**Universidade Nova de Lisboa**

Faculdade de Ciências e Tecnologia

*Departamento de Informática*

# Bitocast

A hybrid BitTorrent and IP Multicast content distribution solution

Por Tiago Norton de Matos de Andrade e Silva

Dissertação apresentada para obtenção do Grau de Mestre
em Informática, pela Universidade Nova de Lisboa,
Faculdade de Ciências e Tecnologia.

Lisboa

2009

Esta tese foi orientada pelo Professor Doutor Sérgio Duarte, Professor no Departamento de Informática, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa.

# Acknowledgements

I want to thank my wife and children for keeping up with me while working to achieving this goal, my friends who reviewed this work, providing very thoughtful comments, and finally and most specially, for the example given by my father in his hard work and focus when pursuing a goal.

# Sumário

Nos últimos anos tem-se vindo a verificar uma crescente utilização da Internet como meio para transmitir grandes conteúdos multimédia. Estes conteúdos têm sido distribuídos recorrendo a diversas tecnologias tais como servidores centralizados, redes de distribuição de conteúdos e mais recentemente protocolos *peer-to-peer*. Entre os protocolos *peer-to-peer*, o mais utilizado tem sido o BitTorrent e tem provado ser dos mais eficientes.

Diversos estudos têm vindo a analisar e propor melhorias ao protocolo BitTorrent de modo a torná-lo mais justo, mais fiável e mais eficiente. Existem também propostas para a criação de redes *overlay* sobre a rede BitTorrent ou mesmo de estender o protocolo de modo a suportar *streaming*. Neste trabalho apresentamos e analisamos uma alternativa de distribuição de conteúdos a que chamámos Bitocast. Esta alternativa recorre à utilização de IP Multicast em conjunto com BitTorrent, de modo maximizar a utilização da largura de banda do provedor, optimizando a distribuição geral da sua carteira de conteúdos para grandes audiências.

# Abstract

In recent years we have observed an increased use of the Internet as a means for transmitting large content. There have been several technology attempts to attack this problem, including costly distribution networks and, more recently, peer to peer (P2P) protocols. Amongst these P2P protocols, BitTorrent has proven itself as an effective means for transmitting large content items and today enjoys great popularity.

Numerous researchers have analyzed BitTorrent and proposed concepts and models to enhance its reliability, efficiency and fairness. Further, there are proposals to extend BitTorrent to support on-demand multimedia streaming. In this Dissertation we present Bitocast, a content distribution system that combines IP Multicast and BitTorrent protocols in order to achieve a more efficient usage of an Internet Service Provider's network and reduce download time when serving large sets of contents to large audiences.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Content production is changing. No more is it solely in the hands of big publishing or media companies. Nowadays anyone can be an author of an article, a video or a talk show and easily make it available to whoever is interested in the matter. The World Wide Web is becoming the central point of content distribution.

Content consumption is also changing. We are at an inflection point when users are no more compulsorily attached to a *push model* - where they consume what others decide - but are migrating to a *pull model* where they decide what, when and where to consume content.

Internet bandwidth has evolved significantly over the last 10 or 15 years, increasing from the typical access of 56 kpbs - established with a modem over an analog line - to the current broadband cable access of 100 Mbps. 100 Mbps seems like a huge bandwidth and that it will suite our needs, but in reality it is a matter of time until it is not enough. We venture to say that Parkinson's law [37], which states that "Work expands so as to fill the time available for its completion", can be re-written regarding bandwidth as *content expands to fill the bandwidth available for its distribution*. The filling of the available bandwidth can easily be seen as a fact by looking at the produced content and the services being offered nowadays: We see a trend from media companies in reselling old content but this time with higher quality (High definition video in blu-ray discs for example); digital will replace physical distribution on many industries: "There's no question digital will overtake physical. It happened in music and will happen to our industry" [7], said Microsoft XBox Europe's Vice President

of strategic marking; New media cameras and higher resolution formats are gaining wider adoption; Voice over IP (VoIP) and Video on Demand (VoD) ; among others.

The usage of media over the Internet is also an example of that increased bandwidth. New services like Video-on-Demand (VoD) are now possible. Although VoD is starting to appear as a commercial offering by telephone and cable television companies, its cost is high for the supplier which in turn makes it also expensive for the consumer. The VoD scalability is hard to achieve due to the limitations of the underlying architecture and the fact that a huge amount of information is being sent every time to each user in a unicast model.

Using the Internet as the means to distribute content is not a niche market. The demand and the number of consumers is increasing steadily. The number of people connected to the Internet is increasing all over the world, being seen as ubiquitous in industrialized countries and still increasing in countries in Africa, The Middle East and Latin America among others. There is even an intent in making Internet access a human right (something that is already a fact in Estonia and Greece [1]). In 2006, the Internet video site YouTube[2] announced that its users were watching 100 million videos per day. This amount of people watching videos makes YouTube responsible for around 10% of all Internet traffic and 20% of Hyper Text Transfer Protocol (HTTP) traffic [15], with an estimate bandwidth usage of 6 Petabytes per month [29]. Clearly, media and content distribution solutions need to be able to serve very large audiences.

Nowadays, the World Wide Web is supported over HTTP [22] and due to its success the protocol started to be expanded and given new uses. It currently supports the download of big files, it enables streaming of audio and video by the means of progressive download, but it clearly was not designed to be a platform for the distribution of large files to a very large number of recipients. This is because HTTP works over a client/server stateless model, where a client sends a request and waits for a reply. The request from client A has no relation to the request of client B and the same information might travel the same network links over and over again, resulting in waste and bad network usage.

HTTP has been used as a "workaround" for large content distribution due to its popular-

[1] http://en.wikipedia.org/wiki/Internet_access (accessed June 2009).
[2] www.youtube.com (accessed June 2009).

ity, standardization and dissemination. The wide adoption of the current solution for high traffic content publishers was, and still is, enabled by the use of *content distribution networks* (CDN) [40]. CDNs act like proxies or caches that are closer to the clients. They take content to the edge of the network and also help by sharing the distribution burden amongst multiple servers. Clearly this model is difficult to scale and has high cost.

Around 2000, the so called *peer-to-peer* (P2P) set of services started to appear in order to enable another way to share files. Its success was so large that in 2003 around 60% [3] of Internet traffic was P2P file sharing (as it can be seen in figure 1.1). In P2P file sharing systems, peers function not only as clients that request content but also as servers of content. Their sole purpose is the exchange of content. Unlike the typical client-server model, in P2P all nodes are seen as being equal [4] and have no central server to coordinate their actions. Nodes participate in the exchange voluntarily and do so by ongoing a best effort attempt without any guarantees. They leverage their upload capabilities by serving content to their peers and augmenting the overall download capabilities of the system. In abstract, the system is self scaling since the available bandwidth grows with the number of peers participating on the system. Peers download content by establishing direct connections with their peers and downloading small parts of the content from one another until the file is fully assembled.

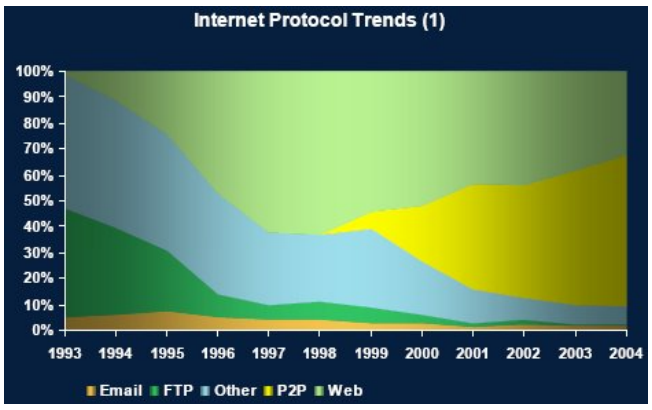


Figure 1.1: Internet Protocol Trends

[3] Source: CacheLogic P2P in 2005.

[4] Enhancements of some P2P protocols do have some notion of Super Nodes but the core idea of equality still exists for most nodes.

P2P applications fall into 3 possible architectures: mediated, pure and hybrid [32]. In all of these architectures, the data transfer is still made by P2P. What differs between them is how control, coordination and search is orchestrated. Some types of P2P systems serve only the purpose of downloading files while others also enable searching.

There are also some P2P solutions that go beyond simple file exchange and allow streaming. Some examples are CoolStreaming [47] and GridCast [14]. These systems benefit from the characteristics of P2P systems, like their capability for self organization, bandwidth scalability, and network path redundancy [28]. But these P2P streaming solutions suffer from high startup time, jitter [5] and seek latency [47] not allowing for a great end user experience and zapping, a pattern showed by most TV viewers.

The first well known P2P system, called Napster, stored a central index with all content available on the network at a given time. It was invented by Shawn Fanning when he was 18 years old. His intention was to enable a easy way for people to swap files through a centralized file server. In Napster, Peers would inform the central server of the content they had to share and could also query the server for existing content. Napster had large popularity as a P2P system and it was followed by Gnutella, eDonkey and BitTorrent.

Amongst P2P protocols BitTorrent is one of the most successful one. It was invented by Bram Cohen in 2001 and was first released on 2nd July of 2001 via a post on a Yahoo message group[6]. Cohen published BitTorrent code as open source and showed it at a hacker conference in 2002. The intent of his project was to ease the sharing and exchange of Linux Software. BitTorrent belongs to the Mediated P2P type of architecture. It uses one or more central servers to establish the first contact between peers. Peers exchange data directly between themselves. They typically select to upload data to the nodes they received from, this way reducing the existence of peers that download and do not contribute to the network (also known as free riders). For this reason the download capacity is very much related to the upload capacity of the peer. BitTorrent does not take care of searching content and delegates this task to other servers that do not take part on the file exchange itself.

[5] The variation in the time between packets arriving, caused by network congestion, timing drift, or route changes.

[6] http://finance.groups.yahoo.com/group/decentralization/message/3160 (accessed July 2009).

It is stated that, in 2005, BitTorrent traffic constitute at least 20% of the entire traffic volume on the Internet [10] and that more than 20 million people downloaded the BitTorrent client application[7]. This popularity has also translated in the adoption of BitTorrent by Content Providers. Some examples are Amazon with their Simple Storage Server (S3), Warner Brothers, 20th Century Fox and BBC. BitTorrent enables the delivery of large files to many users at a reduced deployment cost. Despite its popularity, it was a P2P solution, and as such its use of the network is not very efficient. This is because the same information passes through the same network links many times and can lead to network saturation. BitTorrent doesn't consider any proximity, number of hops or ISP underlying topology in the connections it establishes. This causes losses for the Internet Service Provider (ISP) because it augments the network's bandwidth usage for file transfers and generates a lot of cross ISP traffic.

HTTP and P2P content distribution solutions share a similar problem. The same information is sent several times over the same network links causing inefficiency. P2P is usually even worse in this inefficiency because the communication is typically made from one edge of the Internet to the other, passing through the center or main backbones. *Content distribution networks* reduce this inefficiency by taking content to the edge of the Internet but the problem is still there, only at a reduced level. In both solutions the same information keeps on flowing back and forth for each single request.

On the other hand, IP Multicast is a much better way to send data to multiple peers because it allows for the packet to transverse each network link at most once, greatly reducing the network overhead. The sender only needs to send one packet to the Multicast group and the network and underlying protocols will take care of the delivery of the packet to the members of that group. However, IP Multicast has some limitations. It does not give any guarantee of packet delivery which can be very problematic for certain uses, namely content distribution. Also, it was not widely adopted in the last years due to its scalability problem and because there is no clear business model for it [20]. Despite not being deployed over the Internet, IP Multicast can still be used over private networks such as companies, universities or even ISPs. In order to diminish IP Multicast's drawback of the best effort delivery policy, one could think of a solution that uses IP Multicast and another mechanism to compensate for

[7] http://www.wired.com/wired/archive/13.01/bittorrent.html (accessed July 2009).

the missing packets that IP Multicast might produce.

Content distribution solutions based on IP Multicast require the users to be synchronized in order to start delivery, not allowing for late arrivals to receive a complete copy of the content. These "synchronized" IP Multicast stream solutions have no means to overcome the unreliability of the transfer. In order to solve this issue, some systems use a *data carousel*[8] approach, which is highly inefficient.

## 1.1 Motivation

The Internet is here to stay and every day that passes we see new ways of using it. If one accepts it, network bandwidth is, and always will be, insufficient, so, it is important to find ways to optimize its use. The current foundation for content distribution over the web (HTTP) is not suited to handle a large set of users that wish the same content at the same time. P2P is also not the solution since it is not efficient in the network traffic it generates. Multicast is a solution for this problem because it allows for the packets to transverse each network link at most once, but unfortunately it is not available over the Internet. Despite this fact, there is no reason for it not to be adopted in closed networks. Multicast doesn't also guarantee delivery, so, using it for content distribution might lead to having missing pieces of content.

The issue at hand is how can we create a more cost effective and network friendly alternative to distribute large sets of content to a large audience, at an acceptable price and with a good end-user experience. Combining BitTorrent and IP Multicast to leverage their respective strengths seems to be a great way to reduce the ISP's network usage and also achieve a better end-user experience by possibly reducing the download time and the upload bandwidth consumption.

[8] A format that interleaves information streams in a repeating pattern.

## 1.2 Goal

The goals of this work are to propose and study the use of different alternatives of combining BitTorrent and IP Multicast together. We are looking for a solution that might possibly create a better user experience (by lowering download times or increasing the quality of video files for example) and/or lower the ISP's costs associated with network usage. Based on this idea, we will also elaborate on a means to better transmit to the end-user a large portfolio of content, that should be available on an acceptable time frame and with the best quality possible (in case of media). We elaborate on the alternatives, analyze their impact on a simulated scenario and lay down the path for the best possible solution. The implementation of the solution is not part of our goal, only the simulation of scenarios, their analysis and benefits of their use. Our work assumes the availability of IP Multicast communications, which can easily be the case for any Internet provider that also works as a content provider for its internal clients.

## 1.3 Outline of the Dissertation

In the next sections, we will describe BitTorrent and the IP Multicast protocol. On the third chapter related work will be presented, followed by the details of our proposal called Bitocast. In chapter five we present Bitocast simulator and follow to chapter six where we will analyze and draw conclusions on the simulations that were made to compare Bitocast versus BitTorrent. We will then present a scenario where Bitocast is used to serve a very large portfolio of content on a closed ISP network and conclude with the indication for future work.

# Chapter 2

# Base protocols

The following sections will detail the existing protocols that we base our work upon.

## 2.1 BitTorrent protocol

Following the success of other P2P applications such as Napster, Gnutella and eDonkey, Cohen invented BitTorrent protocol in 2001. His main goal was to ease the sharing of Linux software. BitTorrent is a simple and open P2P protocol that leverages the peer's upload capacity to enable fast and efficient download of static large files. Peers wanting to download a specific content form a *swarm network*[1], where not only do they download the desired content but also contribute parts of it to the swarm network. Since it depends on a central server that enables peers to meet one another, BitTorrent is considered a Mediated P2P type of architecture.

The protocol is often confused with the BitTorrent client software that implements the BitTorrent protocol in order to download files. This piece of software was first developed by Cohen when he created the protocol. Since the protocol was released, a lot of other BitTorrent clients have appeared. There are over 50 BitTorrent software clients developed [2]. Previous versions of this BitTorrent client were open source, but since version 6.0 that BitTorrent, Inc. re-branded the client and it is no longer open source.

[1] a group of peers connected to each other to share a "torrent" is called a swarm.
[2] http://en.wikipedia.org/wiki/List_of_BitTorrent_clients (accessed July 2009).

Since the appearance of P2P protocols, which enable a quick, easy and inexpensive way to download files, that a lot of debate arose around their use. This is because most of the people that use these protocols do it to exchange copyrighted content. The first most successful P2P client that enabled easy exchange of MP3 files was Napster. It was accused of facilitating the transfer of copyrighted material by the Recording Industry Association of America (RIAA) and had to shutdown its services in July 2001 due to a court order. The closing of the service led the way to a new set of decentralized P2P protocols and applications that continue to enable file distribution, which have been very hard to control.

In BitTorrent the only part that is centralized is a service called "Tracker". It serves solely as a meeting point that allows peers to establish connections between them. Some statistical information is sent by the peers to the tracker but it is merely for information purposes. When queried, the tracker always answers with a random list of peers. Other systems suggest different peer selection algorithms but they might "result in a power law graph, which can get segmented after only a small amount of churn" [17]. The Tracker also serves statistical information about the files to be exchanged.

Since the trackers do not have any copyrighted material stored locally, it is debatable if they are in infringement of copyright laws. Despite this fact, several web sites that enable the search of "torrent" files were closed in recent years [3].

When content distributers use *content distribution networks* the traffic is taken to the edge, closer to the consumer and might produce lower cross ISP traffic. As we migrate to use P2P solutions the traffic between ISP's increases. Especially because most P2P protocols do not take into account the network topology in their peering decisions. The traffic inside the ISP's network has no direct cost for the ISP but the same does not apply to the cross ISP traffic. For this reason, several studies and solutions have been proposed in order to shape, block or limit cross-ISP P2P traffic.

[3] http://en.wikipedia.org/wiki/Legal_issues_with_BitTorrent (accessed June 2009).

### 2.1.1 Architecture

A BitTorrent system is constituted by peers, a Tracker that keeps a list of peers (interested in a specific file) and statistics of the system, and a file known as "torrent" that is hosted on a web server. The peers are known as *seeds* if they have a complete copy of the file or as *leechers* otherwise. The "torrent" is a small file that contains metadata about the content to be distributed, such as its creation date, the url of the *tracker*, and checksum information about the pieces of the files. The content that BitTorrent distributes can be composed of one or multiple files. An overview of the system can be seen in figure 2.1.

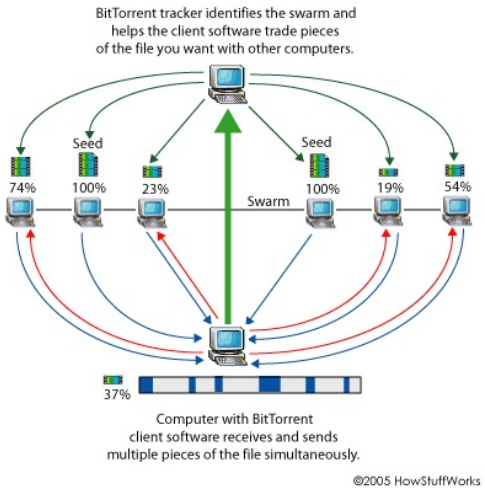


Figure 2.1: BitTorrent overview

To download the content one must first get a copy of the "torrent" file. This can be done by searching specific "torrent" search sites, by directly downloading it from a known url or receiving it by any digital means. With the "torrent" file, the BitTorrent client[4] can connect to a *tracker* and get a list of peers participating in the swarm for that content. After having a list of peers the client then connects to them and starts negotiating the exchange of data directly between peers. The file to be downloaded is split into pieces that are described in the "torrent" file, and each piece is composed of several blocks. Data exchange between peers is performed at the block level.

As opposed to other P2P mechanisms, BitTorrent focus only on the exchange of one content (that can contain multiple files). The performance of the download of one content will not

[4] An application that implements the BitTorrent protocol and allows for the existence of a peer on the swarm network.

impact, positive or negatively, the download rate of another content (aside from consuming available node bandwidth).

Enhanced versions of BitTorrent can also live without the existence of a Tracker. This is feasible by using a *distributed hash table* (DHT) [41] network that enables all users that are downloading the same file to find each other without relying on a central server. A DHT is a decentralized lookup service that works like a hash table (by mapping keys into values) where the responsibility for this lookup is shared amongst the nodes of the network. Several protocols, such as Chord [44], Kademlia [33] and Pastry [42] implement this model.

Despite not being part of the standard protocol many BitTorrent clients implement a feature called *Multitracker*, that enables the existence of multiple Trackers per "torrent". This way, if a certain tracker is not available other trackers can keep on providing the needed role.

### 2.1.2 Swarm

The swarm is composed of the set of peers that don't yet have a complete copy of the content (*leechers*) and the ones just offering it (*seeds*). They are interconnected by a random graph forming an unstructured overlay network.

When queried for a list of peers the *tracker* selects a random set (usually 35) without any consideration. The *tracker* also informs the peer the amount of time it should wait in order to call it again to update the peer's metrics and to send an updated list of peers. If the number of remote peers that a peer is connected to decreases below a certain limit (by default 20), because some of the peers left the swarm for example, the client will reconnect to the *tracker* (even before the specified time has elapsed) in order to enquire about the existence of new peers.

### 2.1.3 Peer Connections

Peers first establish connections with the set of other peers that the *tracker* delivered and start to exchange information about the availability of each piece of the file (also known as bitmap). With whom each peer exchanges pieces is dictated by a cost benefit analysis of each

connection, implemented via mechanisms known as *tit-for-tat*, *anti-snubbing* and *optimistic unchoke*.

The exchange of pieces between peers is based on a *choke / unchoke* policy. If a remote peer is *choked* it means the local peer will not upload data to it. If it is *unchoked* the local peer is ready to upload data to the remote peer.

Every 10 seconds the peer evaluates the downloads of the last 20 seconds ("rolling 20-second average" as defined by Cohen [17]) and decides to whom it will upload. The decision of who to upload to is based on the upload rates of all the peers that transferred data to it on the last 2 cycles of evaluation (20 seconds). This *tit-for-tat* strategy encourages cooperation between the peers and prevents *free riding*[5]. Seeds apply the same principles in order to select peers to unchoke, but since they are not downloading they do so based on their upload rate.

Each peer seeks to maintain connections to a set of 20 to 40 peers (also known as the *peer set*) and from these a maximum of 4 will be unchoked (meaning it will upload through that connection). Every 30 seconds the peer does an *optimistic unchoke* and a random connection is unchoked regardless of the upload rate it offered. This way the peer might find connections that have better rates than the current ones.

Despite the fact that each peer can at a given moment be uploading to a maximum of 4 peers, it can be downloading from more than 4 other peers. This is because choking is not necessarily reciprocal and the evaluation cycles of the peers are not synchronized [26].

If a peer, that we were previously downloading from, does not serve any piece during 1 minute, the peer considers that it is "snubbed" and does not upload to it unless that peer becomes selected in a *optimistic unchoke*.

In order to prevent selfishness, the protocol gives an equal amount of bandwidth amongst the 4 established connections [30]. This sacrifices performance in such a cooperative environment. In Cohen's paper [17] he empirically suggested the number of simultaneous connections to be 4 in order to "allow TCP's built-in congestion control to reliably saturate the upload capacity". Although it works well when there are a lot of pieces in the system,

[5]Instead of contributing to the overall download of the file a Free Rider does not upload at all or at least not as much as he downloaded from the network.

better bandwidth allocation algorithms have been proposed with BitMax [30].

### 2.1.4 Piece Selection

The content to be exchanged by the system is divided into pieces and each piece into blocks. The decision of which pieces to exchange over a connection is made using a *local rarest first* policy. Since the peer knows the availability of the pieces of each other peer he is connected to, it can calculate what piece is the rarest amongst its peer set. By downloading the rarest pieces first, it ensures that the rarest pieces will become more popular and diminishes the probability of the content becoming extinct in case the *seed* leaves. Every time a peer receives a new piece it checks its data integrity by calculating the SHA1 hash and comparing it with the hash that is in the "torrent" file. If it passes the test the peer informs its neighbors of the new piece it has.

One exception to this selection algorithm is when the peer first connects and has no piece available yet. In this case, the selection for the first piece is random. The arriving "empty" peer will only be unchoked due to the existence of *optimistic unchoke* policy. After the first piece arrives the mode changes to Rarest First.

Another exception is when the download is almost complete. Since there is a tendency for the arrival of the last pieces to be slow [2] a *End Game* mode is implemented when approaching the end of the download. The peer will send requests over all its connections for all of its missing pieces. Whenever the peer receives one of these last pieces he cancels the request to all other peers who have been asked for it. Since the *End Game* period is very short [17] there isn't a lot of bandwidth wasted and a quick finish of the download is enabled. A standard definition of when *End Game* mode is supposed to start does not exist. Research identified[6] two main alternatives about when to start the End Mode:

- All blocks have been requested.
- The Number of blocks in transit is greater than the number of blocks left, and no more than twenty.

Piece size should be chosen based on the size of the content but taking into account that

piece sizes too large cause inefficiency, and too small will result in a large metadata file. The most common sizes are 256 kB, 512 kB, and 1 MB [2]. Every piece is of equal size except for possibly the last one. Pieces are subdivided into blocks, that are the units that are sent over the wire. Block size is typically 16Kb.

## 2.2 IP Multicast protocol

The well known TCP/IP protocol only provides a unicast transmission possibility. Meaning that over a TCP/IP channel one host can only communicate to one other host and no more (over that channel). It is a point to point communication. Applications that need to send the same information to multiple recipients using TCP/IP must do so by replicating the same message to the intended audience. One can easily see how inefficient this solution can be. In order to better address this issue, the Multicast protocol was created.

Although it was invented in the 80's, IP Multicast [19] was presented as a doctorate paper in December 1991 by Dr. Steve Deering and subsequently published as RFC (Request for Comments) 1112 - Host extensions for IP multicasting. Its first real use was in March 1992 over the Multicast Backbone (MBone) network to transmit the audio stream of a meeting of the Internet Engineering Task Force (IETF). The MBone was an experimental Internet backbone for use with IP Multicast.

IP Multicast is designed and implemented in a way that group management is separate from routing and has different protocols and ways to deal with each of these issues. The first Multicast solution was made of 2 distinct protocols: IGMP (Internet Group Message Protocol) and DVMRP (Distance Vector Multicast Routing Protocol). IGMP handled the arrival and departure of machines to Multicast groups. DVMRP was a Multicast Routing Protocol that enabled cooperating routers to exchange information in order to enable the connection of recipients to multicast sources. Due to the its limitation of only being able to handle 32 hops [46], DVMRP was not ready for the world wide web, so two other Multicast Routing Protocols were proposed: PIM (Protocol Independent Multicasting) and MOSPF (Multicast Open Shortest Path First). PIM has several variants: Sparse Mode (PIM-SM), Dense Mode (PIM-DM), Bidirectional and Source Specific Multicast (PIM-SSM). Amongst

these PIM-SM has been the most widely used.

When delivering the same network packets to multiple recipients IP Multicast allows for a better use of the network than unicast. As shown in figure 2.2, instead of sending a individual packet to all recipients it creates a copy of the packet only when the physical link to the multiple destinations splits. The sender only sends one packet and it will be received by multiple receivers. It does not require that the sender has prior knowledge of where or how many receivers there are. This fact brings a tremendous benefit in terms of overall used bandwidth since the number of sent packets is the same regardless of the number of users that are connected to that transmission. Reducing the used bandwidth on the links also leads to a lesser probability of having network congestions.

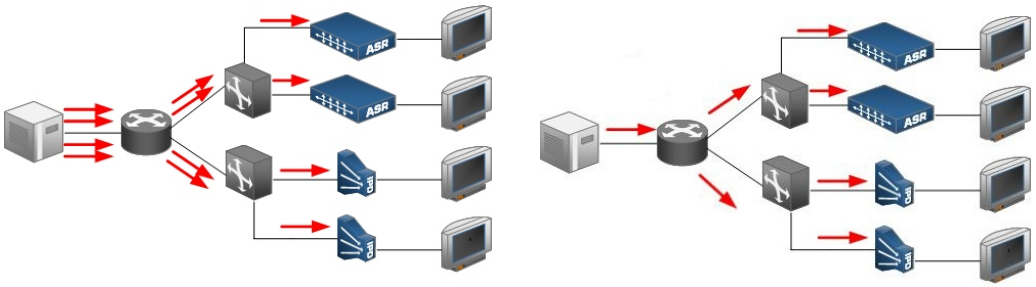


Figure 2.2: Unicast versus IP Multicast

### 2.2.1 IP Multicast Limitations

At the application level, IP Multicast is actually a particular case of UDP (exposed like unicast UDP). And, like UDP, the transmission is not reliable and duplicate or loss of packets can occur. It also does not have a congestion control mechanism and allows for clients to join a IP Multicast group that is transmitting at a rate that is higher than the client's networking capabilities[46].

IP Multicast is enabled by using a notion of group of recipients that are interested in receiving packets sent to the group. Any host that is interested in joining the group must use Internet Group Management Protocol (IGMP). Since IP Multicast allows for an arbitrary host to send information to any group [18] (many-to-many communications) some issues might arise when using it. Wrong or fake data can easily be sent to the IP Multicast address. There is no authorization or authentication means to control the communication. There are

already some alternatives that support one-to-many IP Multicast applications. One of those solutions is EXPRESS: EXPlicitly REquested Single-Source IP Multicast [23]. Using standard IP Multicast might be more applicable on a case where the solution is controlled by the provider from source to peers (like a cable operator with set-top-boxes). This is the case we suggest in chapter 7.

IP Multicast was most promising when it appeared but its adoption was initially not as expected due to several issues. Some of the issues were the inexistence of a pricing and business model to support its commercialization, its complicated architecture and, more important, its open group management strategy [38]. Another disadvantage arises with scalability issue of IP Multicast routing in large-scale networks such as the Internet. Anyway, nowadays the tide is turning and more and more people are using IP Multicast to build large scale content provider networks, deliver IPTV and radio [3] among others.

As stated, the monetization of Multicast was one of the barriers to its adoption. Some ISPs base their cost on the bandwidth consumption and this model is not suited to charge Multicast traffic. This is because the ISP must endure a much higher effort to deliver a Multicast packet than it is to send a Unicast packet.

Due to these limitations: low security and content emission control, lack of business model and scalability, IP Multicast is not widely available over the Internet. But there is no particular reason why one cannot use IP Multicast on a controlled or private environment.

# Chapter 3

# Related work

In this chapter, we present projects that share a common goal with our work: the distribution of content. We first analyze the different approaches to solve the issue at hand and then a few selected alternatives for each one.

We see 3 main approaches in content distribution solutions that go beyond a single server or a single protocol to fulfil the goal:

- Infrastructure based solutions
- P2P
- Hybrid

We'll also address the issue of *application level multicast* since IP Multicast is one of the base protocols of our work.

## 3.1 Infrastructure based solutions

Infrastructure based solutions use the typical client-server model plus a "brute force" solution to try to solve the problem. They work by distributing the burden of serving content across multiple servers that have a copy of the file to download. Depending on the number of clients that the solution needs to cope with, it can be implemented by one or more

servers on a balanced server farm or by the means of a *content distribution network*. The foremost alternative (centralized servers) has scaling issues and might not be prepared for *flash crowd* scenarios and the latest is only feasible for large clients with a commercial distribution model that is able to pay the costs.

*Content distribution network* services have been commercially available for some time by companies such as Akamai, Limelight or CDN Networks. They are the most commonly used way to distribute commercially managed popular content over the Internet. CDN servers form an overlay network on top of the Internet, with their servers acting as proxy caches that serve copies of the content from the server that is closest to the end user. This way CDN's reduce the bandwidth requirement on the central server and deliver a better download time. They take care of transparent server replication and mirroring that enable scalable and highly available content distribution systems. If the load can be predicted these systems can work very well but are typically expensive to deploy and maintain. Some of the most successful content and video sites like CNN, Youtube and Hulu use this approach. Figure 3.1 depicts a CDN layout.

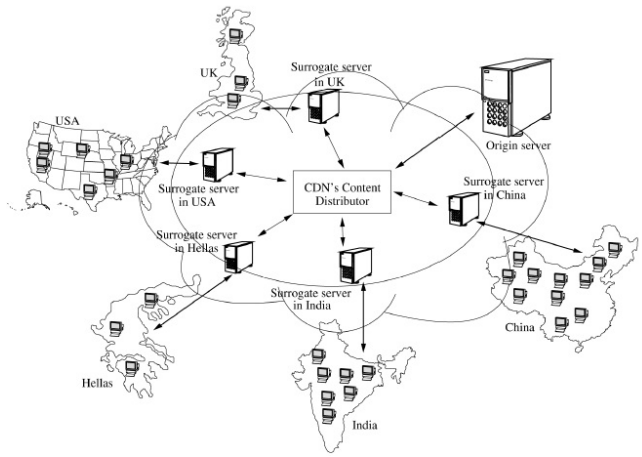


Figure 3.1: Content distribution network example [29]

## 3.2 Peer-to-peer

Due to the increase in number of users connected to the Internet, the bandwidth intensive content and the associated costs and limitations of CDN's, P2P networks are viewed by

many as an alternative to CDNs [25]. Despite its unreliability, several studies support the prospect that P2P is an excellent alternative to centralized solutions or CDN's, because of its ability to serve large files at reduced cost, self organization capability, bandwidth scalability and network path redundancy [28].

The first P2P solutions like Napster, Gnutella and BitTorrent were aimed at file sharing, but recently P2P has also been used for other services like streaming of content and enabling *End Host Multicast*. Instead of relying on a IP Multicast infrastructure at the network level these *End Host Multicast* systems use an overlay network to emulate a IP Multicast service type. These types of systems will be analyzed in the next section.

We further analyze two P2P content distribution solutions (Slurpie [43] and GridCast [14]) and also an *End Host Multicast* solution called Narada [16]. Slurpie is very similar to BitTorrent and addresses the issue of downloading a whole file while reducing the load from the Source Server. Gridcast is a media streaming solution that works on top of a P2P overlay network.

### 3.2.1 Slurpie

As other content distribution solutions, Slurpie [43] tries to solve the issue of handling multiple concurrent users downloading popular content from a central server. It does so by implementing a P2P protocol that acts separately from the source server. The protocol addresses the transfer of files as a whole, and aims at reducing the load on the source servers and the download time of popular files. Slurpie assumes that the data transfer rate of the server is the bottleneck and that the clients are willing to share their resources for the overall common good. So, Slurpie tries to achieve its goals by using the spare bandwidth and processing power of the clients (the basis of P2P) instead of saturating the server.
The goals of Slurpie are "to be":

- *Scalable*: The load on the server should be independent of the number of clients on the network, allowing for a maximum of one million simultaneous users.
- *Beneficial*: Reduce the download time when compared to direct download from the server.
- *Deployable*: easy to deploy without infrastructure support (no need for router or source server changes).

- *Adaptive*: Adapt the client's download rate to different network / processing capacity.
- *Compatible*: Easy to integrate with existing data transfer protocols like HTTP and FTP.

Like other P2P solutions the resources focused on data transfer are directly proportional to the number of clients that want that data, thus enabling the scalability of the solution. The system is only worried with the transfer of the overall file, thus Latency and jitter are of secondary importance. It also does not guarantee data integrity, which it must be implemented at the application level.

#### 3.2.1.1 Architecture

As it can be seen in this section Slurpie is very similar to BitTorrent. It uses a central server known as *topology server* that acts as a meeting point of the clients. The server's role is similar to that of the tracker on the BitTorrent network. Its topology is randomly generated. It also splits the files in parts (known as blocks) and the exchange of data is made at this level. Slurpie does not have a goal of maximizing network-level efficiency. It does not work in order to have a good topology that would reduce the P2P connections like Multicast protocols do when creating, for example, a shortest path tree.

Unlike BitTorrent, Slurpie allows the clients to download from a source server outside of the Slurpie network, typically over HTTP or FTP. Traditional content distribution solutions, or even Multicast solutions, always retrieve content from the same location while Slurpie dynamically decides where to fetch the content based on network conditions.

Slurpie coordinates group downloading decisions without a central server. Slurpie protocol is able to independently take decisions, like the number of peers to keep in the mesh, the information propagation rate and the number of simultaneous data connections.

The mesh formation procedure is also very similar to that of BitTorrent. The *topology server* (tracker in BitTorrent) keeps information regarding the last $x$ nodes that queried the server but replies to queries with the last node that made a query to it. The logic is that the last node to query the server is the one with highest probability of still being alive. After knowing of this first node, the joining node starts to receive updates from it and starts to discover new peers participating in the mesh. After having an initial set of nodes it starts to create links

to a random subset of these nodes. The number of nodes to connect to is decided at runtime according to available bandwidth.

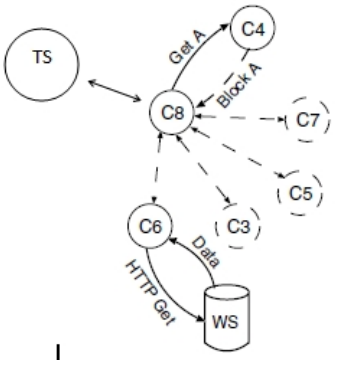


Figure 3.2: Slurpie peer communications

Like BitTorrent, Slurpie also uses a bitmap of available blocks at each node. Each client knows the bitmap of availability of a certain subset of peers. By doing a tree of logical OR's of each pair of nodes, it is able to locally identify which blocks are not available from it's peer set. If a given piece is not available on its neighbors a *backing off* procedure is implemented: the peer can only connect to the source server according to a certain probability 3/n (where n is the estimated mesh size). This will keep overall server throughput despite the group size.

An AIMD (Additive Increase and Multiplicative Decrease) flow control is used in order to decide the number of messages to be exchanged with the neighbors for propagating updates. If the peer has under utilized bandwidth it should augment its knowledge of the world so it exchanges more messages. If, on the contrary, he is well using its network capacity it does not need to know about more peers since he cannot handle any more traffic.

In order to manage its connection, Slurpie runs a bandwidth estimation technique, once per second, that returns one of 3 states: *underutilized*, throttle-back, and at-capacity. The sum of used bandwidth over existing connections is compared to a moving average of the last sums and one of the 3 states is depicted according to the relation of the sum and the average. If the state is *underutilized*, and there are neighbors that have blocks that we need, a new connection if opened. When multiple peers have the block that we need, a random selection is made.

#### 3.2.1.2 Comparison to BitTorrent

Slurpie tries to address the issues that already have been studied and somehow solved by BitTorrent (and other P2P solutions). Some of these issues are: last block, scalability of the meeting point, division of the file in blocks, number of connections per peer and how the mesh is formed.

Slurpie innovates on the possibility of enhancing a possibly problematic content distribution solution without changing it. It does this by incorporating the source server into the solution. And also on the ability of keeping the load on the source server independent from the size of the peer set downloading the file. So, a new problem specific to Slurpie is the decision of when to download from the Slurpie network or from the central server. The source server (HTTP or FTP) in Slurpie can be seen as a seed in BitTorrent, but the benefit of this solution is that the server does not need to know about the P2P solution that was implemented. The source server only needs to be able to serve data at the block level.

Slurpie simulations were run on PlanetLab and on a local environment for the download of a 100 MB file. It is stated that "Slurpie out performs BitTorrent, with respect to both average download times and also download time variance.". We think it is hard to believe that Slurpie outperforms BitTorrent. The simulation test were run a small number of clients (<50) not comparable with the current number of clients that BitTorrent serves in the real world with such a great success. Also, the similarities of the protocol are so many that we should expect the results to be side-by-side. The seeds in BitTorrent act no differently than the source server in Slurpie, and in BitTorrent there are typically many seeds. Slurpie was only found on academic work with no real world implementation. BitTorrent has real usage scenarios and has, without doubt, proven its benefits. It can be seen by the wide adoption of BitTorrent while Slurpie was not.

Slurpie incorrectly states that "BitTorrent does not adapt to varying bandwidth conditions". BitTorrent is always adapting to network conditions due to its tit-for-tat strategy.

### 3.2.2 GridCast: P2P VoD Streaming

GridCast [14] is a P2P Video on Demand streaming solution. Gridcast streams multimedia content from source servers but extends the solution by also using a P2P network for data exchange. While the peers are consuming media, they are also actively participating in the exchange of information between themselves. Like Slurpie, Gridcast tries to protect the source servers from being overloaded and only allows the peers to make connections to them on certain conditions.

Factors like the high churn rate (frequency of peers joining and leaving the system) and the typical end-user asymmetric links with reduced upload capabilities present a great challenge for the successful implementation of a VoD streaming solution over a P2P network.

Gridcast has 4 main components as it can be viewed in figure 3.3: The central servers that store a copy of every video file, a tracker that indexes all peers on the network, a web portal that provides the list of channels and finally the Peers that whenever possible exchange data amongst them selves and only connect to the source servers if the desired data does not exist within its partners or they are not able to serve the peer with the desired quality.

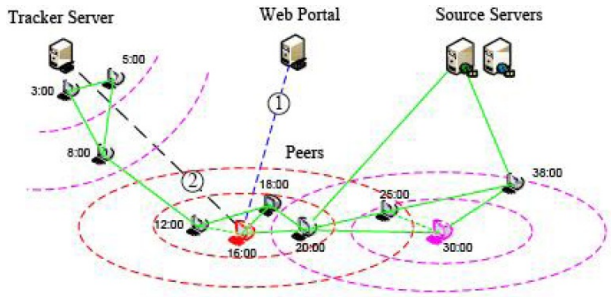


Figure 3.3: GridCast Architecture

Unlike other content distribution solutions the goal here is not to download the whole file but to enable a great media viewing experience. Being a Media viewing experience peers are only interested in contents after its current playing position. This position is typically different from most other peers that might be watching the same video. So, each peer can typically only contribute to its partners (the list of up to 25 peers that he might be connected to) that are before him in the playing position and can only consume from partners that are ahead of him.

Since the peers can, and often do, change the playback position a Video on Demand system must cope with this behavior. The role of the tracker is to keep the list of peers that are watching a certain video, and it groups peers accordingly to their playing position. These groups are created as concentric "rings" distanced using relative playback positions of the peers. Knowing these rings the tracker can inform the list of peers that have playing positions close to a certain spot in time.

The file being served is split in chunks representing one second each. Peers select partners to start exchanging chunks based on the distance to him regarding the playing position, meaning in the innermost ring, and then start going outwards in the rings or chose the source server otherwise. The peer needs to be able to download 10 seconds before starting to play the video. Playback is not started before this condition is met. When the first 10 seconds arrive he then tries to download the next 200 seconds.

In order to enhance the in-file seek experience, and only if the available bandwidth allows it, the peer will try to pre fetch contiguous blocks of 10 seconds from the video. These blocks are known as *anchors* and as it can be seen in figure 3.4. They are spread in fixed intervals (typically 300 seconds) throughout the file. When the user wants to jump to another viewing position inside the same file, the position is forced to be one of the closest anchors to the desired position [12].

The system's performance is directly related to the availability of peers watching the same content and preferably on a position close together or the central servers will be stressed out to answer to a set of sparse peers.

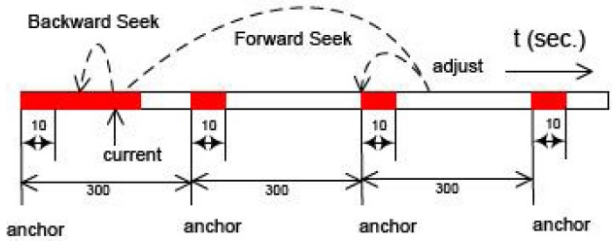


Figure 3.4: GridCast Anchors

When compared to file downloading P2P solutions, P2P VoD streaming is more complex and less studied. Despite that, GridCast was deployed in May 2006 in China and the perfor-

mance of utilization in a real case scenario was already studied in [13]. Analysis shows that it is feasible to implement a cost effective P2P streaming solution that delivers an acceptable user experience even with limited concurrent users.

The user experience is acceptable but still far from what we are used to on a TV set. Playing of video typically starts only 10 seconds after a peer joins a channel, completely disabling the typical channel zapping attitude.

## 3.3 Hybrid

A Hybrid solution is one that combines two or more existing solutions to form a new one. In this section we present the only solution we found that takes the same approach as Bitocast, that is, combining IP Multicast and BitTorrent for the distribution of content. That solution is called Delco [39].

### 3.3.1 Delco

Like Bitocast, Delco [39] also proposes a solution that uses IP Multicast and P2P techniques in order to deliver content to a very large group of users in a reliable manner. It enables the delivery of files as a whole by splitting the file in blocks and serving the blocks via IP Multicast and BitTorrent. It also recognizes the subtilization of IP Multicast but suggest its use at local branches to improve network delivery efficiency.

The main requirements in the development of Delco were that the system should survive in *flash crowd* scenarios, be scalable, enable the use of Digital Rights Management (DRM), be secure enough to resist malicious attacks and content tampering, and be able to create a distribution network that is independent from the access network and end systems.

Delco supports three different modes of content delivery: IP Multicast- only, combined IP Multicast and P2P, and P2P only. It is flexible enough to exchange the delivery mode according to the clients capabilities (supporting IP Multicast or not) or in regard to the popularity of the content being distributed and the number of clients connected to the IP Multicast server.

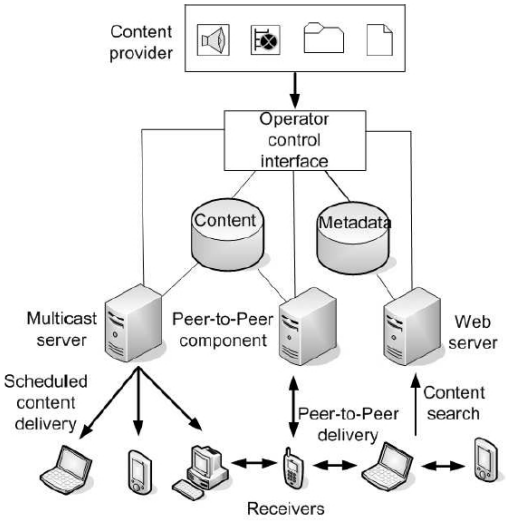


Figure 3.5: Architecture of the Delco system

Delco defines two service types that can work with any of the previously presented delivery modes: Download and Channel. The download mode is used to serve content that the user explicitly shows the wish of having or viewing. In the channel mode the client subscribes a channel and downloads whatever content is published to that channel, much like TV. The download mode is capable of serving content that is completely available before download begins (a static file or group of files), known as "Progressive Download", or stream content as it becomes available, known as Live Streaming. In the case of "Progressive Download" the Delco client is able to play media files while downloading. It is not stated how Delco achieves the ability of playing content since downloading out of order blocks with the purpose of completing 100% of a file is much different than downloading content to show in real time.

It is also mentioned that the system does not send the same blocks over P2P and IP Multicast. But it is not stated how the issue is handled.

Like BitTorrent it also uses a web server for users to search content and obtain the metadata that will enable them to use the Delco service.

While Bitocast is tightly coupled to BitTorrent, Delco's architecture enables it to work with any multicast and P2P technique. Delco was implemented using File Delivery over Unidirectional Transport (FLUTE) [36] and BitTorrent. FLUTE is defined in RFC 3926 and is "a protocol for the unidirectional delivery of files over the Internet, which is particularly suited

to multicast networks". One of FLUTE's characteristics is its ability to scale due to its lack of uplink signalling.

The content tampering issue is solved in a more or less naive way by limiting the content injection in the network through the operator control interface. Delco also implements a block alignment mechanism to deal with the differences in the block sizes of the Flute and BitTorrent communications. In contrary to BitTorrent Delco takes care of the content discovery phase.

In terms of the efficiency of Delco there is little or no information. The available paper does not give any metrics on how Delco performs in the distribution of different type and sizes of contents and across different client bandwidth scenarios. It is too focused on implementation and not on results. It also does not refer how it selects pieces for IP Multicast distribution or how it performs in multiple conflicting files distribution.

In overall, by dealing with Digital Rights Management, Mobile users, protocol independence and security Delco aims at a broader target than Bitocast. It requires a heavier infrastructure than Bitocast and does not show working results, real or simulated in order to evaluate its benefits and feasibility. The paper also does not address how Delco solves some very important issues like block serving model and block selection model for streaming media.

## 3.4 Application level Multicast

Some of the reasons for the barriers to multicast adoption were the difficulty to deal with stateful per group management and its implications on the routers. The idea behind solutions like Narada [16] and others, is to shift multicast responsibilities from the routers to the end systems that have spare resources and are willing to cooperate and potentially solve ip multicast problems at another network level. These types of solutions are known as *application level multicast* in which the peers are responsible for managing the groups, routing and forwarding the packets over an overlay network. Obviously there is a performance overhead in this approach because, unlike pure IP Multicast, duplicate packets might occur at the physical network links, causing higher network usage than IP Multicast.

In *application level multicast*, state is kept at the peers thus resolving the problem of scalability of routers. These types of systems also do not require a global naming convention and can be application specific, easening a well known burden of IP address distribution.

*Application level multicast* can be implemented over unstructured or structured overlay networks. In a unstructured overlay network, nodes connect to the network "according to measures unrelated to content, such as join-order, connection speed, and even physical proximity, creating a random connection topology" [21]. In a structure overlay network, such as *distributed hash tables*, the connections are based on a specific property of the network logical address. In this section we present an overview of an unstructured and structured overlay network, namely Narada [16] and Splitstream [11].

#### 3.4.0.1 Narada: A Case for End System Multicast

Narada uses a overlay spanning tree for data delivery. The tree construction is made in a two step process. First it creates an arbitrary mesh of the complete virtual graph that connects all peers in the system. Then it constructs a shortest path spanning tree, per source, using standard routing algorithms.

The responsibility of managing group membership is shared amongst all members. There is no central group coordination. This decentralized coordination is possible by having the

members periodically exchange its knowledge of the group with its neighbors. With the exchange of this information each member tends to know about all members of the group. Associated with each member's update message is a sequential number. This sequential update number enables other members to decide whether the member is dead or partitioned when an update message is not received for some predefined "sequence time".

Bootstrap of the members is not part of the specification. Narada assumes that somehow a joining member is able to get to know of at least one active group member. After obtaining the first list of members the joining member randomly connects to a subset of the members in the list, asking each one to be recognized as a neighbor. When accepted as neighbor by one of the members, the joining member becomes part of the group and its existence is reflected in the regular group membership messages from there on.

When a member leaves the system it should notify its neighbors of the fact. If it crashes or fails to inform, the neighbors will sooner or later notice that the member is not sending update messages. Other members will discard the missing member from the list and start informing their other neighbors of the fact. When a member that is the sole responsible for a link in the graph leaves, the mesh is partitioned. In these cases the members became aware of the fact because they stop receiving update messages from the members on the other side of the partition. When it happens they start a repairing process that is feasible because the members keep a list of zombie or potential dead members and try to connect to them one at a time, hoping to reconnect the partitions of the mesh. There is a probabilistic approach that only enables a small set of members to try to repair the mesh.

Narada tries to refine the mesh in order to create a better graph of overall connections. The graph is not optimally constructed because there is no initial condition on the establishment of connections when the members arrive. Even if there where those conditions they would be changing over time due to the arrival and leaving of members, the partitioning of the mesh and the changing network load conditions. To create a better graph the members periodically probe each other randomly in order to find better connections. They also monitor existing connections in order to drop any connection that is not being useful. The system will not drop a connection that would lead to a partitioning situation.

Narada uses a distance vector protocol in order to create a least cost routing table between all members. The cost is the latency between the members thus allowing the system to adapt to changes in the network.

It is a fact that an overall multicast solution cannot out perform IP Multicast because it works one layer above. Despite the fact that Narada is capable of shifting multicast support from routers to end systems with small performance penalties when used with small and sparse groups. It is not suited for applications like Internet TV [16] but enables the use of some type of Multicast solutions despite the unavailability at the Internet IP level. Besides that, it also nicely solves the issues of system failures and changes of group members and was one of the first papers to address self-organizing and self-improving protocols to work with overlay networks.

#### 3.4.0.2 Splitstream

The *application level multicast* solutions that are based on a single distribution tree, where each node is responsible for forwarding information until the leaves, are unfair. In such cases the interior nodes carry most of the forwarding burden, while the leaves only receive information and do not need to share it. Splitstream [11] tries to solve the issue of unbalanced forwarding load in conventional tree based multicast systems. It does so by dividing the multicast content into several pieces or stripes and distributing those using separate trees. Fairness is attained by ensuring that each node is an interior node in a number of trees corresponding to the number of stripes it is willing to forward and that it is a leaf node in the remaining ones.

According to the authors of SplitStream, and based on their simulations and live experiments, the overhead introduced by having multiple multicast trees is low and the overall performance is good, even for highly dynamic groups.

# Chapter 4

# Bitocast

Bitocast is a hybrid content distribution solution that combines BitTorrent and IP Multicast to achieve its goal. The BitTorrent client is modified in order to, not only receive data via BitTorrent, but also receive pieces from one or multiple IP Multicast channels. The BitTorrent protocol is unchanged. Only the client is enhanced in order to allow for this duality, thus allowing for an easy enhancement of existing solutions without interfering with the ongoing distribution.

In this chapter we delve into the inner workings of Bitocast.

## 4.1 Overview

The system is composed of the same entities as in the standard BitTorrent network, plus one or more IP Multicast servers. A minor change is made to the "torrent" file in order to add information regarding the IP Multicast server. This information is the IP Multicast group(s) address, the respective bandwidth and pieces that are sent on that channel and its schedule. The "torrent" file is obtained in the same way as traditional BitTorrent. This is the way to inform the location of the IP Multicast Server to peers. An overview of the system is shown in figure 4.1

The client can decide what bandwidth to allocate to BitTorrent and to IP Multicast, from 0 to 100% working as a standard BitTorrent client (0%), receiving exclusively from IP Multicast

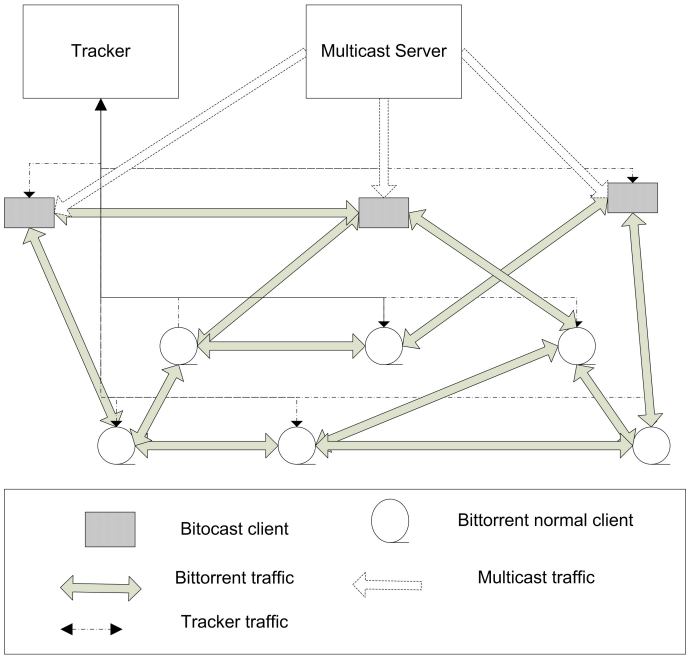


Figure 4.1: Bitocast

(100%) or as a hybrid.

When the system is started, the client normally connects to the BitTorrent swarm network, but it can also connect to one or more IP Multicast channels that are delivering pieces of the "torrent's" content at that time. When connecting to IP Multicast channels the client also needs to take into consideration the percentage of its bandwidth that was reserved for IP Multicast communication. So, the client is also a normal node for traditional BitTorrent peers, downloading and uploading from them as any other.

## 4.2 IP Multicast communication

As we've seen before, IP Multicast enables a much more efficient usage of the physical network but has some limitations. These limitations, namely the unreliability of the protocol, can be overcome by combining IP Multicast with BitTorrent. A piece that is sent by the Multicast server, that is lost and never reaches a peer, can afterwards be recovered over the BitTorrent protocol.

When a BitTorrent client receives a piece from one of its neighbors, it sends a *have message* to all connected peers, in order to inform them of the fact. Should the same procedure occur

when a new piece is received from the IP Multicast server ? In the standard BitTorrent implementation the management overhead of the protocol can reach 23% [31], and a big part is the responsibility of the HAVE messages, leading to a situation known as *HAVE message storm* [27]. In order to reduce the *HAVE message storm* effects, an extension of HAVE messages has been proposed in [27]. Since a big part of the peers in the swarm network are potentially connected to the same IP Multicast servers, these HAVE messages would, in a significant percentage, serve no purpose. So, we suggest that Bitocast should also use the HAVE extension to avoid the storm that is even stronger on our scenario. In Bitocast, the update period of the HAVE extension should be even larger than the proposed time frame for standard BitTorrent. This is because the probability of several peers receiving the same piece is higher in Bitocast than BitTorrent. If any blocks of the received piece were previously requested over BitTorrent protocol to one of its peers a *cancel message* should be sent informing of this fact.

Since the presence of *seeds* is essential and of utmost importance for the upload capacity of the network [26] and the bandwidth of the original *seed* a precious resource [9], we need to optimize its use in order to have the best possible distribution of content from the source. IP Multicast will help in this optimization. The IP Multicast server will start providing the content as it becomes available and at least once, in order to guarantee that enough *seeds* exist on the network. After that first round, it can move on to serve other content or keep on sending that same content for a certain number of times.

The existence and ways to overcome malicious users, that could easily send fake data to the IP Multicast channel, were not considered on our work.

### 4.2.1 Channel Address and Synchronization

Since a IP Multicast server will not be transmitting the same content forever, each IP Multicast channel has a window of time when it will be "on air" transmitting pieces of that specific content, as defined in the schedule window of the "torrent" file. If the client connects to the IP Multicast channel after that period of time, it will download pieces that are from another content and will not pass the SHA1 hash validation process. In IP Multicast communica-

tions the pieces of the content are disassembled at the source and sent in several network packets. They are reassembled on the recipient and put together to form the original piece. After the piece is received the client will also perform a SHA1 hash validation. Knowing that IP Multicast will work in a controlled environment and in IPv6, the successor of IPv4, the address space is very big, each torrent will be served by a IP Multicast server with a unique address. The IP Multicast server will have multiple IPv6 addresses according to the torrents it is serving.

### 4.2.2 Multiple IP Multicast Servers

As the unofficial feature of BitTorrent called Multitracker[4], which supports the existence of multiple *trackers* in order to abolish the single point of failure (aka *spof*) of BitTorrent, Bitocast could also support multiple IP Multicast servers specified on the "torrent" metadata file. This would also abolish the *spof* of Bitocast regarding IP Multicast communications. The selection of the IP Multicast server could be based on the distance from the server (hop count) in order to reduce network traffic. Besides eliminating the single point of failure it would also allow for a higher redundancy and possibly scalability of the solution.

Like John Hoffman's proposed extension to BitTorrent metadata format, that enables the existence of multiple trackers [1], Bitocast "torrent" file would include a "mc-announce-list":

```
d['mc-announce-list'] = [ [MulticastServer1], [BackupMulticastServer1], ... ]
```

## 4.3 IP Multicast server

Despite the average client's bandwidth being increasing over the years, there are still many clients that do not share such a high bandwidth. Low bandwidth conditions might exist due to several reasons such as long distances of ADSL lines, commercial packaging of internet access products and the appearance of a lot of new devices and applications, namely mobile, that also start to participate in BitTorrent (SymTorrent [24] is one of those examples). Also, since BitTorrent does not take into account physical location of peers, there can be a great mix of clients connected in the overlay network, from several parts of the globe and with very different bandwidth capabilities.

In order to find the best fit between the clients download bandwidth and the bandwidth offered by the IP Multicast server, Bitocast supports several channels over which the content can be emitted. If these channels use several bandwidth configurations, the possibility of filling the slot reserved for IP Multicast communications is increased. For example if the IP Multicast server offers channels with the following bandwidths (Kbps): 2048 , 1024, 512, 256, 64 a client that has only 900 Kbps allocated to IP Multicast communications might be able to connect to the 512, 256 and 64 channels, summing up 832 instead of just 512 Mbps. For this reason Bitocast can be configured on how IP Multicast bandwidth of the server is split amongst channels[1]. It is obvious that serving the same content over several IP Multicast channels will create redundant packets on the network, which is something that Bitocast is trying to reduce. But, when it happens, it is done in a controlled and intentional manner with the goal of creating better download speeds.

Since we can have multiple channels serving the same content, each IP Multicast channel must also define what pieces of the "torrent" it sends (Piece Model). The Piece Model defines what pieces of the content are split over each channel. The IP Multicast channel can also send the pieces that it is responsible for in several ways (Scheduler Model). The Scheduler Model defines the way the next piece to be sent by the IP Multicast channel will be selected. In the following sections we will further explain the Piece and Scheduler models.

[1] channel or IP Multicast group refer to the same thing.

### 4.3.1 Piece Model

As depicted in figure 4.2, the **Piece Model** has the following possibilities:

- *Multiples* or *Interleaved*, The pieces that belong to each channel are defined in multiples of the channel ID (Channel A among 10 channels will send piece 1,11,21,31... for example).
- *Interval*, Each channel sends contiguous blocks of pieces of the file.
- *Full*, Each IP Multicast channel serves the full file.

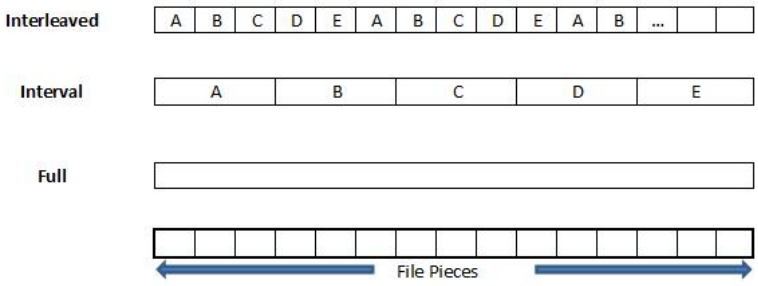


Figure 4.2: The way the file is split amongst IP Multicast channels according to the *Piece Model* (If there were 5 channels, A to E)

If the *Full* piece model is used the Bitocast client should not connect to more than one channel because different channels would be sending the same content (at different speeds) and duplicate pieces would be received.

One can see the graphical "fingerprint" or signature of each Piece Model when looking at the global availability chart of pieces on the network for each model.

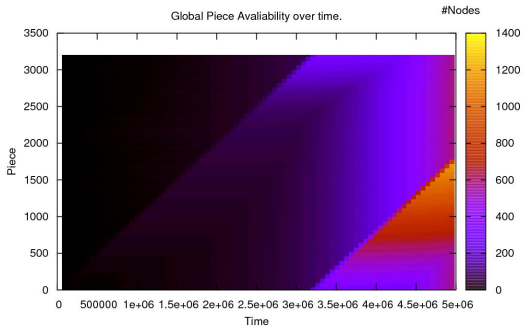


Figure 4.3: Piece Model: Full

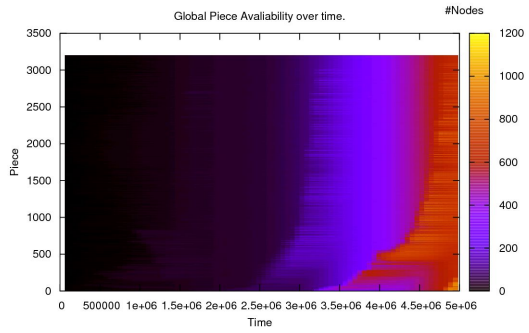


Figure 4.4: Piece Model: Multiples

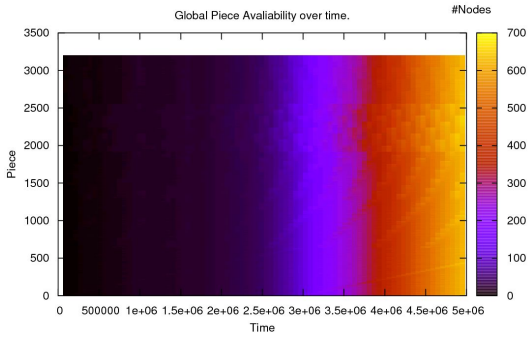


Figure 4.5: Piece Model: Interval

On the simulation results shown in figure 4.3 we see that one IP Multicast channel is sending data from the beginning until the end (in a round robin pattern), with the availability of pieces increasing significantly as pieces are served by the IP Multicast server.

In figure 4.4 we notice the interleaved pattern of distribution by having regular intervals of piece availability. In this case several channels were used. It is a more uniform mode then Full mode, in a sense that all pieces have more or less the same availability, but it is not as efficient.

In interval mode shown in figure 4.5 each range of pieces is sent by a different channel. In this case, each channel has different bandwidths resulting in different patterns of piece distribution. But in the overall it tends to be a even more uniform piece distribution mode but not as efficient as Full or Multiples mode.

### 4.3.2 Scheduler Model

The **Scheduler Model** has the following possibilities:

- *Round Robin*, sends pieces in a Round Robin fashion.
- *Random*, sends pieces in random order.
- *Rarest*, sends the rarest pieces available in the swarm.

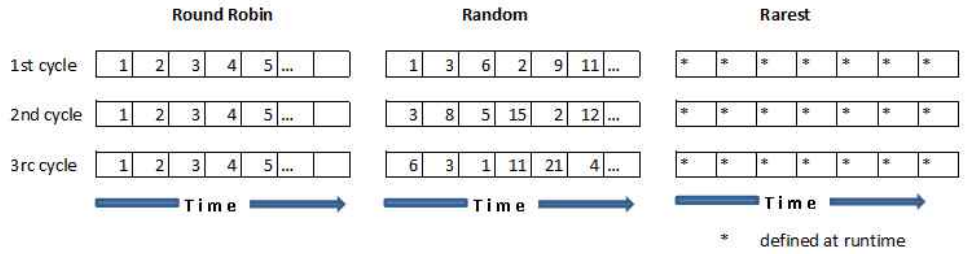


Figure 4.6: The way the pieces are sent over a IP Multicast channel according to the *Scheduler Model*

*Rarest* Scheduler model sends the pieces based on their availability on the swarm. In order to implement this functionality, the *tracker* needs to be enhanced to receive the bitmap of piece availability every once in a while. If there is more than one *tracker* (Multitracker) the notion of rarest would be the same as it is applied at the node level, meaning that it is the local rarest based on the pieces that the *tracker* knows.

## 4.4 Integration of Bitocast with BitTorrent

One of our goals was the coexistence of Bitocast with BitTorrent. Bitocast can be used to extend an existing BitTorrent file distribution solution without causing any interference.

In Bitocast the "torrent" file needs to have more information than in the traditional BitTorrent solution. That information concerns Multicast communications. The standard "torrent" file contains the information shown in table 4.1. One could propose a new entry to the "torrent" file in order to include the needed specific Bitocast information. But since that might break compatibility with existing clients, we propose to include that information inside the "torrent's" free comment section. This allows for both Bitocast and BitTorrent clients to obtain

| Key | Description |
|---|---|
| Info | A dictionary that describes the files |
| - length | Length of file in bytes.(integer) |
| -md5sum(optional) | A 32 character hexadecimal string corresponding to the MD5 sum of the file. |
| -name | The filename of a string(string) |
| -piece length | Number of bytes in each piece(integer) |
| -pieces | String consisting of the concatenation of all 20-byte SHA1 hash values, one per piece.(raw binary encoded) |
| Announce | The announce URL of the tracker |
| Announce-list(optional) | This is an extension to the official specification, which is also backwards compatible. This key is used to implement lists of backup trackers. The full specification can be found at http://home.elp.rr.com/tur/multitracker-spec.txt. |
| Creation date (optional) | The creation time of the torrent, in standard Unix epoch format (integer seconds since 1-Jan-1970 00:00:00 UTC) |
| Comment(optional) | Free form text comments.(string) |
| Created by(optional) | Name and version of the program used to create. |

Table 4.1: Structure of Torrent(MetaInfo) File [8]

the needed information from the same "torrent" file.

This way, both solutions can coexist as depicted in 4.7.

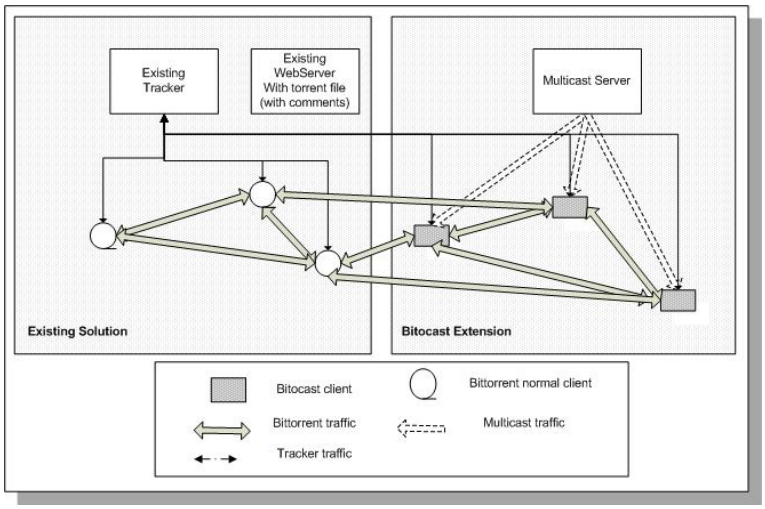


Figure 4.7: Integration of Bitocast with existing BitTorrent solution

Since the tracker used in BitTorrent and Bitocast is the same, the swarm BitTorrent network that is formed is based on the same peer set. Bitocast clients also connect to normal BitTorrent peers in order to exchange pieces with them. The difference here is that Bitocast clients have the benefit of also receiving information from the Multicast server. For the tra-

ditional BitTorrent peer, the Bitocast peer behaves normally in terms of BitTorrent protocol exchanges.

In this chapter we presented Bitocast its architecture and inner workings. In the next chapter we'll present how we built its simulator.

# Chapter 5

# Bitocast Simulator

There is no standard simulator used for P2P research. Most authors use their own implementation of a simulator. NS-2 [35] is a popular network simulator with extensive documentation but has a steep learning curve[34] and it would not be ease to map Bitocast in the simulator. NS can support very precise and detailed simulations but has significant problems regarding scalability. To implement the simulator for Bitocast we build on top of the existing BTSim[5].

## 5.1 BTSim

BTSim was the simulator used by Microsoft Research on their work to study BitTorrent[9] and that was generously made available. Their main goal was to analyze BitTorrent's performance in *flash crowd* scenarios.

It is a discrete-event simulator that models peer activity (joins, leaves and piece exchanges) while also implementing BitTorrent's policies (local rarest first, tit-for-tat, etc). Piece exchanges are modeled and delayed according to each peer's bandwidth, defined asymmetrically in terms of upload and download capabilities, and also taking into account the concurrent transfers at both ends of the connection.

The simulator implementation does not take into account backbone network capabilities, propagation delays, shared bottleneck links in the interior of the network or packet level dy-

namics of TCP connections, the data exchange is at the piece level and not at the block level as it is the case in BitTorrent, but the authors of the simulator believe that these limitations do not have a significant impact on the overall results. Since our work is mainly focused on local or ISP networks, where network latency is smaller and more homogeneous, these simulator limitations are even less relevant. The *End game* mode is also not taken into account and it only allows to simulate and analyze the download of a single "torrent" per simulation run.

The simulator is very configurable. It allows the setting of many parameters such as the size of the file being exchanged, the size of the pieces and number of initial *seeds* amongst others. In terms of simulation data, it can be instrumented by a workload file that defines the joins and leaves of the nodes or by defining a set of parameters like a join leave model (joining rate, the leaving probability), the probability of the *seed* leaving, etc. Please refer to appendix A for a more detailed description of the simulator configurations.

Since all the core functionalities of BitTorrent were already implemented, and the limitations of BTSim where not critical for the goal of our work, we choose to extend this simulator to test Bitocast.

## 5.2 Bitocast Simulator

We built the Bitocast Simulator by extending BTSim to support the functionalities proposed in this dissertation. The main changes were the implementation of the IP Multicast server and adaptation of the BitTorrent client in order to be able to listen to the IP Multicast channels and receive pieces from them at the same time it receives from BitTorrent.

The simulator collects data regarding each IP Multicast channel and Bitocast client statistics. It also keeps logs of all its actions in log files so they can be processed after the simulation is complete in order to produce overall data and charts analysis.

### 5.2.1 Configurations

Bitocast allows for the configuration of specific properties on top of the ones provided by BTSim. Namely:

- The number of multicast channels and their bandwidths.
- The peer's bandwidth that is reserved to BitTorrent vs IP Multicast
- The piece and scheduler model used by the multicast server channels.
- Should the multicast channel repeat the emission or send just once its content.
- If there is a initial BitTorrent seed or the seed is just the Multicast Server
- If BitTorrent piece selection takes into account what will be sent by the Multicast channel in order to reduce wasted deliveries (cooperative mode).
- Continuous Mode: Yes / No
- Continuous Mode Sliding window
- Continuous Mode Number of Subtorrents
- Continuous Mode Sliding Window bandwidth Head

Continuous mode will be explained in chapter 7.

### 5.2.2 Limitations

Bitocast simulator does not try to overcome BTSim's limitations, so, it also has all of the limitations we previously referred to. Besides those limitations, our simulator has the following:

- Multicast Control traffic messages overhead is not taken into account.
- Multicast group management network packets are not taken into account.
- It continues to only allow the simulation of the download of one "torrent" alone.

### 5.2.3 Implementation Details

Bitocast was implemented in C# over Microsoft .Net 3.0 Framework. The development environment was Visual Studio 2008. The class diagram of the Bitocastsimulator specific classes is depicted in figure 5.1.

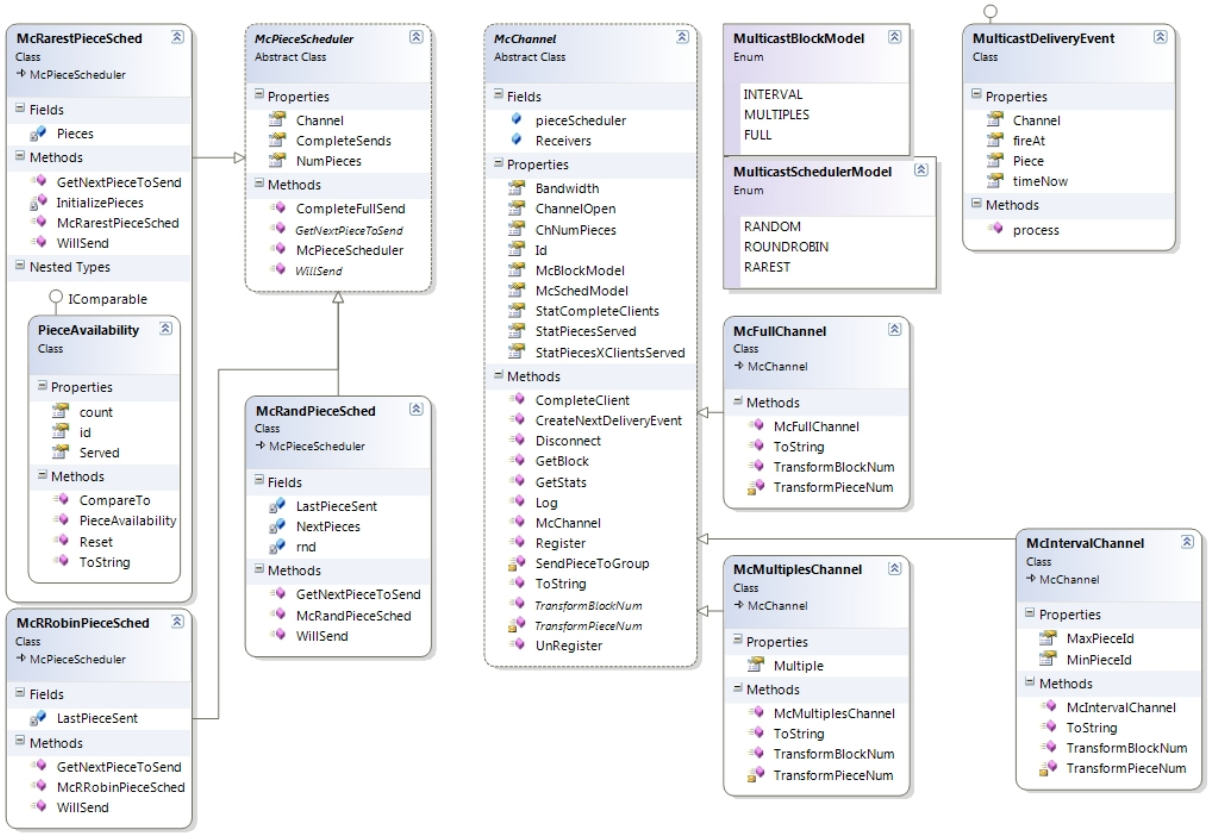


Figure 5.1: Bitocast Class Diagram (BitTorrent classes not included)

The application was built as a discrete event simulator. In discrete-event simulation, the operation of a system is represented as a chronological sequence of events. It is implemented as a single threaded console application that will process the events according to their time on the queue. Some of the events can be fed in a workload file or just be created by the simulator.

On its simple form, the simulator takes the following high level steps:

- Process command line parameters and initialize the environment.
- While not reached simulation end time or end of workload file:
  - Process workload file and put events in queue.
  - Process events in queue.
  - Collect statistics and log data.

One could be tempted to implement the simulator as multi-threaded application where each node would have a separate thread, but this type of architecture would not allow the simulator to scale to a large number of peers on the network.

The main data structures of the simulator are the Event queue and the Peer List. The Event queue is implemented as a Splay Tree [1] and the Peers are stored in a SortedList. The data structures used by each peer are the connections to remote peers (Hashtable) and it's piece availability (Array). The IP Multicast server also has a list of all clients connected to each channel (List<McPieceReceiver>).

Depending on several factors like the file being analyzed, the block size, the piece size, the number of peers amongst others, the memory occupied by these data structures can lead to situations where the simulator runs out of memory. Also, increasing simulation details also effects simulation time considerably.

On the simulations results shown in figure 5.1 , where we kept the same variables and only changed the block size from 256 to 2048, it is noticeable that the simulation times vary by a factor of 14 times while the overall download time does not change more than around 10%.

With these results in mind, we took the decision to use 2048 as the block size for all our simulations in order to reduce simulation run times, allowing to run more different scenarios and variable changes.

[1] A splay tree is a self-balancing binary search tree that benefits from the fact that recently visited elements are quick to access again.

| Simulation Duration | Block size | Download Time (sec.) |
|---|---|---|
| 3859 | 256 | 1234 |
| 1188 | 512 | 1306 |
| 569 | 1024 | 1280 |
| 262 | 2048 | 1181 |

Table 5.1: Simulation and download times variation per block size

### 5.2.4 Simulation Environment

The simulations were conducted, and the results analyzed, using a set of different virtual machines and technologies. The virtual machines were hosted on VMWare cluster of 2 HP Proliant DL380 servers, with 8 Xeon processors at 2.33 Ghz and 14 Gb of memory.

Two different types of virtual machines were used. As shown in figure 5.2, the simulations were run on windows machines and the results were analyzed on a Ubuntu machine. Two Windows Machines were used in order to be able to ran several simulations at the same time. The simulation machines were running Microsoft Windows Server 2008 and had 4 virtual CPU's with 4 GB of memory each.

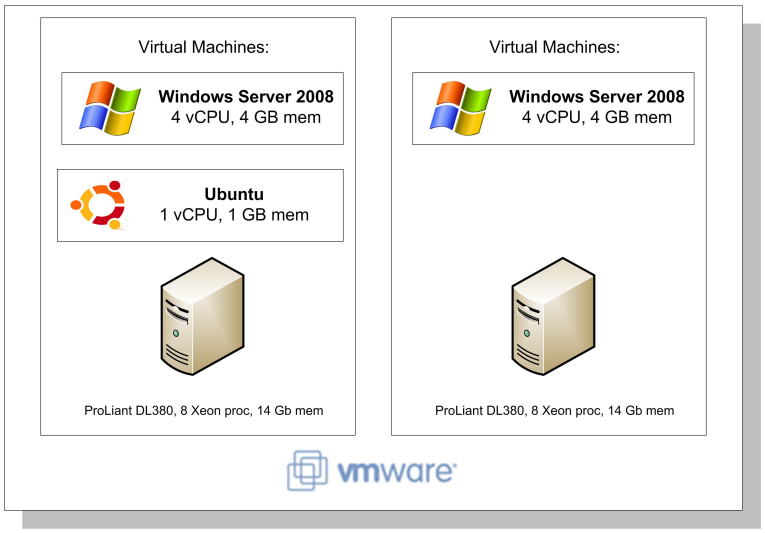


Figure 5.2: Simulation Environment

The Windows machine that ran the simulations did so orchestrated by a perl script. The script cycled through all the different parameters and passed them via the command line to the simulator. The simulator produced a set of files with all the information regarding the

simulation. In order to be able to run as many simulations as possible in parallel, besides using two different virtual machines, up to four simulations were executing at the same time on each machine. This was in order to best use the four available processors.

Besides the Windows machines, there was another virtual machine that just took care of the analysis of the simulation data. This machine was running Ubuntu, a Linux based operating system. Since this process was quicker then the simulation itself, only one machine was needed. It ran a set of perl scripts and graphical libraries. The scripts would run through all the simulation generated files, calculate a set of information such as averages, maximum and minimums and produce a series of charts in order to enable the visualization of the simulation data. For each simulation that was ran a PDF file was generated with the overall input and outputs, and all generated charts. A *monitor* script was constantly looking at the output folder of the windows machines and checking for new analysis jobs. Once it found a new job it launched a new analysis script that would do its work according to the simulations parameters and would produce and publish the results to the window machine. The virtual machine had one virtual CPU and 1 GB of memory. The process is illustrated in figure 5.3.

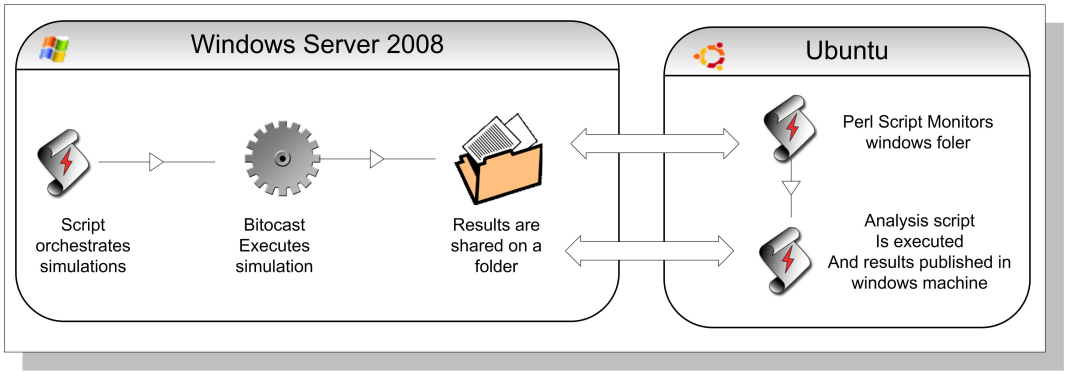


Figure 5.3: Simulation execution and analysis

Each simulation produced an average of 900 MB of simulation data and 50 MB of generated charts and aggregated data.

In this chapter we presented how we developed and enable a simulation environment for Bitocast. In the next chapter we show the results of running Bitocast in this simulated environment.

# Chapter 6

# Comparing Bitocast to BitTorrent

One of the goals of our work was to compare the performance of Bitocast when related to BitTorrent, in order to understand if it was in fact beneficial and what benefits where those. Since reality and the implemented simulator have a lot of variables that can be considered, it is complicated to decide on what factors to analyze when running the simulations. We need to be able to focus on results that allow us to reach conclusions in the most trustful way, and with the most probability to be applied in the real world.

Several thousands of simulations were run, collecting data over varying configurations in order to enable a thorough analysis of the proposed solution. While comparing the metrics of standard BitTorrent versus Bitocast we also wanted to infer what configurations would produce the best results. The same events were fed into BitTorrent and Bitocast simulations and the results compared and analyzed.

In order to understand the impact of each configuration on the overall results several simulations were run changing only one configuration and keeping all other equal. The results were put side by side in order to draw the conclusions.

## 6.1 Simulation details

The simulation was driven using data gathered during the first days of distribution of Red Hat 9 [26]. We utilized the real pattern of appearance of nodes on the network as they

happened for fact in a real case scenario. The pattern of node arrival was according to the chart in figure 6.1.

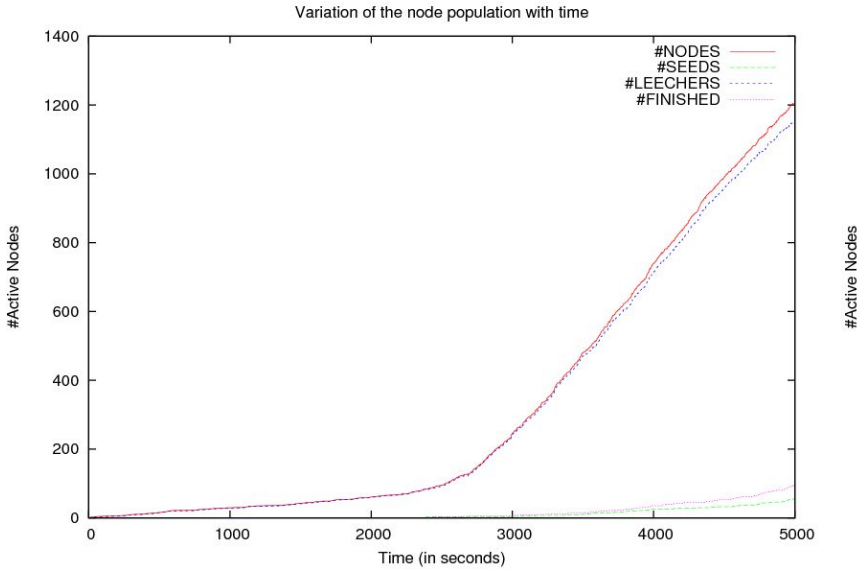


Figure 6.1: Node arrival pattern of the distribution of Red Hat 9

These first simulations were based on a distribution of a file of 100 MB. Several simulations with different options were run and compared to BitTorrent under the same conditions. The download bandwidth allocated for BitTorrent and Bitocast traffic was 50% the available bandwidth of each node. On these simulations the IP Multicast channels repeated the emission of their content until the end. On each simulation there was only one Multicast channel emitting at 50% of the download speed of the peers. The simulations were executed several times for each condition and the results gathered and averaged. The results are shown on the following sections.

## 6.2 Piece and scheduler model simulation analysis

We first studied which of the IP Multicast Scheduler and Piece model would produce the best average results. Figure 6.2 compares the average number of nodes that downloaded a complete copy of the file. The chart shows the different Schedules and different nodes bandwidths. The red bars represent Bitocast and the blue line BitTorrent. The numbers in the bottom represent the bandwidth in the form "download:upload:percentage of nodes". In

this case the percentage of nodes is always 100% because each simulation was run with all nodes having the same bandwidth in order to study each case isolated.

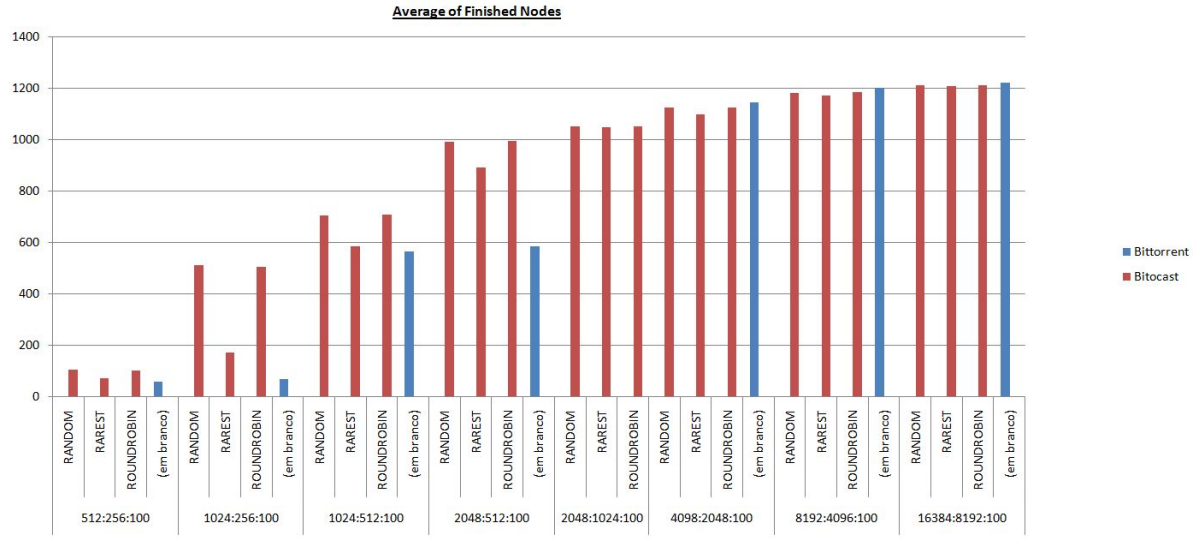


Figure 6.2: Average number of nodes that finished downloading the file, according to the Scheduler model, and compared to BitTorrent

From the chart we notice that:

- Bitocast is more efficient than BitTorrent when in low bandwidth conditions, tending to be equivalent on higher bandwidths and upload /download rations of 2/1.
- Bitocast is more efficient when the difference between download and upload capabilities are higher. The difference is much higher for 2048:512 and 1024:256 (that have a ratio of 4 to 1 of download / upload) than in 1024:512 or 2048:1024 (2 to 1 ratio).
- Rarest model is always less efficient than the others.
- Round Robin and Random mode have very similar results.

At first one could think that the Rarest model would produce better results than the other models, but that is not the case. The reason is that the rarest model results in a high waste of pieces that are sent because it is using the same piece selection model as BitTorrent. When in Random mode the probability of a node receiving a piece from the IP Multicast server that it already has is higher than other modes and produces inefficiencies.

We discard the Rarest Scheduler because its implementation would require some overhead in order to reach the conclusion of what were the rarest pieces on a given time. It would also

| Product / Provider | Download | Upload | Ratio |
|---|---|---|---|
| ZON Netcabo 30Mb | 30 Mb | 1 Mb | 30 |
| Ya.com | 20 Mb | 1 Mb | 20 |
| ZON Netcabo 18Mb | 8 Mb | 1 Mb | 18 |
| Orange | 10 Mb | 320 Kb | 32 |
| ZON Netcabo 8Mb | 8 Mb | 512 Kb | 16 |
| Arrakis | 6 Mb | 512 Kb | 12 |
| ZON Netcabo 4Mb | 4 Mb | 256 Kb | 16 |
| ZON Netcabo 2Mb | 2 Mb | 128 Kb | 16 |
| Telepac | 1 Mb | 128 Kb | 8 |

Table 6.1: Download / upload bandwidth ratios of main Internet Service Providers

require more structural changes to the existing solutions that we base our work upon. Since that overhead was not being taken into consideration on the simulations and the overall result is very similar to the Round Robin model, the Rarest model was discarded.

The difference in performance of Bitocast when the ratio of download / upload is 4 to 1 versus 2 to 1 is due to the fact that multicast delivery does not take into account the upload capabilities of the node while in BitTorrent the upload capability is directly related to the download bandwidth the node can achieve (if we do not take into account the number of available BitTorrent seeds in the system). This is one of the reasons why the higher the ration the higher performance Bitocast will have when compared to BitTorrent. The tendency is for this ratio to increase. As we can see in table 6.1, the ratio of download / upload capabilities offered by the main ISPs is increasing with the available bandwidth. Since having a higher ration benefits the performance of Bitocast, the scenario offered by ISPs is aligned with the case where Bitocast makes a difference. This is not always the case since some enterprizes use dedicated Internet access that typically have the upload capacity equal to download.

Regarding the Piece model simulations in figure 6.3 we notice that their differences are less significant than the Schedule models. They perform more or less in the same way with minor differences.

Since the different Piece models produce similar results, the Full Piece Model was selected on further simulations. This is because it is the easiest to implement in the real world and does not need to take into consideration other variables. Deciding for the Full Piece Model

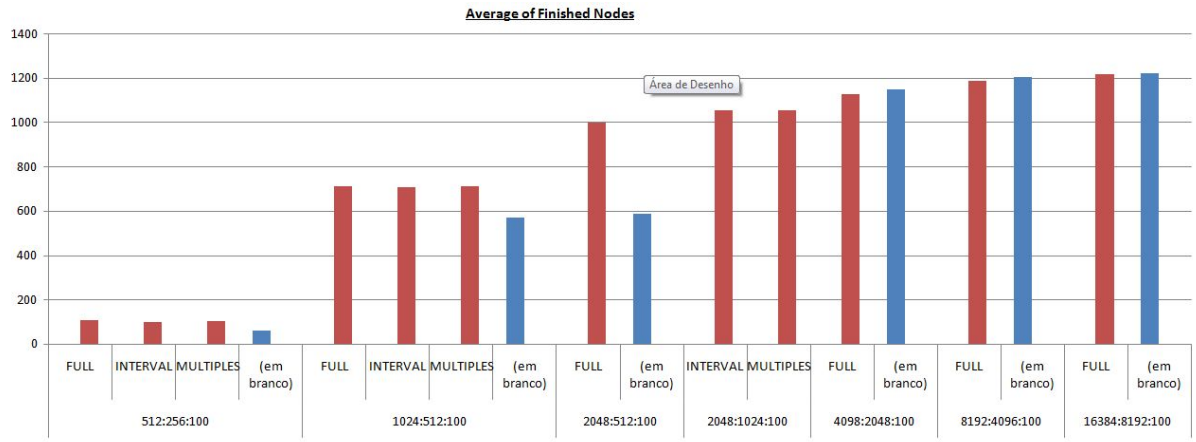


Figure 6.3: Average number of nodes that finished downloading the file, according to the Piece model, and compared to BitTorrent

does have one disadvantage in that the allotted bandwidth for IP Multicast communications might not be fully used because IP Multicast channels cannot be combined since the same content will be sent over the different channels. The better the fit between the % of bandwidth reserved for IP Multicast and the available IP Multicast channels bandwidth the better IP Multicast will perform. Obviously the bandwidth that will not be used in IP Multicast communications is used in BitTorrent transfers.

From these simulations we conclude that the best Scheduler Model to use is *Round Robin* and the best Piece Model is *Full*.These will be the models used on the subsequent simulations.

| Mode | BT Waste | MC Waste | Finished | Download Time | Pieces Sent |
|---|---|---|---|---|---|
| BitTorrent | | | 822 | 901862 | 343 |
| Bitocast | 10085 | 120193 | 937 | 558923 | 197 |

Table 6.2: Overall simulation results

## 6.3 Comparison to BitTorrent

In order to better understand the difference of BitTorrent and Bitocast we need to delve deeper into the analysis of a simulation of BitTorrent and another of Bitocast.
The details of the simulations were the following:

- File Size: 100 MB
- Block Model: Full
- Scheduler Model: Round Robin
- Share of bandwidth between BitTorrent and Bitocast: 50%.
- Multicast channels were in repeat mode (always sending the same content over and over)
- There was an initial seed that had at 3Mbit upload capacity.
- 5 Multicast channels with the following bandwidths: 128,512,1024,2048 and 4096. These numbers match with the share ratio of multicast communication of each node (50%).
- After downloading the file and becoming seeds the nodes had a probability of leaving the network of 50%.
- Bandwidth distribution of the nodes were:

| Download (Kbps) | Upload (Kbps) | Probability (%) |
|---|---|---|
| 512 | 256 | 5 |
| 1024 | 254 | 25 |
| 2048 | 512 | 25 |
| 4096 | 1024 | 40 |
| 8192 | 2048 | 5 |

The overall results are show in table 6.2. All figures represent the average values of the nodes that participated in the simulation. *BT Waste* identifies the number of pieces that were received via BitTorrent that were previously received via Multicast. *MC Waste* identifies the pieces that multicast delivered that meanwhile arrived via BitTorrent. *Finished* is the number of nodes that received a complete copy of the file during the simulation period. *Download time* is the time (in milliseconds) it took to receive a full copy of the file. *Pieces Sent* is the number of pieces that the nodes uploaded via BitTorrent.

As a first conclusion we notice that, in this simulation, Bitocast increased the number of nodes that finished the download from 822 to 937 ( 14% increase), it decreased the download

time from 901 to 558 seconds (38% decrease), and also decreased the number of pieces that each node had to upload (from 343 to 197 representing a 42% decrease). We also note that in the Bitocast alternative a lot of waste is produced. A total of 130278 pieces were sent, consumed bandwidth and were afterwards discarded.

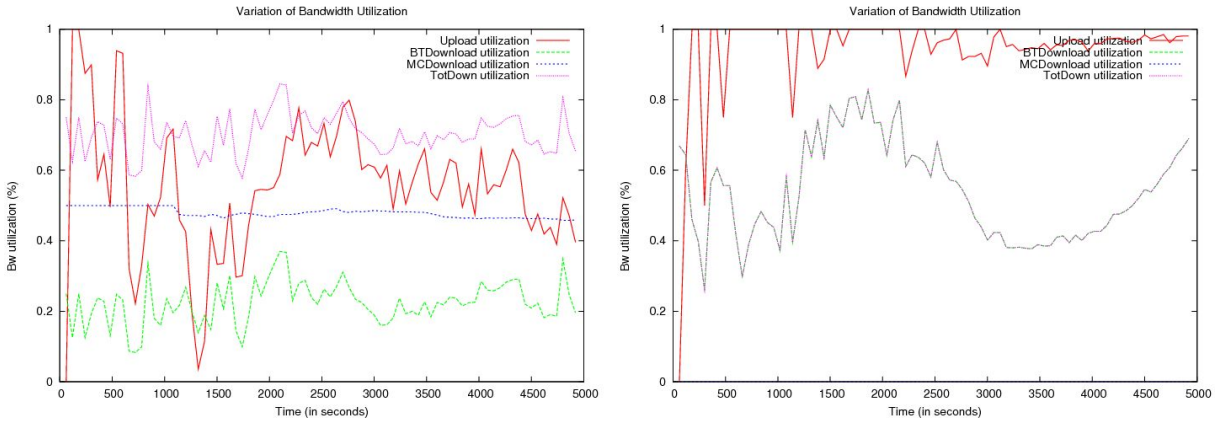


Figure 6.4: Capacity Utilization (Bitocast vs BitTorrent)

By looking into more detail on the simulations and putting their results side by side, Bitocast on the left and BitTorrent on the right, we can see how the bandwidth was used on each case. On the right chart of figure 6.4 we notice that the upload capacity of BitTorrent was most of the time close to it's maximum and the download capacity around 50%. On the left we can see that the upload usage was highly decreased and the overall download capacity increased to an average of 70%. The network usage of BitTorrent decreased significantly meaning a better use of the overall network.

Figure 6.5 shows the existing nodes on the network during the simulation. We notice the *flash crowd* scenario representing the interest of people on the Red Hat distribution (the pattern of node arrivals that was used on the simulation). We also note that the number of seeds is higher in Bitocast when compared with the same time slot in BitTorrent. The number of leechers is also smaller. The number of the nodes on the network is also smaller in Bitocast since downloads finish earlier and some of the seeds start to leave.

Analyzing the chart that shows the average download time aggregated per node bandwidth in figure 6.6 (format of series is download and upload speed: d:1024:u254), we notice that Bitocast clearly puts each node in clusters according to its bandwidth, reducing the differ-

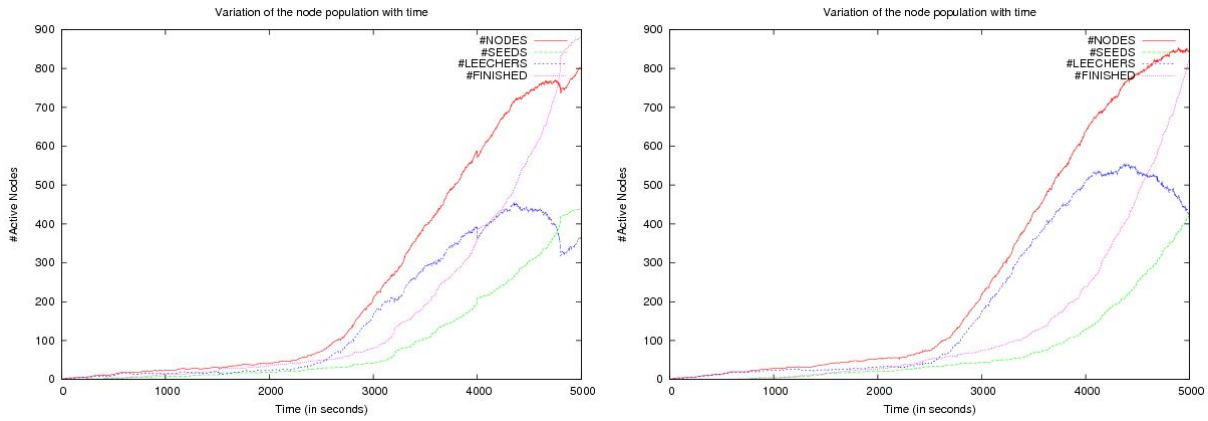


Figure 6.5: Nodes across time (Bitocast vs BitTorrent)

ence in performance for the same capacities. The maximum and minimum download time is much more defined than in BitTorrent and is clearly lower.

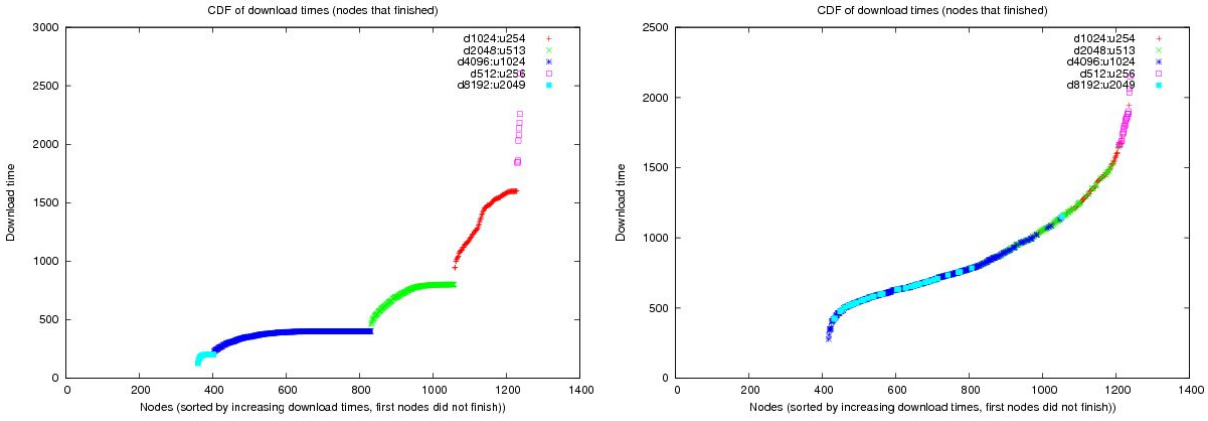


Figure 6.6: Download times of different nodes bandwidth (Bitocast vs BitTorrent)

Figures 6.8 and 6.7 show the average number of incoming and outgoing transfers over BitTorrent . It is clear the impact of Bitocast in reducing the overall number of outgoing and incoming transfers. In Bitocast we also notice that the lower the capacity of the node the lower number of outgoing transfers it has. It should be noticed that number of transfers is not dependent on the bandwidth since we can have the same number of transfers on 512 Kbps or 1024 Kbps, just the Kbps per transfer would be lower on the first case.

Figure 6.9 shows the piece availability bitmap of both simulations (The time frame is the same on both graphs but the scale of BitTorrent is incorrectly multiplied by 1000). The Round Robin model is clearly depicted on the left chart (Bitocast). It is also clear that the first pieces of the file become much more popular earlier in time in Bitocast than in BitTorrent.

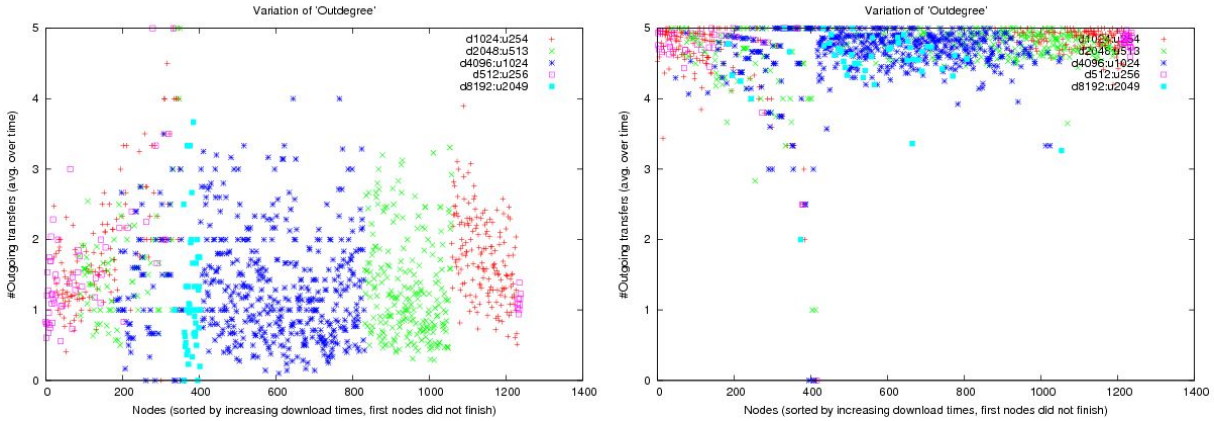


Figure 6.7: Mean outgoing degree (num ftransfers) of a node over time (Bitocast vs BitTorrent)

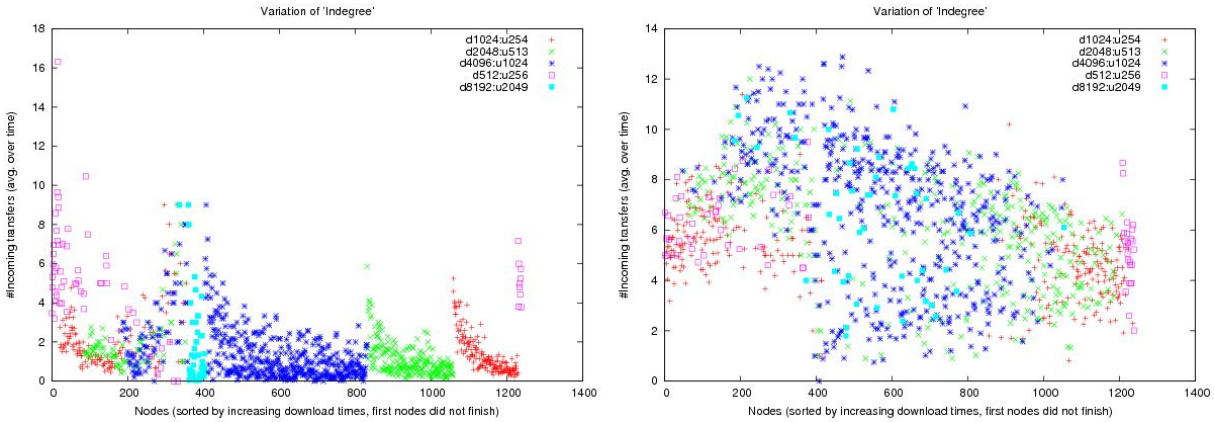


Figure 6.8: Mean incoming degree (num transfers) of a node over time (Bitocast vs BitTorrent)

In BitTorrent pieces tend to have equal popularity as time goes by. The Number of pieces present on the network is higher in BitTorrent in terms of maximum. The reason is that nodes take longer to download the file and stay longer on the network, thus increasing the number of nodes with the same pieces.

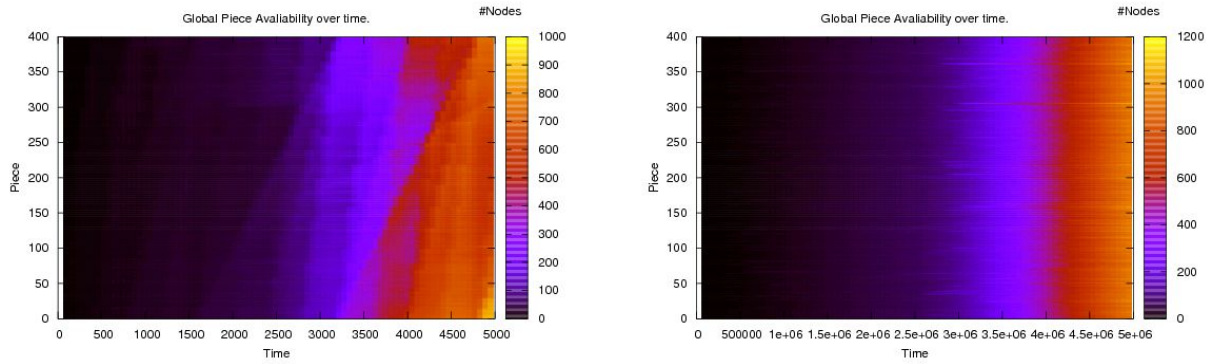


Figure 6.9: Piece availability over time (Bitocast vs BitTorrent)

## 6.4 Cooperative and Repeat Mode

In the previous simulation we saw that Bitocast suffers from some waste in its transmissions due to the fact that we have two distinct and concurrent means of receiving pieces. This specific simulation showed that a total of 130278 received pieces were wasted (10085 from BitTorrent and 120193 from IP Multicast). Neither the Multicast server nor BitTorrent were aware of what the other was transmitting and collisions occurred.

In order to try to reduce the number of wasted pieces in Bitocast a *Cooperative* mode was created and implemented in the simulator. *Cooperative* mode makes the Bitocast node be aware of what the Multicast channels it is connected to will emit in the near future. This way the node will not ask for those pieces via BitTorrent, thus reducing waste and increasing efficiency.

The cooperative mode was implemented by exchanging the BitTorrent piece selection algorithm. The changes make BitTorrent aware of the pieces that IP Multicast will send on the next x cycles. The number of cycles to consider is reached by:

- First calculating the estimated time it will take to download the complete file based on the average download rate until now. If we are in the beginning of the download it is just assumed that we will be using 50% of the download capacity. ETA = number of missing pieces / piece per simulator second until now.
- Then we convert the ETA in IP Multicast cycles according to the IP Multicast channels bandwidth. In this context a IP Multicast cycle is the event of sending one piece to the IP Multicast network channel.
- The number of cycles cannot be more than 25% of the total number of missing pieces.

| Mode | Repeat | Cooperative | BTWaste | MCWaste | Finished | Download Time | Pieces Sent |
|---|---|---|---|---|---|---|---|
| Bittorrent | na | na | na | na | 822 | 901.862 | **343** |
| Bitocast | False | False | 3.287 | 24.201 | 912 | 788.514 | 332 |
| Bitocast | True | True | 19.494 | 43.622 | 879 | 667.047 | **83** |
| Bitocast | True | False | 10.085 | 120.193 | 937 | 558.923 | 197 |

Figure 6.10: Comparison of simulation runs

By running a more complete set of simulations, with the same settings as before except for Cooperative mode changes, and comparing them altogether, enables to study the impact of the Cooperative Mode.

Table 6.10 shows that Cooperative mode is successful in reducing waste. When directly compared with the previous simulation it reduced the total waste from 130278 pieces to 63116. But BitTorrent number of wasted pieces increased (from 10085 to 19494) while IP Multicast waste decreased (from 120193 to 43622). Despite the reduction on the number of wasted pieces the download time increased from 558 to 667 seconds and the number of nodes that completed a download was also reduced from 937 to 879 (still superior to standard BitTorrent that was 822). Even though cooperative mode reduced the overall waste it does not deliver better end results in download time and number of nodes that obtained a complete copy of the file. The waste seems to be the price to pay for this better performance in non cooperative environment. Despite not delivering the best end-user experience Cooperative Mode is a more interesting alternatives for ISPs that are looking at ways to reduce the traffic associated with content distribution.

In table 6.10 we also compared the simulations with Repeat mode set to False. In this mode the IP Multicast channels disconnect after sending their full content once, despite the number of clients that were connected to it from the beginning. Even in this case the results are better than standard BitTorrent. When compared with BitTorrent we notice that the number of finishing nodes increased from 822 to 912 and the download time reduced from 901 to 788 seconds, while keeping more or less the same amount of pieces sent per node and generating a modest total waste of 27488 pieces. It is clear that when sending just once (Repeat mode False) Bitocast is not as efficient as when IP Multicast repeats its emission, but it still is better than BitTorrent and provides benefits for the end-user and the ISP. Releasing

the IP Multicast channel to tune to another content might also be beneficial in the case where the content provider is delivering multiple "torrents", as we will see in the next chapter, and not just one. In either cases (Cooperative and Repeat modes) Bitocast is beneficial for the ISP because it reduces the network usage.

In this chapter we directly compared BitTorrent's performance to that of Bitocast. We did this comparison regarding it's overall performance, the Scheduler and Piece Model. We also introduced the Cooperative and Repeat mode and studied their impact. In the next chapter we will present a rich media distribution solution that uses Bitocast.

# Chapter 7

# Rich Media Distribution solution

## 7.1 Commercial scenario

Building from Bitocast we envision a commercial scenario to serve rich media.

### 7.1.1 Existing situation

With the increase of network bandwidth offer and the will of end-users to consume media (as we can see by the success of sites like Youtube) companies started to sell Video On Demand (VoD) products. Some companies sell products that are actually not real video on demand but multiple channels with predefined schedules. The ones that do sell real VoD do have a scalability problem due to the limitations of the distribution network model and the impact on the central servers. Some companies rely on *content distribution networks* and other technologies in order to be able to scale to many users and achieve high availability but pay high price for that.

Internet TV also exists and utilizes the existing Internet network to deliver its content. Since the Internet TV provider has no control over the network it can only provide the service on a "best effort" basis [45] which allows for a limited end-user experience: low media quality, high probability of interruptions, etc.

In order to provide scalable and low cost alternatives several solutions of streaming media

over P2P networks have been proposed. CoolStreaming [47] and Gridcast [14] are some examples. As previously stated, these systems build upon the benefits of P2P networks but do not produce a high quality and rich end-user experience by having high startup times, interruptions in the transmission, loss of quality, and difficulty in coping with constant changes in the content being viewed by the user, a attitude also known as zapping.

Streaming video in real time has high requirements in terms of bandwidth, small packet delays and reliability. Due to the lack of coordination, the limited peer bandwidths and low system stability these requirement are not at the heart of unstructured P2P systems [28].

### 7.1.2 Proposed solution

We envision a commercial scenario that could be sold by a company that owns content and the network infra-structure (such as PT Multimedia in Portugal), where a potential big set of digital assets - that is constantly changing - would be provided to their costumers in very high quality, on a short time frame from the date of publication, and served directly from the local device. New published content would be continuously disseminated to the clients and the decision of what and when to watch the content would be on the user.

Users would be able to select media from the existing set, that was recently distributed, or from old media that was distributed in the past and that was marked as favorite. The user would also benefit from the fact that he is watching a local movie and not some media being streamed over the network. There would be no startup time (at least in terms of network), no network jitter, and instantaneous seek time. He would also be able to quickly zap between existing content, something that was not easily feasible when streaming content from the network. Since the content set would be large and new, the user would most probably have media of its interest to consume. One could also enhance the scenario to allow for the selection of groups of content to be delivered to ones device.

Content broadcast companies, that also own the dissemination network, often do not use their network to its full potential. Bitocast comes into play in order to enable this vision. These types of media companies, that also own the network, can easily utilize IP Multicast because they control the network from end to end.

## 7.2 Implementing the vision

In order to build the previously mentioned scenario we take for granted that each node is a "black box" device and that it is most of the time turned on and connected to the network. The architecture is controlled from end to end and the device is dedicated to the implementation of the solution. It is assumed that nodes do not appear in a *flash crowd* scenario since they are already, and always connected, exchanging information as it appears from the source. In terms of nodes present on the network, we can say that a *flash crowd* scenario would only exist on the initial startup of the solution. Regarding a *flash crowd* scenario towards new content, that situation would always exist because the device is always fetching new content.

Since disk space of the local device is limited we would need to manage the content of the local storage. New content would be fed into the network by the content provider and the devices would save it until it runs out of space and then dispose of old content. The user would be allowed to mark some content as favorite so it is not deleted, but it is compromising the space available for new content. A certain amount of the disk space could be reserved for the favorite content and the rest would hold the content being served through the network. Obviously, with this solution, the network will be used to disseminate content that might never be consumed by the user.

The content provider is, for sure, continually producing new content that wants to deliver to its costumers. So, IP Multicast will be continually serving new content once for each new individual piece of media. Since it is not feasible to serve all content at all times, a notion of sliding window is created. BitTorrent will serve pieces of the files that are in that sliding window. This way BitTorrent will serve as an alternative to fill in the gaps of the pieces that the clients were not able to receive via IP Multicast, either because they were not connected, because some failure occurred or simply because their bandwidth is not enough to connect to a IP Multicast channel.

Regarding BitTorrent communications, there is a unavoidable compromise between the size of the sliding window and the speed of distribution of new content. If the size of the sliding window is too big, new content will take longer to be spread to nodes. If the size is small, the

system will focus on a smaller set of content and will cooperate to disseminate that content on a shorter time frame.

Users connecting to the network in the future, would not be able to receive content via Bitocast that is not part of the actual set of "active content" or sliding window. As an enhancement of the scenario, a certain bandwidth of BitTorrent communications could be set apart in order to allow "old" content (content that is not part of the sliding window) to be exchanged between peers. This way allowing for new devices connected to the network to also be able to receive those old files, but in a much lesser efficient way.

It would also be feasible to include in the solution some peers that could not connect to the multicast channels. In that particular case those peers would only be able to communicate via BitTorrent. The size of the sliding window would need to be large enough to accommodate these peers. Despite the fact that the solution would not be as efficient as having multicast delivery it would enable the deployment of the solution to a broader client base.

## 7.3 Simulation details and analysis

In order to study the viability of the proposed scenario the Bitocast simulator was extended.

### 7.3.1 Changes to the Simulator

The Bitocast simulator only takes into account the transfer of one "torrent" at a time. It is not prepared to consider the exchange of multiple "torrents" at the same time. When a node is exchanging multiple "torrents" at the same time there is a competition for the nodes bandwidth and much more details need to be taken into account in order to simulate such a scenario.

Due to time limitations on the elaboration of this dissertation, and to be able to simulate the scenario, we adapted the existing simulator to study this scenario. The adaptation was the creation of a *Continuous* Mode. This mode was implemented not by allowing the coexistence of multiple "torrents" on the simulator but by splitting the one "torrent" the simulator is transferring into multiple "sub-torrents". The simulator still delivers just one "torrent"

but statistics about each sub "torrent" are gathered at regular times in order to know how many pieces where exchanged for each sub "torrent". This way it is feasible to conclude how many nodes have a complete copy of each "sub-torrent". With this workaround we are simulating competition among "sub-torrents" as if they were real "torrents" being downloaded at the same time. This scenario is shown in figure 7.1.

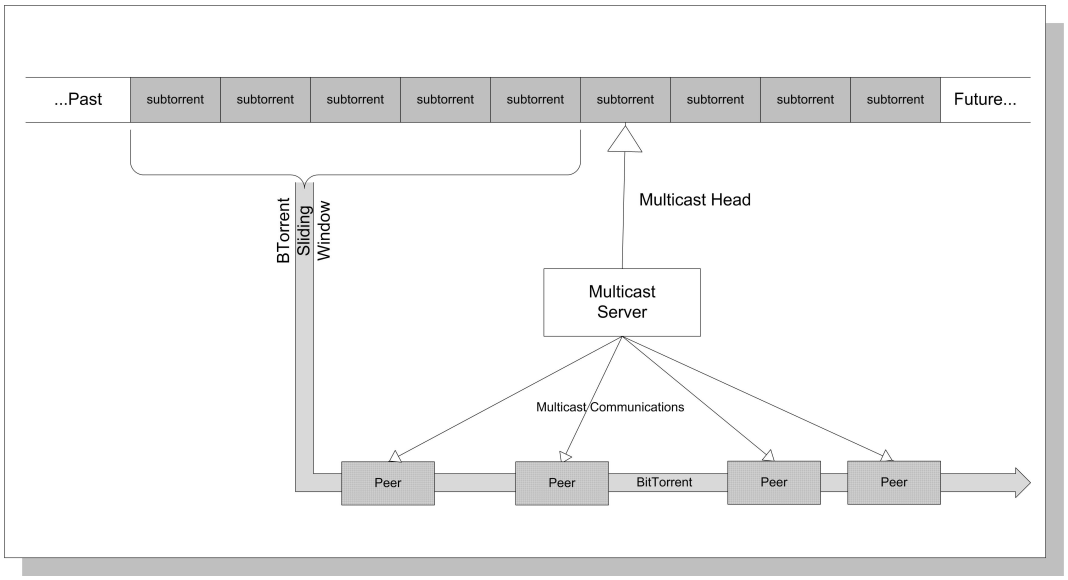


Figure 7.1: Rich media scenario with a sliding window and multicast emission head

The piece selection algorithm was also changed so it takes into account the notion of a *sliding window* of available "sub-torrents". The sliding window limits the "sub-torrents" that will be exchanged via BitTorrent. At the same time that "sub-torrents" pieces are being exchanged over BitTorrent the IP Multicast channels are serving the pieces of one "sub-torrent" ahead of the sliding window, so, we take out the waste of Bitocast communications.

The number of "sub-torrents" and the size of the sliding window is fed into the simulator. The simulator divides the total number of pieces of the "torrent" in the desired number of "sub-torrents" and only serves pieces accordingly. A pointer head that flows with time was also implemented in order to specify where the sliding window starts and ends. This pointer is defined in Kbps so it can follow what the IP Multicast server just sent.

### 7.3.2 Running the simulations

In the proposed scenario, nodes are usually not expected to leave the network. The only reason to leave would be a power failure or network problems, or simply because the user turned off the device. The typical pattern seen in normal BitTorrent networks, of the peers disconnecting after becoming seeds, would also not be applied in this case because the file is not supposed to end (being a continuous stream of new information). In following simulations, the size of the file is large and we do not give enough time for the download to end.

If we admit that IP Multicast is available to all peers of this closed solution, we might as well enable the peer to use a lot of its bandwidth for IP Multicast communication and let BitTorrent work only as a backup. On previous scenarios, BitTorrent would need to work on a situation were IP Multicast is partially available, so the network share between IP Multicast and BitTorrent needed to be balanced in order to keep both running. In this case, we can use BitTorrent only as the means to compensate for IP Multicast's lack of reliability, but also serve some "old" media that was outside of the peer's sliding window. For this reason we establish a bandwidth share of 80% for IP Multicast and 20% for BitTorrent for the following simulations. This value is empirical and different scenarios should be studied on future work.

The best scenario to test this solution would be to simulate a long period of execution, such as one week, while downloading several Gigabytes of information. Since the simulator cannot handle such a load, leading to out of memory situations, we had to work with lower numbers and extrapolate the results from there.

In terms of available node's bandwidth, despite working in a closed network solution, it does not mean that all clients have a good connection. It can be the case that the provider wants to have a national reach but does not have the most recent technology all over the country, so, we run our simulations with a mix of bandwidth distributions shown on the next page.

We run the simulations with the following parameters:

- Node bandwidth distribution:
  - Down: 1024kbps, Up: 254Kbps Probability = 25%
  - Down: 512Kbps, Up: 256Kbps Probability = 5%
  - Down: 2048Kbps, Up: 513Kbps Probability = 25%
  - Down: 4096Kbps, Up: 1024Kbps Probability = 40%
  - Down: 8192Kbps, Up: 2049Kbps Probability = 5%
- Multicast bandwidth share: 80% (representing 80% of 1024 Kbps, the second smallest channel).
- Multicast channel: 768 Kbps (equal to the client's Multicast bandwidth).
- Speed of sliding window (is equal to Multicast channel): 768 Kbps.
- File size: 1250 MB.
- Nodes on the network: 2001.
- Sliding Window Size: 20 (The number of "sub-torrents" that are available for BitTorrent exchanges).
- Nodes arrive all in the beginning of the simulation.
- Number of "sub-torrents" in simulation: 100, size per "sub-torrent": 12.5 MB.
- Considered "sub-torrents" in analysis: 72 (took first 2 and only considered until the "sub-torrent" that was sent by the Multicast Server in order to study the system at normal flow).

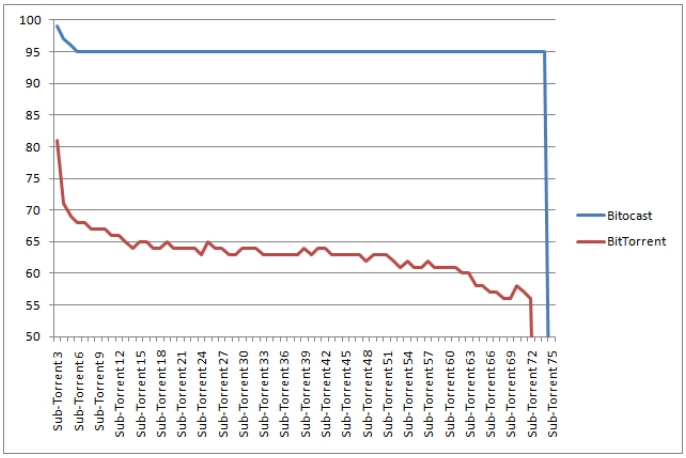


Figure 7.2: Percentage of complete downloads, per "sub-torrent", in Continuous mode (Sliding Window = 20)

We started by considering a sliding window of 20. The results of this simulation, regarding the percentage of peers that downloaded a full copy of each "sub-torrent", can be checked in figure 7.2. The Average percentage of Bitocast versus BitTorrent were:

- Bitocast: 95 %
- BitTorrent: 63 %

As we can see, Bitocast achieves a much higher success than BitTorrent in the same circumstances. Bitocast cannot achieve 100% because of 5% of clients cannot connect to the multicast channel due to not having enough download bandwidth (the ones with 512 Kbps). So, they must rely exclusively on BitTorrent to exchange data. Further analysis should be made in order to better understand why, in Bitocast, the 5% clients that are not able to connect to IP Multicast are not able to significantly download "sub-torrents". As we can see, they are able to download in the beginning of the simulation, when the focus of BitTorrent exchanges is on 5 or 6 "sub-torrents", but from there on they are not able to complete. If they would be using their bandwidth to fully transfer one "sub-torrent" via BitTorrent they would take around 00:3:20 (3 minutes and 20 seconds at 512 Kbps) to download a "sub-torrent" (in ideal conditions), since the sliding window head is "running" at 768 Kbps, and its size is 20, each "sub-torrent" would be available for 00:44:27, which should give enough time for the download to happen. But since the sliding window is too big, the download is being spread along many "sub-torrents", not resulting in the successful download of complete "sub-torrents".

If we run the same simulations with a sliding window that considers 50 "sub-torrents" (figure 7.3) we notice that we achieve a better performance of BitTorrent alone, but Bitocast remains more or less the same (in average percentage of clients that received a complete copy of each "sub-torrent"):

- Bitocast: 95 %
- BitTorrent: 66 % (3% increase)

In terms of overall average, BitTorrent performs better with a sliding window of 50, when compared with the size of 20, because we are giving a longer time slot for the "sub-torrents" to exchange pieces, but clearly there is an inflection point where a too big sliding window will not produce good results.

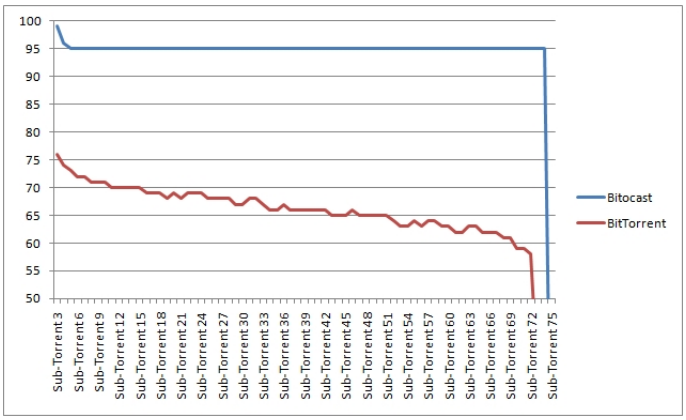


Figure 7.3: Percentage of complete downloads, per "sub-torrent", in Continuous mode (Sliding Window = 50)

As it can be noticed in both figure 7.2 and 7.3, BitTorrent performance is much better in the beginning and degrades in the end. This is because of the focus of the sliding window. In the beginning the window does not yet have the expected 20 "torrents" so BitTorrent focuses on less "sub-torrents", achieving higher performance. When the sliding window has 20 "sub-torrents" the BitTorrent performance stabilizes, and in the end performance is reduced because the last "sub-torrents" are not given enough time to download and the sliding window still has 20 past "sub-torrents" active.

Clearly there is an area of future work in order to analyze the best working scenarios for this solution. Simulations need to be run in order to establish the best parameters for the share ratio between IP Multicast and BitTorrent and the size of the sliding window. Also, we should be able to simulate a long time frame of content distribution with several Gigabytes of information in order to fully understand the advantages of this model.

# Chapter 8

# Conclusions

Much like Parkinson's law, and despite the evolution of ISP's internet access bandwidth offer, the quality, size and number of media content on the web will expand to fill the available bandwidth for its distribution. The existing solutions to distribute these contents have some difficulty to maintain an acceptable service level when there are many users wishing the same content. At least without significantly raising the distribution costs.

Since its appearance, HTTP has been used as the main means to distribute content over the web. But its main design goal was not the distribution of very large files to a large audience. As a consequence, a new set of so called *peer-to-peer* protocols emerged and took over a very significant portion of internet traffic. These protocols removed the dependency from a central server in order to distribute content, thus enabling higher scalability without a direct correlation to distribution cost. This is possible by having the peers, that want a certain content, exchange data amongst themselves ultimately without relaying on a central server.

One of the best known and most widely adopted *peer-to-peer* protocols is BitTorrent. It is responsible for a very big slice of the overall internet bandwidth consumption. BitTorrent only relies on a central server in order for the peers to find each other, and newer versions take this barrier away by using distributed hash tables, all else is enabled at the peer level. Data exchange is made directly between peers. This exchange happens between the peers that have a complete copy of the file, known as *seeds*, and the ones that are seeking to have a complete copy, known as *leechers*. Despite offering a low cost alternative for content dis-

tribution and having a wide adoption, its use of the network is far from optimal. This is because BitTorrent builds upon an "unicast overlay network", where the same information flows from peer to peer, potentially passing many times over the same physical network links, resulting in inefficient usage of resources.

Another alternative to distribute content is achieved by using the IP Multicast protocol. With IP Multicast the overhead for data transmission to multiple peers is much smaller. The physical network is used in a more efficient way by having each packet travel at most once on each different link that connects the sender to the recipients. Unfortunately, IP Multicast has not seen a huge proliferation over the internet due do its unreliability, lack of commercial model and scalability problems among other reasons. But despite the fact that is is not available on a broad scale over the web, it does not mean one cannot use it, or should not use it on a smaller environment like local or ISPs networks, or together with other protocols in order to overcome its limitations.

In this endeavor we proposed and evaluated a new alternative to distribute content over the Internet. We build upon the fact that BitTorrent is a very successful and proven alternative to distribute content with a very low implementation cost. Also, IP Multicast brings a much more efficient way to distribute content and that, despite its barriers to adoption, it can still be used in same cases, namely on internal ISP networks. Both BitTorrent and IP Multicast have drawbacks that we advocate can be offset by using them together. BitTorrent produces a lot of network overhead, with the same information being sent multiple times, and IP Multicast is not reliable neither available everywhere. Our goal was to study and propose a content distribution solution that combined BitTorrent and IP Multicast, study its performance and propose a possible usage scenario.

We established the architecture and details of the implementation of a solution that takes this approach and named it Bitocast. We adapted a simulator to test our suggestions and run it in several scenarios as close as possible to reality. We first directly compared BitTorrent to Bitocast and analyzed the performance differences. Then we evolved Bitocast to support a rich media distribution solution.

We concluded that:

- When compared to BitTorrent, Bitocast reduces the download time. It is much smaller when in reduced bandwidth availability. The reduction is greatest if the proportion download bandwidth versus upload is higher;
- The network usage of BitTorrent decreased significantly meaning a better use of the overall network;
- By using IP Multicast the number of pieces served by each peer is also reduced;
- Due to the download time being shorter, the number of seeds is higher in Bitocast when compared to BitTorrent;
- Regarding download times, Bitocast clearly puts each node in clusters according to its bandwidth, reducing the difference in performance for the same capacities;
- Bitocast augments the capabilities of the system to support *flash crowds* because of the increase in the number of seeds;
- Bitocast can be deployed in co-existence with BitTorrent to enable the improvement of the distribution experience without interference.

Our work suggests that Bitocast is a viable solution to be implemented in a real world scenario when IP Multicast is a possibility such as in private networks.

## 8.1 Summary of Contributions

Our work produced the following contributions:

- Original work:
  - We proposed and studied a new content distribution model that mixes two existing technologies: BitTorrent and IP Multicast. Several different scenarios of using the proposed solution were analyzed and the best ways to use it were depicted.
  - A business use for such a solution was proposed and studied.
- Community contribution:
  - We extended BTSim (an existing BitTorrent simulator) in order to study the impact of our solution.

## 8.2 Future work

There are several enhancements that can be studied, tested and possibly applied to the proposed solution. Besides enhancements, more thorough analysis can be made in order to understand other implications of the proposed solution. We conclude with a list of suggestions for future work.

In order to deepen the understanding of Bitocastit will be worthwhile to:

- Implement Bitocast as a prototype in real network conditions;
- Analyze the benefits of using Bitocast at the network level. Comparing the number of physical packets passed on the overall network for the distribution of the same file with BitTorrent and Bitocast;
- Extend the simulator in order to evaluate a partial multicast scenario. This could be done by not allowing some peers to connect to multicast channels. In the current simulator implementation the peer does not connect to the multicast channels if the overall bandwidth for multicast is 0% or if the available bandwidth is not enough to connect to any existing channel;

- Extend the simulator in order to implement the unreliability of IP Multicast;
- Further study the Rich Media distribution solution regarding the best parameters to use.

In order to enhance Bitocast:

- Introduce dynamic bandwidth management mechanisms to Bitocast. These mechanisms should dynamically enhance the allocation of % of peer bandwidth to IP Multicast vs BitTorrent based on several conditions like network availability and number of clients requesting the file;
- Study the use of Erasure Codes within Bitocast. Erasure codes bring the benefit of resilience to packet loss and have been used efficiently for file transfers [43];
- Implement measures to avoid or overcome malicious users. BitTorrent and IP Multicast would need to be considered.

# Bibliography


[1] *Multitracker metadata entry specification*, 2009 (accessed August 20, 2009). http://www.scribd.com/doc/13839289/BitTorrent-MultiTracker-Specification.

[2] *Bittorrent Protocol Specification v1.0*, 2009 (accessed January 2, 2009). http://wiki.theory.org/BitTorrentSpecification.

[3] *BBC Multicast radio and television*, 2009 (accessed January 20, 2009). http://www.bbc.co.uk/multicast/.

[4] *BitTorrent (protocol)*, 2009 (accessed July 23, 2009). http://en.wikipedia.org/wiki/BitTorrent_(protocol).

[5] *Microsoft Research Simulator of the Bittorrent Protocol*, 2009 (accessed March 12, 2009). http://research.microsoft.com/projects/btsim/.

[6] *BitTorrent Strategies: The End Game*, 2009 (accessed March 19, 2009). http://niallohiggins.com/2007/12/26/bittorrent-strategies-the-end-game/.

[7] *Microsoft: No question downloads will overtake retail*, 2009 (accessed March 20, 2009). http://www.mcvuk.com/news/31441/Microsoft-No-question-downloads-will-overtake-retail.

[8] Firasath Riyaz Arun Chokkalingam. *Bittorrent Protocol Specification v1.0 (1.2 Structure of Torrent(MetaInfo) File)*, 2009 (accessed August 15, 2009). http://cs.ecs.baylor.edu/~donahoo/classes/5321/projects/bittorrent/BitTorrent%20Protocol%20Specification.doc.

[9] A. R. Bharambe, C. Herley, and V. N. Padmanabhan. Analyzing and improving a bittorrent networks performance mechanisms. 2006.

[10] Ruchir Bindal, Pei Cao, William Chan, Jan Medval, George Suwala, Tony Bates, and Amy Zhang. Improving traffic locality in bittorrent via biased neighbor selection. In *International Conference on Distributed Computing Systems 2006*.

[11] Miguel Castro, Peter Druschel, Anne M. Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. Splitstream: high-bandwidth multicast in cooperative environments. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 298–313, New York, NY, USA, 2003. ACM Press.

[12] Bin Cheng, Hai Jin, and Xiaofei Liao. Rindy: A ring based overlay network for peer-to-peer on-demand streaming. In *Ubiquitous Intelligence and Computing Conference*, pages 1048–1058, 2006.

[13] Bin Cheng, Xuezheng Liu, Zheng Zhang, and Hai Jin. A measurement study of a peer-to-peer video-on-demand system. In *International Workshop on Peer-to-Peer Systems (IPTPS)*, 2007.

[14] Bin Cheng, Lex Stein, Hai Jin, Xiaofei Liao, and Zheng Zhang. Gridcast: Improving peer sharing for p2p vod. *ACM Trans. Multimedia Comput. Commun. Appl.*, 4(4):1–31, 2008.

[15] Xu Cheng, Cameron Dale, and Jiangchuan Liu. Understanding the characteristics of internet short video sharing: Youtube as a case study. *School of Computing Science. Simon Fraser University*, Jul 2007.

[16] Yang-Hua Chu, Sanjay G. Rao, and Hui Zhang. A case for end system multicast. In *SIGMETRICS '00: Proceedings of the 2000 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, volume 28, pages 1–12, New York, NY, USA, June 2000. ACM Press.

[17] Bram Cohen. Incentives build robustness in bittorrent. Technical report, bittorrent.org, 2003.

[18] Douglas Comer. *Internetworking with TCP/IP: principles, protocols, and architecture*, volume 1. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 4th edition, 1988.

[19] Stephen E. Deering and David R. Cheriton. Multicast routing in datagram internetworks and extended lans. *ACM Transactions on Computer Systems*, 8:85–110, 1990.

[20] Christophe Diot, Brian Neil, Bryan Lyles, Hassan Kassem, Brian Neil Levine, and Doug Balensiefen. Deployment issues for the ip multicast service and architecture, 2000.

[21] Diego Doval and Donal O'Mahony. Overlay networks: A scalable alternative for p2p. *IEEE Internet Computing*, 7(4):79–82, 2003.

[22] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol – http/1.1, 1999.

[23] Hugh W. Holbrook and David R. Cheriton. *IP Multicast Channels: Express Support for Large-scale Single-source Applications*. Applications, Technologies, Architectures, and Protocols for Computer Communication. Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication., 1999.

[24] P. Ekler I. Kel´enyi and Zs. Pszota. *SymTorrent*, 2009 (accessed August 21, 2009). `http://amorg.aut.bme.hu/projects/symtorrent`.

[25] Johannes M. Bauer Imsook Ha, Steven S. Wildman. *The Economics of Internet Video Distribution*. Department of Telecommunication, Information Studies, and Media - Quello Center for Telecommunication Management and Law, 2008.

[26] M. Izal, Urvoy G. Keller, E. Biersack, P. Felber, A. Hamra, and Garces L. Erice. Dissecting bittorrent: Five months in a torrents lifetime, 2004.

[27] Zhengzhou Jianyong Li Daoying Huang Jianhua Huang Chengren Liang Jianchun Li Jie Zhang Sch. of Comput. Commun. Eng., Zhengzhou Inst. of Light Ind. *An Extension of HAVE Message in BitTorrent Systems*. Fifth International Conference on Grid and Cooperative Computing Workshops, 27-30 Nov. 2006. `http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4146299`.

[28] Dan Jurca, Jakob Chakareski, Jean-Paul Wagner, and Pascal Frossard. Enabling Adaptive Video Streaming in P2P Systems. *IEEE Communications Magazine*, 45(6):108–114, 2007.

[29] Dimitrios Katsaros, George Pallis, Konstantinos Stamos, Athena Vakali, Antonis Sidiropoulos, and Yannis Manolopoulos. Cdns content outsourcing via generalized communities. *IEEE Trans. on Knowl. and Data Eng.*, 21(1):137–151, 2009.

[30] Nikolaos Laoutaris, Damiano Carra, and Pietro Michiardi. Uplink allocation beyond choke/unchoke: or how to divide and conquer best. In Arturo Azcorra, Gustavo de Veciana, Keith W. Ross, and Leandros Tassiulas, editors, *CoNEXT*, page 18. ACM, 2008.

[31] Arnaud Legout, Guillaume Urvoy Keller, and Pietro Michiardi. Understanding BitTorrent: An Experimental Perspective. Technical Report, 2005.

[32] Keong Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *Communications Surveys & Tutorials, IEEE*, pages 72–93, 2005.

[33] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. pages 53–65, 2002.

[34] S. Naicken, B. Livingston, A. Basu, S. Rodhetbhai, I. Wakeman, and D. Chalmers. The state of peer-to-peer simulators and simulations. *SIGCOMM Comput. Commun. Rev.*, 37(2):95–98, April 2007.

[35] The Network Simulator ns-2 (v2.1b8a). http://www.isi.edu/nsnam/ns/, October 2001.

[36] Toni Paila, Michael Luby, Rami Lehtonen, Vincent Roca, and Rod Walsh. Flute — file delivery over unidirectional transport. Internet RFC 3926, October 2004.

[37] C. Northcote Parkinson. *Parkinson's Law and Other Studies in Administration*, 1957.

[38] Ning Wang & George Pavlou. *An Efficient Routing Protocol with Group Management for Peer-to-Peer Multicast Applications*. Centre for Communication Systems Research. University of Surrey. United Kingdom.

[39] Jani Peltotalo, Sami Peltotalo, Alex Jantunen, Lassi Vaatamoinen, and Jarmo Harju. A massively scalable persistent content distribution system. *Proceeding (575) Communications, Internet, and Information Technology - 2007*, 2007.

[40] Gang Peng. Cdn: Content distribution network. *CoRR*, cs.NI/0411069, 2004.

[41] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, volume 31, pages 161–172. ACM Press, October 2001.

[42] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218, 2001.

[43] Rob Sherwood, Ryan Braud, and Bobby Bhattacharjee. Slurpie: A cooperative bulk data transfer protocol. In *Proceedings of IEEE INFOCOM*, March 2004.

[44] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for internet applications. In *ACM SIGCOMM*, pages 149–160, 2001.

[45] Wikipedia. Iptv. http://en.wikipedia.org/wiki/IPTV, April 2009.

[46] Beau Williamson. *Developing IP Multicast Networks: The Definitive Guide to Designing and Deploying CISCO IP Multi- Cast Networks*. Cisco Press, 1999.

[47] Meng Zhang, Jian-Guang Luo, Li Zhao, and Shi-Qiang Yang. A peer-to-peer network for live media streaming using a push-pull approach. In *MULTIMEDIA '05: Proceedings of the 13th annual ACM international conference on Multimedia*, pages 287–290, New York, NY, USA, 2005. ACM.

# Appendix A

# Bitocast Simulator Parameters

Bitocast is compiled as an executable console application that can be run at the command prompt. The parameters for its execution can be passed via command line arguments or/and in a workload file.

Here is the list of command line arguments:

| Command | Description |
|---|---|
| -w | Identifies the file that has the list of commands to be executed by the simulator.<br>argument: workload_file |
| -o | the file where simulation output is sent ("default.out")<br>SimParameters.outputFile |
| -t | the time (milliseconds) for which the simulation continues<br>SimParameters.simulationTime |
| -j | the time (milliseconds) for which nodes continue coming into the system (not used by current simulator, IIRC)<br>SimParameters.joinTime |
| -jr | "Join Rate" - joins per second<br>Join Leave model is also affected<br>SimParameters.joinRate = [rate]<br>SimParameters.stayTimeParams =staytime:uniform:100:300 |
| -slp | Defines the probabily of the seed (after download complete) leaving. default is 0.2<br>SimParameters.seedLeavingProbability |

| | |
|---|---|
| -sfb | if used, this would say "stay as a seed until you serve" these many blocks |
| -rnd | Used to create the random number generator.rnd_rng = new Random(); (if true)dtm_rng = new Random(1111); (if False) |
| -maxu | max# of concurrent uploads that can be going on. min=1 (optimistic unchoke) |
| -b | defines the size (kilo bits) of each block of the file. default is 256 * 8<br>SimParameters.blockSize |
| -c | invoke the bittorrent choker every these many milliseconds (BT default = 10seconds) |
| -r | how many pieces to get in "random" mode first. after these many pieces are received, mode changes to LR. |
| -sbw | defines the seed upload capacity (in Kbps). default is 3000. SimParameters.seedBandwidth |
| -fec | FEC = k ==> "make" k times as many unique packets |
| -fsize | defines the size (kilo bits) of the file to be transferred. default is 100 * 1024 * 8<br>SimParameters.fileSize |
| -d | number of peers to return when the node first asks; in the paper, (d =~ 1.5 * nInitialPeers) |
| -bw | Link capabilities available and their distribution.<br>Affects SimParameters.bwProbabilities<br>Sample:<br>now setparam bw_model 56:56:0.2 784:128:0.25 1500:400:0.25 6000:3000:0.2 100000:100000:0.1 end |
| -ibw | Do not understand this. used in optimistic unchoke. |
| -nou | disable/enable optimistic unchoke in BitTorrent. careful: one must employ some other bootstrapping mechanism as well in this case |
| -pairtft | |
| -grouptft | |
| -fht | number of blocks a connection can be off by (in pairwise/group TFT) |
| -nini | #blocks a node is "given" when joining the system. if 0 no blocks are given. |
| -pint | how often to print node utilization |
| -seeds | #seeds to start with initially |
| -permutations | if not passed ChoosingPolicy is LR |
| -smartseed | should the seed act smart? refer paper for 'smartseed' policy. |
| -nsu | should the seed NOT have unfinished transfers? normally, 'smartseed' implies this. used only in Connection.Choke() |

| | |
|---|---|
| -nwb | node-with-block<br>Argument: period:percentage :downcap/upcap |
| -pfcbatches | Argument: offset:interval: endtime:batchsize: downcap/upcap |
| -sptft | special pairwise tft |
| -maxnodes | hard-coded limit for the active nodes in the simulator. |
| -originload | maximum #file copies to serve. After serving these many blocks, the seed will go offline. If Negative Seed will not die. |
| -tmb | should the tracker be bandwidth-aware in giving out nodes? |

Bitocast Specific Parameters:

| Command | Description |
|---|---|
| -mc | Defines a series of IP Multicast Channels |
| -mcs | IP Multicast Scheduler |
| -btc | Will Bittorrent cooperate with what will be delivered via multicastfalse= normal bittorrenttrue = changed piece selection algorithm |
| -mcr | Multicast Repeat - If the multicast channels will serve more than once each channel (true) or just once (false). |
| | |
| -mcb | Multicast Block Model |
| -share | Multicast BW versus BitTorrent - How is the total bandwidth of the channel shared between Multicast and bittorrent ? 0,3 =30% means 30% for MC and 70 for BT ; |
| | |
| -cont | Indicates that simulator will work as if we were in a continous mode, with BT working with a sliding window of "sub torrents" |
| -nst | Subtorrents that will be available when in SimParameters.ContinousMode=true (Window) |
| -cw | the size of the sliding window in ContinousMode |
| -cbw | The bandwith that will be simulated in continuous mode to find what is the current subtorrent window |
| | |

# Appendix B

# Bitocast Simulator workload file

**.nds - output_stream - One line per simulator's event.**

# Format of the file: [time] [node] [event] <details>

Is written by Logger.cs in node_log(

Sample:

970 4 join leech B d 56 u 561998921 242 r p 222 n 43

| | |
|---|---|
| LogEvent.JOIN | ("{0} {1} join {2} B d {3} u {4}", timeNow, node.ID, (node.IsSeed ? "seed" : "leech"), node.DownCap, node.UpCap); |
| LogEvent.SEEDIFY | ("{0} {1} seedified",timeNow, node.ID); |
| LogEvent.LEAVE | ("{0} {1} leave", timeNow, node.ID); |
| LogEvent.RECV | ("{0} {1} r p {2} n {3}", timeNow, node.ID, (int) piece, (Node).ID; |
| LogEvent.SEND (NOT USED) | ("{0} {1} s p {2} n {3}", timeNow, node.ID, (int) piece, (Node).ID; |
| LogEvent.FINISHED | ("{0} {1} finished sent {2:f3} recv {3:f3}", timeNow,node.ID, (float) node.TotalSent / (float) SimParameters.blockSizeInBits, (float) node.TotalReceived / (float) SimParameters.blockSizeInBits); |
| LogEvent.UTIL | ("{0} {1} d {2} u {3}", timeNow, node.ID, (float) args[1], (float) args[2]); |
| LogEvent.CHGRATE | ("{0} {1} c {2}->{3} r1 {4} r2 {5}", timeNow, node.ID, ((Node) args[1]).ID, ((Node) args[2]).ID, (float) args[3] /* oldrate */, (float) args[4] /* newrate */); |

**.bw - timed_stream - Collection of all nodes in the system at each dump time.** At the UtilDumpEvent it writes to disk: time [The simulator Time] and then one line for each node in the simulation. Each line has the result of node.Dump(): **Sample:** time 10001 #d 0 0 #u 3 168 p 400 0 s 1 #p 3 0 3 0 3 3 3 D 02 #d 1 56 #u 0 0 p 0 0 s 0 #p 3 1 3 1 0 3 0 D 1 Every line contains the following information about each node:

| | | |
|---|---|---|
| {0} | this.ID | The ID of the node |
| #d {1} | m_Downloads.Count | Number of element in the list of ongoing downloads |
| {2} | GetTotalDownloadRate() | The sum of Transfer Rates of the list of ongoing downloads (GetTotalTransferRate) - This only re |
| #u {3} | m_Uploads.Count | Number of element in the list of ongoing uploads |
| {4} | GetTotalUploadRate() | The sum of Transfer Rates of the list of ongoing uploads (GetTotalTransferRate) |
| p {5} | m_FinishedPieces | number of finished blocks |
| {6} | GetUnfinishedPieces() | number of currently pieces in middle of download |
| s {7} | (m_AmSeed ? "1" : "0") | If node is seed or not |
| #p {8} | m_Connections.Count | Number of active connections |
| {9} | interested | Number of nodes that we are interested in |
| {10} | allowed | Number of nodes that are not choking us |
| {11} | useful | Number of nodes that we are interested in and that are not choking us |
| {12} | uinterested | Number of nodes that are interested in us |
| {13} | uallowed | Number of nodes that we are not choking |
| {14} | uuseful | Number of nodes that are interested in us and that we are not choking |
| D {15} | (m_Distance == -1 ? 999 : m_Distance) | The distance from the seed node (node.ID=1) |

{0} #d {1} {2} #u {3} {4} p {5} {6} s {7} #p {8} {9} {10} {11} {12} {13} {14} D {15}

**.gph - graph_stream - used by the -GraphDumpEvent**in each DumpEvent it lists all Nodes, and for each node a list of all connected peers. One Node per line. [Node id] > [NodeConnected] _ [NodeConnected] _ [NodeConnected] _ ...

Sample:

time 610001 -> 124 123 122 120 119 116 115 114 109 103 101 100 98 96 94 92 85 84 83 82 81 75 70 69 68 67 66 64 63 61 58 56 55 54 53 51 50 49 48 46 45 44 43 42 41 40 39 38 37 36 35 34 32 31 30 28 27 26 25 24 22 21 20 19 18 17 16 15 14 13 12 11 10 8 7 6 5 4 3 22 -> 123 119 116 114 112 107 106 104 103 100 99 95 90 88 86 85 82 80 78 77 74 70 68 67 66 65 63 61 60 59 58 57 56 54 52 51 50 49 48 47 45 44 42 41 40 39 38 37 36 35 33 32 31 30 29 28 27 26 24 23 22 21 20 19 18 17 16 15 14 13 12 10 9 8 7 6 5 4 3 1**.prm - created by command print_param in workload file - Dumps the parameters of the** simulation.