um IP fixo e todos pertencentes à mesma subrede, por exemplo sala 1 => 10.1.1.X, sala 2 => 10.1.2.X, etc.
2. Como mencionado anteriormente, os clientes devem arrancar o GRML através de PXE por forma a que no final do arranque sejam executados os scripts de instalação do btHddImg bem como as suas dependências.
3. No servidor Apache, deve existir uma estrutura de diretorias do tipo "hddimgs/sala1", "hddimgs/sala2", etc. Para permitir ao automatismo detetar que computador pertence a que sala, o módulo "mod_rewrite" (The Apache Software Foundation, 2012b) que tipicamente já vem incluído, deve ser configurado de forma a mapear os endereços IP para a diretoria da imagem. Isto é conseguido com uma configuração como esta:

```
<Directory "/var/www/html/hddimgs">
  RewriteEngine on

  RewriteCond %{REQUEST_URI} /autodiscovery/([^/]+)$
  RewriteCond %{REMOTE_ADDR} ^10\.1\.1\.([0-9]+)?$
  RewriteRule ^autodiscovery/(.*)$ /hddimgs/sala1/$1 [R,L]

  RewriteCond %{REQUEST_URI} /autodiscovery/([^/]+)$
  RewriteCond %{REMOTE_ADDR} ^10\.1\.2\.([0-9]+)?$
  RewriteRule ^autodiscovery/(.*)$ /hddimgs/sala2/$1 [R,L]

</Directory>
```

4. Nos scripts finais de arranque do GRML e após a instalação do *script* do btHddImg bem como das dependências necessárias, este deve ser chamado com a opção de *Auto Discovery* (`./btHddImg.py -a`).
5. No final do restauro da imagem, o sistema irá reiniciar automaticamente e arrancar pelo sistema operativo nativo que acabou de ser restaurado.

## 5.9 Testes

Como mencionado anteriormente, já tinham sido realizados testes na fase de escolha das tecnologias de transferência bem como testes de despiste de *bugs* ao longo do desenvolvimento.

Assim partiu-se logo para a instalação final, desde a qual a aplicação se encontra em plena utilização na ESTCB.

# 6. Conclusões

## 6.1 Sobre o sistema desenvolvido

No seguimento do trabalho realizado pode-se concluir que o BitTorrent é uma ferramenta válida no cenário apresentado, típico em estabelecimentos de ensino. Apesar de ter apenas um *seeder* inicial (o servidor), o facto de todos os restantes *peers* se localizarem após o *bottleneck* leva a que após uma *piece* ser recebida por um destes fica facilmente disponível para os outros que se encontram no mesmo *switch* de rede. Introduzindo o *download* por blocos faz com que o protocolo perca alguma da sua eficiência, permanecendo no entanto muito superior à eficiência do HTTP. O protocolo *multicast* revelou-se inadequado para este cenário concreto na rede da ESTCB, no entanto esta conclusão não pode ser generalizada.

Existem vários factores que afetam o desempenho do protocolo BitTorrent, pelo que se algum dos componentes neste cenário for diferente (como a capacidade do *switch* da sala de aula, a versão do *software* "Transmission", o número de *broadcasts* na rede local ou até mesmo a aleatoriedade do protocolo) os resultados poderão variar substancialmente, mas o protocolo deverá manter a sua vantagem em relação ao HTTP principalmente à medida que o número de clientes aumente.

Seguindo estas conclusões foi desenvolvida a aplicação "btHddImg" em *python* e este sistema é atualmente usado para distribuir imagens de disco aos computadores das salas de aula da Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco.

## 6.2 Sobre o cumprimento dos objectivos

O objectivo geral inicialmente estabelecido foi plenamente cumprido no sentido que foi concebida uma aplicação que permite transferir da forma mais eficiente possivel imagens de um servidor para um ou vários clientes. No entanto dois objectivos mais particulares foram ajustados no decurso do projecto. Inicialmente previa-se que o sistema iria inteligentemente escolher a melhor forma de distribuição das imagens. Tal objectivo revelou-se desnecessário pois os testes no terreno revelaram que apenas há duas escolhas: ou é apenas um cliente e a transferência é por HTTP ou são vários e é por BitTorrent. Assim esta escolha passou a ser responsabilidade do utilizador. Outro objectivo ajustado prende-se com a concepção de uma interface *web* para gerir as imagens no servidor. Considerando que o tipo de utilizador ao qual este *software* se destina são os Administradores de Sistemas, e o tipo de operações que esta interface iria permitir (apagar imagem, mudar nome de imagem, etc.) optou-se por permitir que o utilizador faça directamente estas operações no servidor através do protocolo FTP.

## 6.3 Trabalho futuro

Este projeto pode ser melhorado desenvolvendo um cliente de BitTorrent que aplique os conceitos de *windowing* como os apresentados em (Savolainen and Raatikainen, 2008) o que permitiria: abolir a necessidade de dividir o ficheiro da imagem em blocos do lado do servidor, o que levaria a que bastaria um ficheiro *torrent* e, consequentemente, o atraso inicial que se nota antes do inicio do *download* de cada bloco passaria apenas a existir uma vez.

## 6.4 Considerações finais

Com a realização deste projecto espero ter contribuído para a resolução do problema identificado na instalação de *software* nas salas de aula da ESTCB. Tendo já algum conhecimento das ferramentas e *software* usados, este trabalho ajudou-me essencialmente a adquirir competências na área da análise e produção de trabalhos científicos, onde a utilização de métodos de trabalho estruturados e sistematizados é de extrema importância.

# Referências bibliográficas


Aalto, S., Lassila, P. and Raatikainen, N., 2010. P2P video-on-demand: Steady state and scalability. *2010, 2010 IEEE*, Available from: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5683557> [Accessed 14 March 2012].

Acronis Inc., 2012a. *Acronis Snap Deploy*. [online] Available from: <http://www.acronis.eu/enterprise/products/snapdeploy/> [Accessed 1 May 2012].

Acronis Inc., 2012b. *Acronis Snap Deploy Features*. [online] Available from: <http://www.acronis.com/enterprise/products/snapdeploy/#features> [Accessed 9 June 2012].

Bestofmedia Group, 2012. *Tom's Hardware: Streaming Writes Benchmark Pattern*. [online] Available from: <http://www.tomshardware.com/charts/hdd-charts-2012/IOMeter-2006.07.27-Streaming-Writes,2929.html> [Accessed 5 April 2012].

BitTorrent Inc., 2012a. *BitTorrent Glossary*. [online] Available from: <www.bittorrent.com/help/manual/glossary> [Accessed 16 May 2012].

BitTorrent Inc., 2012b. *Local Peer Discovery*. [online] Available from: <http://www.bittorrent.com/help/manual/glossary#LPD.

BitTorrent Inc., 2012c. *Peer Exchange*. [online] Available from: <http://www.bittorrent.com/help/manual/glossary#PEX> [Accessed 16 May 2012].

BitTorrent Inc., 2012d. *uTorrent Connection Setup*. [online] Available from: <http://www.utorrent.com/help/guides/connection-setup.

Blizzard Entertainment Inc., 2012. *Blizzard Downloader*. [online] Available from: <http://us.blizzard.com/en-us/company/about/legal-faq.html> [Accessed 16 May 2012].

Brouwer, A.E., *blockdev man page*. [online] Available from: <http://man.cx/blockdev> [Accessed 9 June 2012].

Carle, G., 1997. Survey of error recovery techniques for IP-based audio-visual multicast applications. *Network, IEEE*, Available from: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=642357> [Accessed 24 March 2012].

Cheng, B. et al., 2008. Evaluation and optimization of a peer-to-peer video-on-demand system. *Journal of Systems Architecture*, 54(7), pp.651-663. Available from: <http://linkinghub.elsevier.com/retrieve/pii/S1383762107001300> [Accessed 8 March 2012].

Cohen, B., 2003. Incentives build robustness in BitTorrent. *Workshop on Economics of Peer-to-Peer systems*, Available from: <http://www.ittc.ku.edu/~niehaus/classes/750-s06/documents/BT-description.pdf> [Accessed 11 March 2012].

Cohen, B., 2008. *The BitTorrent Protocol Specification*. [online] Available from: <http://www.bittorrent.org/beps/bep_0003.html> [Accessed 11 March 2011].

Deering, S., 1989. *Host Extensions for IP Multicasting*. [online] Available from: <https://tools.ietf.org/html/rfc1112> [Accessed 1 May 2012].

Edmonds, B., *free man page*. [online] Available from: <http://man.cx/free.

Engling, D., *opentracker Official Website*. [online] Available from: <http://erdgeist.org/arts/software/opentracker/> [Accessed 12 April 2012].

Evans, C., 2012. *vsftpd Official Website*. [online] Available from: <https://security.appspot.com/vsftpd.html> [Accessed 12 April 2012].

Fielding, R. et al., 1999. Available from: <http://tools.ietf.org/html/rfc2616.

Finley, B.E. et al., 2007. SystemImager and BitTorrent : a peer-to-peer approach for Large-Scale OS Deployment.

Free Software Foundation Inc., 2010a. *dd man page*. [online] Available from: <http://linux.die.net/man/1/dd> [Accessed 27 March 2012].

Free Software Foundation Inc., 2010b. *df man page*. [online] Available from: <http://linux.die.net/man/1/df> [Accessed 27 March 2012].

Free Software Foundation Inc., 2011. *wget Official Website*. [online] Available from: <http://www.gnu.org/software/wget/> [Accessed 9 April 2012].

Grml Live Linux, 2012. *Grml Live Linux*. [online] Available from: <http://www.grml.org/> [Accessed 5 April 2012].

Hubert, B., 2012. *tc man page*. [online] Available from: <http://linux.die.net/man/8/tc> [Accessed 16 May 2012].

Intel Corporation, 1999. Available from: <http://download.intel.com/design/archives/wfm/downloads/pxespec.pdf.

International Business Machines Corp., 2012. *IBM Tivoli Provisioning Manager for OS Deployment*. [online] Available from: <https://www-01.ibm.com/software/tivoli/products/prov-mgr-os-deploy/> [Accessed 11 March 2011].

Jacobson, V., 1988. Congestion avoidance and control. *ACM SIGCOMM Computer Communication Review*, Available from: <http://dl.acm.org/citation.cfm?id=52356> [Accessed 10 April 2012].

Kildau, C., 2011. *How to PXE boot GRML and OpenBSD without NFS*. [online] Available from: <http://www.chrisk.de/blog/2011/03/how-to-pxe-boot-grml-and-openbsd-without-nfs-like-boot-kernel-org/> [Accessed 1 May 2012].

Knaff, A., 2011. *udpcast Official Website*. [online] Available from: <http://www.udpcast.linux.lu/> [Accessed 31 March 2012].

Lakshman, T., Madhow, U. and Suter, B., 2000. TCP/IP performance with random loss and bidirectional congestion. *IEEE/ACM Transactions on*, 8(5), pp.541-555. Available from: <http://dl.acm.org/citation.cfm?id=355155> [Accessed 12 June 2012].

Lange, T., 2012. *FAI - Fully Automatic Installation Official Website*. [online] Available from: <http://fai-project.org/> [Accessed 1 May 2012].

Loewenstern, A., 2008. *DHT Protocol*. [online] Available from: <http://bittorrent.org/beps/bep_0005.html.

McKenney, P., 1991. Stochastic fairness queuing. *Internetworking: Theory and Experience*, Available from: <http://onlinelibrary.wiley.com/doi/10.1002/cbdv.200490137/abstract> [Accessed 26 March 2012].

Microsoft Corporation, 2012a. *Windows 7 Backup and Restore*. [online] Available from: <http://windows.microsoft.com/en-US/windows7/products/features/backup-and-restore> [Accessed 1 May 2012].

Microsoft Corporation, 2012b. *Windows 7 Backup and Restore - Restore your computer from a system image backup*. [online] Available from: <http://windows.microsoft.com/en-US/windows7/Restore-your-computer-from-a-system-image-backup> [Accessed 1 May 2012].

Microsoft Corporation, 2012c. *Windows Deployment Services*. [online] Available from: <http://technet.microsoft.com/en-us/library/cc772106(v=WS.10).aspx> [Accessed 1 May 2012].

Microsoft Corporation, 2012d. *Windows Deployment Services - FAQ*. [online] Available from: <http://technet.microsoft.com/en-us/library/cc732729(v=ws.10).aspx#General> [Accessed 1 May 2012].

Netcraft Ltd, 2012. *Netcraft April 2012 Web Server Survey*. [online] Available from: <http://news.netcraft.com/archives/category/web-server-survey/> [Accessed 12 April 2012].

O'Donnell, C.M., 2008. Using BitTorrent to distribute virtual machine images for classes. In: Proceedings of the 36th annual ACM SIGUCCS fall conference: moving mountains, blazing trails, 2008. ACM. Available from: <http://dl.acm.org/citation.cfm?id=1450040> [Accessed 11 March 2012].

Pfleeger, S., 2004. *Engenharia de Software: teoria e prática*. Prentice-Hall, Inc. Available from: <citeulike-article-id:435480.

Postel, J., 1980. Available from: <http://tools.ietf.org/html/rfc768.

Postel, J. and Reynolds, J., 1985. RFC959, File Transfer Protocol (FTP). *Internet Engineering Task Force*, Available from: <http://tools.ietf.org/html/rfc959.

Python Software Foundation, 2012. *Python Programming Language – Official Website*. [online] Available from: <http://www.python.org/> [Accessed 12 April 2012].

Red Hat, *KickStart*. [online] Available from: <https://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/5/html/Installation_Guide/ch-kickstart2.html> [Accessed 1 May 2012].

Savolainen, P. and Raatikainen, N., 2008. Windowing BitTorrent for video-on-demand: Not all is lost with tit-for-tat. *Conference, 2008. IEEE*,pp.1-6. Available from: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4698248> [Accessed 14 March 2012].

Schulze, H. and Mochalski, K., 2009. *Internet Study 2008/2009*. Available from: <http://www.ipoque.com/sites/default/files/mediafiles/documents/internet-study-2008-2009.pdf> [Accessed 11 March 2012].

Silva, A.M.R.D. and Videira, C.A.E., 2001. *UML, Metodologias e Ferramentas CASE*. E. C. Atlântico (ed.), Centro Atlântico.

Smoorenburg, M. van, *pidof man page*. [online] Available from: <http://man.cx/pidof> [Accessed 9 June 2012].

Stanford_University, 2010. *Stanford IT Services - System Deployment*. [online] Available from: <http://itservices.stanford.edu/strategy/system-deployment> [Accessed 18 April 2012].

Stenberg, D., 2012. *curl Official Website*. [online] Available from: <http://curl.haxx.se/> [Accessed 27 March 2012].

Stevens, W.R., 1994. *TCP/IP Illustrated, Volume I: The Protocols*. B. W. Kernighan (ed.), Addison-Wesley. Available from: <http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:TCP+/+IP+Illustrated+,+Volume+1+:+The+Protocols#1.

Svensson, E., 2011. *transmissionrpc Official Website*. [online] Available from: <http://packages.python.org/transmissionrpc/> [Accessed 12 April 2012].

Symantec Corporation, 2012. *Symantec Ghost Solution Suite*. [online] Available from: <http://www.symantec.com/ghost-solution-suite> [Accessed 11 March 2011].

Tanenbaum, A.S., 2002. *Computer Networks (4th Edition)*. Prentice Hall. Available from: <http://www.amazon.com/Computer-Networks-4th-Andrew-Tanenbaum/dp/0130661023.

Tanenbaum, A.S., 2001. *Modern Operating Systems, 2nd edition*. Prentice Hall. Available from: <http://www.amazon.com/exec/obidos/ASIN/0130313580/acmorg-20.

The Apache Software Foundation, 2012a. *Apache HTTP Server Security Tips*. [online] Available from: <https://httpd.apache.org/docs/2.4/misc/security_tips.html> [Accessed 12 April 2012].

The Apache Software Foundation, 2012b. *Apache Module mod_rewrite*. [online] Available from: <http://httpd.apache.org/docs/current/mod/mod_rewrite.html> [Accessed 10 May 2012].

Transmission Project, 2012a. *Transmission Configuration Options*. [online] Available from: <https://trac.transmissionbt.com/wiki/EditConfigFiles> [Accessed 16 May 2012].

Transmission Project, 2012b. *Transmission Official Website*. [online] Available from: <http://www.transmissionbt.com/> [Accessed 11 April 2012].

Tsai, T., 2012. *partclone Official Website*. [online] Available from: <http://partclone.org/> [Accessed 27 March 2012].

Tuxera, 2012. *ntfsclone man page*. [online] Available from: <http://linux.die.net/man/8/ntfsclone> [Accessed 27 March 2012].

Wang, R., Pau, G. and Yamada, K., 2004. TCP startup performance in large bandwidth networks. *2004. Twenty-third*, 00(C), Available from: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1356968> [Accessed 23 March 2012].

Wood, A., 2012. *pv Official Website*. [online] Available from: <http://www.ivarch.com/programs/pv.shtml> [Accessed 9 June 2012].

Zak, K., 2012. *util-linux Official Website*. [online] Available from: <https://github.com/karelzak/util-linux> [Accessed 9 June 2012].

# Anexos

No CD encontra-se a seguinte informação complementar:

- *Scripts* usados na realização dos testes
- Folha de cálculo com todas as medições obtidas
- Código fonte do software desenvolvido em *python*
- Partes relevantes do ficheiro de configuração do software *vsftpd*
- Partes relevantes do ficheiro de configuração do software *apache* com o módulo *mod_rewrite*
- Partes relevantes do ficheiro de configuração do software *pxelinux*
- Partes relevantes do ficheiro de configuração do software *ISC dhcpd*