MASTER THESIS

TITLE: An Analysis of incentives mechanisms and evaluation on BitTorrent

MASTER DEGREE: Master in Science in Telecommunication Engineering & Management

AUTHOR: David Serra Pàmies

DIRECTOR: Roc Messeguer Pallarès

DATE: October 27th 2013

Títol: An Analysis of incentives mechanisms and evaluation on BitTorrent

Autor: David Serra Pàmies

Director: Roc Meseguer Pallarès

Data: 27 d'octubre de 2013

Resum

D'ençà l'aparició de les primeres xarxes peer-to-peer, el seu nombre d'usuaris no ha cessat de créixer com a conseqüència dels beneficis que presenten en comparació amb les altres arquitectures alternatives per a la compartició i distribució de contingut multimèdia. Tanmateix, atesa la seva natura distribuïda, poden experimentar un important problema de mal ús: el free-riding. El free-riding consisteix en que alguns usuaris consumeixin recursos sense contribuir al sistema. Aquest comportament no només no és just per a la resta d'usuaris, sinó que també amenaça l'èxit d'aquest tipus de xarxes.

Amb la motivació de posar fi a aquesta conducta van aparèixer els mecanismes d'incentius, que proporcionen al sistema un mètode per a incentivar els nodes i, així, aconseguir que comparteixin els seus recursos amb la resta d'usuaris. En altres paraules, ofereixen a la xarxa la justícia necessària per tal que tots els usuaris gaudeixin de bon rendiment.

Aquest projecte està organitzat en dues parts principals. En la primera s'ha realitzat un estudi exhaustiu sobre l'estat de l'art en relació als mecanismes d'incentius que ha donat com a resultat una classificació segons les característiques dels algoritmes estudiats. Aquest estudi proporciona al lector una primera visió de les fortaleses i debilitats de cada algoritme. En la segona part s'ha construït un escenari de test basat en la virtualització de màquines que ha servit per a avaluar empíricament alguns dels algoritmes estudiats. Finalment s'han realitzat una sèrie d'experiments per a comparar determinades característiques d'aquests algoritmes i, d'aquesta manera, s'ha pogut confirmar o desmentir les conclusions fruit de l'estudi de l'estat de l'art.

Title: An Analysis of incentives mechanisms and evaluation on BitTorrent

Author: David Serra Pàmies

Director: Roc Meseguer Pallarès

Date: October 27th 2013

Overview

Since the first peer-to-peer communities appeared, their number of users has increased considerably owing to the benefits they offer compared to their alternative architectures in the sharing and distribution of multimedia content. However, due to its distributed nature, they can suffer an important problem of misuse: free-riding. Free-riding consists on users consuming resources without contributing to the system. Such behaviour not only is not fair for the rest of the users, but also threatens the success of this type of nets.

With the motivation to avoid free-riding, the mechanisms of incentives were born. They provide the system with a method to motivate the nodes and make them share their resources with the other users. In one word, they provide the net with the needed fairness to achieve a good performance for all users.

This thesis is organised in two main parts. In the first part there is a comprehensive study of the state of the art regarding the incentive mechanisms, resulting in a classification depending on the characteristics of the studied algorithms. That study provides the reader with a first sight of the strengths and weaknesses of each algorithm. In the second part there is a test scenario based in the virtualization of machines that was useful to evaluate empirically some of the studied algorithms. Finally, a series of experiments were carried out in order to compare some characteristics of these algorithms and thus verify or deny the conclusions resulted in the study of the state of the art.

“I dedicate this thesis to my professors, parents, friends, classmates and Sònia who always encouraged me to achieve my goals and that without their support I could not have completed this work.”

Thank you all.

INDEX

INTRODUCTION

Since the first peer-to-peer communities appeared, the use of such nets has increased considerably, becoming more and more popular. However, they suffer from a fundamental problem of unfairness due to the fact that many users in these communities try to use these network resources while they contribute little or not at all. Providing the net with the needed mechanisms so that it is fair for all its users is needed for the success of these networks.

Today there are numerous peer-to-peer nets with different algorithms of incentives using a range of mechanisms. Therefore, the motivation of this master thesis is to study and present in a clearly way these algorithms and accomplish an experimental studio which will confirm or deny the benefits provided by each algorithm.

The peer-to-peer technologies and the incentives for the peers are presented in the first chapter, as well as an analysis of the existing methods and problems to be solved by the algorithms related to security and performance. After that, the two main objectives to be accomplished during this thesis will be tackled. The first objective of this master thesis is to analyse the state of the art related to the different existing algorithms detailing their characteristics and profits according to their authors, and provide a classification by characteristics of the algorithms. The second important goal is to build an scenario where different peer-to-peer algorithms could be experimentally tested and compared. Those scenarios should be capable to obtain experimental results allowing to compare several algorithms among others and confirm the results obtained in the theoretic comparative. Finally, it is pretended to verify or deny the presumed profits provided by each algorithm.

This master thesis is organized into five chapters:

An introduction to peer-to-peer technology is provided in the first chapter. It describes what that technology is, the different types of existing peer-to-peer networks and the motivation of its existence. Then it is compared with other existing paradigms in existing networks. Related to the incentive mechanisms, there is an explanation about what they are and why they are necessary. Finally, the key concepts of incentive mechanisms are defined for a proper understanding.

The second chapter contains an analysis of the detail of the state of the art in terms of incentive mechanisms. It presents the classification most widely accepted within the research communities in incentive mechanisms based on how they work. Each algorithm is presented giving its features and compared among the others in order to identify the strengths and weakness of each of them.

The third chapter corresponds to the beginning of the experimental part. After the state of the art research some of the algorithms are tested to obtain an experimental view of the algorithm performance. That will confirm or deny the features and strengths presented by their authors in their papers. In this chapter the technologies and scenario used to carry on that test
beds are presented.

The fourth chapter presents the results of the experimental works comprised by several testbeds. In this experimental part, the fairness, the resilience to strategic clients and the convergence time are the features tested.

Finally in the fifth chapter the conclusions obtained during all the study are evaluated.

CHAPTER 1. PEER-TO-PEER COMMUNITIES

1.1. Introduction

In the beginning of the first computer networks the client-server architecture was the predominant. In such networks, clients consume information from a single server. In the course of time, new technologies and new multimedia contents appeared. This lead to new user needs. This kind of information caused an exponential increase of the size of the information to be transferred. Along with that, the content sharing evolved from a paradigm of a few information sources with many clients to a new paradigm where all the users create and consume contents.

In this new scenario, the first peer-to-peer communities began to grow because it provides new features and capabilities to make more efficient to share and distribute multimedia information. This new architecture fits better to the new scenario and provides several benefits such as load balancing, performance improvement and fault tolerance by decentralization, among others. However, it also brings new problems due to the decentralization. For instance, new security issues.

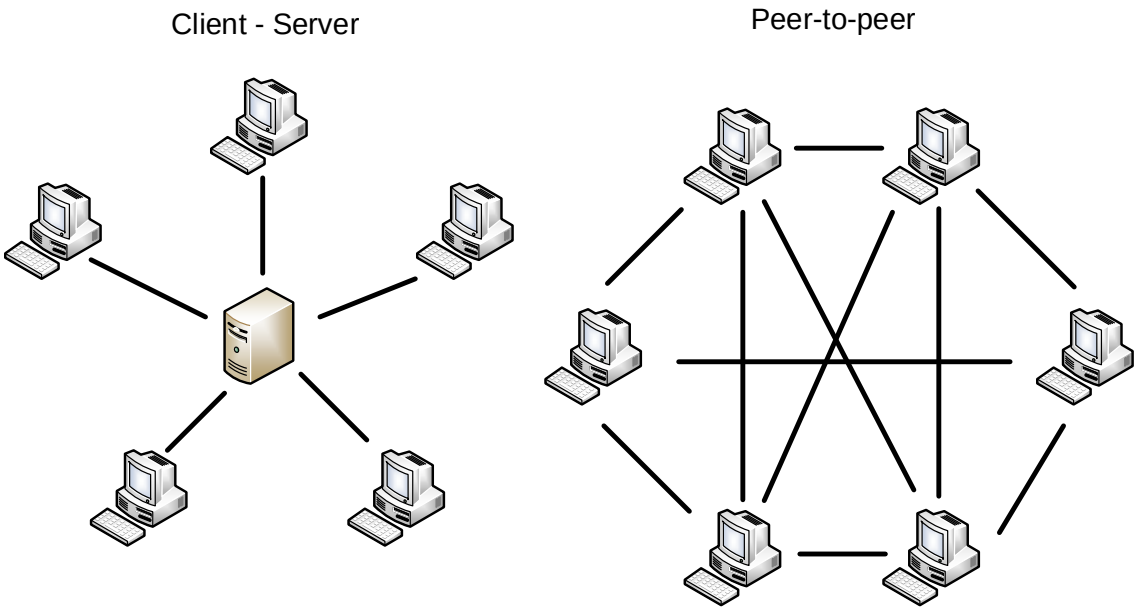

Figure 1.1: Client – Server versus Peer-to-peer architecture

From the beginning, these issues have been tried to be solved and therefore the peer-to-peer system has enhanced a lot. Additionally new features and

information formats have led to new specific peer-to-peer systems like peer-to-peer live video broadcasting.

1.2. What is a Peer-to-peer Network?

A peer-to-peer computer network is one in which each computer in the network can act as a client or server for the other computers in the network, allowing shared access to various resources such as files, peripherals and sensors without the need for a central server [27].

Nowadays many different peer-to-peer networks exist. Each one of them requires that all the computers in the network use a compatible software using the same protocol to connect to each other and access to the resources found on other computers. Peer-to-peer networks can be used for sharing content such as audio, video, data or anything in digital format.

Peer-to-peer file transfer protocols provide more scalable architectures for distributing large files than the traditional server-client paradigm. The idea is that the peers that are downloading also contribute uploading to the system, thus scaling the available bandwidth as more peers join the system. Even centralized services with large network connections can be overwhelmed by a large number of clients, while peer-to-peer services can ostensibly continue to scale, even in extreme scenarios.

Peer-to-peer content distribution systems range from relatively simple direct file sharing applications, to more sophisticated systems that create a distributed storage infrastructure for securely and efficiently publishing, organizing, indexing, searching, updating and retrieving data.

Peer-to-peer systems often implement an abstract overlay network, built at Application Layer. Such overlays are used for indexing and peer discovery and make the peer-to-peer system independent from the physical network topology. Based on how the nodes in the overlay network are linked to each other, two general categories of systems can be identified in this respect: the unstructured and the structured networks.

In structured peer-to-peer networks, peers are organized following specific criteria and algorithms, which lead to overlays with specific topologies and properties.

Unstructured peer-to-peer networks do not impose any structure on the overlay networks. Peers in these networks connect in an ad-hoc fashion based on a loose set of rules. Ideally, unstructured peer-to-peer systems would have absolutely no centralized elements, but in practice there are several types of unstructured systems with various degrees of centralization. Three categories can easily be identified [27]:

- In pure peer-to-peer systems the entire network consists solely of equipotent peers. There is only one routing layer, as there are no preferred nodes with any special infrastructure function.
- In centralized peer-to-peer systems, a central server is used for indexing functions and to bootstrap the entire system. Although it has similarities with a structured architecture, the connections between peers are not determined by any algorithm.
- Hybrid peer-to-peer systems allow such infrastructure nodes to exist.

Some advantages of Peer-to-peer networking over client–Server networking are [27]:

- All the resources and contents are shared by all the peers, unlike server-client architecture where server shares all the contents and resources.
- Peer-to-peer is more reliable as central dependency is eliminated. Failure of one peer does not affect the functioning of other peers. In case of client–server network, if server goes down, the whole network gets affected.
- There is no need for full-time system administrators. Every user is the administrator of his own machine. User can control their shared resources.
- The overall cost of building and maintaining this type of network is comparatively very less.

But peer-to-peer architecture has also some drawbacks over a client-Server one [27]:

- In this network, the whole system is decentralized thus it is difficult to administer. That is, one person cannot manage by itself the whole network. Each user manages its node in the network.
- Security in this system is lower. Virus and other malwares can be easily transmitted over this peer-to-peer architecture.

1.3. Selfish peers and incentive mechanisms

Peer-to-peer content distribution networks are powerful systems that use the bandwidth resources of their users. However, in those networks, the performance is highly dependent on the user's willingness to contribute with their bandwidth. Selfish users tend not to share their bandwidth and only want their own benefit. Some studies have showed that in the real peer-to-peer communities, a large number of peers become selfish if the system does not implement any kind of incentive mechanism. For example, in Gnutella more than 70% could behave as free-riders [21].

So peer-to-peer networks suffer from a fundamental problem of unfairness. Free-riders cause slower download times for others by contributing little or no upload bandwidth while consuming much download bandwidth. Cooperation is

a key to the success of a peer-to-peer system, but it is difficult to encourage without an incentive mechanism.

In order to be successful, a peer-to-peer network has to be fair and, therefore, it has to implement an effective incentive mechanism. Fair bandwidth allocation in peer-to-peer systems can be difficult to achieve for several reasons [1]:

- No central entity controls and arbitrates access to all resources.
- The amount of bandwidth resources available is not known in advance and peers cannot be relied upon to specify their own resources honestly.
- Strategic peers and free-riders may try to take advantage of the system by contributing little or nothing to the bandwidth, while consuming others' resources.

Developing an effective incentive mechanism is not an easy task. This implies a thorough statistical analysis of existing networks and their peers. Some attempts to address this fair bandwidth allocation problem may suffer from slow peer discovery, inaccurate predictions of neighbouring peers' bandwidth allocations, underutilization of bandwidth, and complex parameter tuning. All this can lead to a worse or even an unusable network due to a bad incentive mechanism [1].

1.4. Incentives mechanisms to bring fairness

Incentives play an inherently crucial role in a peer-to-peer system. Users generally wish to download their files as quickly as possible and since peer-to-peer is a decentralized system, users are therefore free to attempt to strategically manipulate others into helping themselves to download faster. The role of incentives in peer-to-peer is to motivate users to contribute their resources to others so as to achieve desirable global system properties.

Usually, the incentives are resources that the network can give to a peer in order to motivate it. Among others, some of these incentives can be [8]:

- Uploading bandwidth: A local peer will divide his uploading bandwidth capacity between downloaders based on their contribution. Then, as much as the local peer contributes to the remote peer, the remote peer will reward it providing a bigger upload rate.

- Number of peers per search: In peer-to-peer systems with an overlay, this is the number of remote peers that the system will provide to a peer. As much as the local peer contribute, the system will provide more remote peers location to provide him with a larger view of the network and therefore a better performance.

- Uncommon segments exchange: The peers revelation strategy dictates the piece revelation to a peer. Building an effective strategy will lead to reward the peers with high cooperation by providing them with uncommon pieces before than doing so to the other peers.

1.5. Attacks related to unfairness in peer-to-peer

Due to the decentralised nature of the peer-to-peer networks, the security in such networks tends to be a weak point. Some users exploit it in order to obtain their own benefit. Although there are several kinds of benefit from attacking a peer-to-peer system, one very extended is to obtain more resources from the system than the resources provided back to the system.

1.5.1. Free-riding

Particularly, for the content delivery peer-to-peer networks, Free-riding means to obtain great downloading rates while the contribution uploading to the network is little or nothing. Users acting that way are called free-riders [24].

Selfish users in a peer-to-peer network only want to consume resources of the network without contributing to the network in order to only improve their own performance. Peer-to-peer networks without an effective incentive mechanism will have a large amount of free-rider users and therefore they will suffer from very bad performance.

Peer-to-peer networks with an incentive mechanism are more proof against free-riders. The main reason is that if a user wants to be a free-rider, he not only has to modify their upload/download maximum ratios, he also has to use more complex tasks like to cheat to the incentive mechanism. It that point, it begins a battle between the free-rider users and the incentive algorithms. Here is the description of some types of attacks used by the free-rider users in a peer-to-peer network .

1.5.2. The large view exploit

The large view exploit is an attack used by free-rider users against peer-to-peer networks such as BitTorrent. The attack is based in a client that acquires a larger than normal view of a BitTorrent swarm and connects to all peers in its view. At the same time, the client does not upload any data to its peers while it downloads from all the peers connected to him.
Experimental results demonstrates that the modified client can achieve better download rates than a compliant client in most common-case public torrents [7].

1.5.3. The sybil attack

Sybil attack is an attack against identity in which an individual entity masquerades as multiple simultaneous identities. In the context of peer-to-peer applications, that is to control multiple peers inside the swarm. A peer-to-peer incentive mechanism, has to manage to be proof to sybil attack. Otherwise, the incentive mechanism cannot work at all. With sybil attack, the attacker can be beneficiated obtaining better performance from attacking some mechanisms.

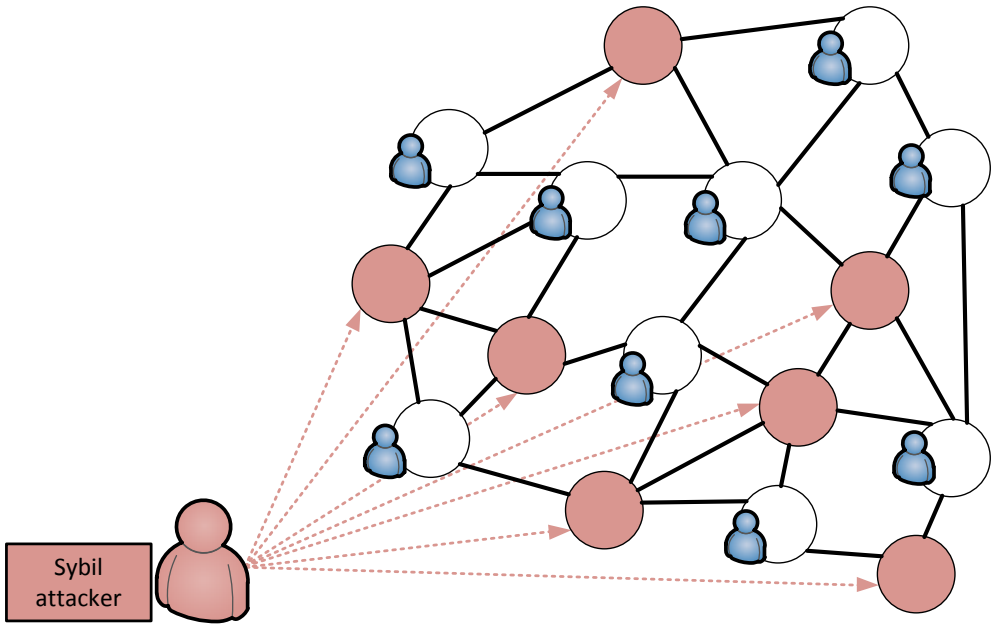

Figure 1.2: Sybil attack in a peer-to-peer network: A unique user has achieved the control of several entities within the swarm

A common example is in the bootstrapping mechanism. When a new peer joins the swarm doesn't have any block. In order to be able to begin to upload information, other peers will have to provide him with new blocks without having previous information about him. These peers will provide him blocks without expecting anything in return, before they realise that the new peer does not share anything at all.

Therefore, an attacker can be generating new identities and for each of them, obtain a few blocks from other peers without nothing in exchange, and finally substitute this identity by another and begin the same procedure again. The result is that in a peer-to-peer system without a protection against the sybil attack, a peer can obtain the file without noting in exchange and with very good performance to him and causing a performance deterioration throughout the swarm.

1.5.4. Collusion

The definition of collusion is an agreement between two or more peers, to limit open competition by deceiving, misleading, or defrauding others to obtain an objective by gaining an unfair advantage. Specifically in peer-to-peer networks, it means that colluders can cooperate with each other and artificially boost their upload-to-download ratios, thereby free-riding the system [22].

Collusion can very effectively against peer-to-peer systems based on a shared reputation of the peers. If there are a large number of colluders in a system based on a shared reputation, the reputation of each peer can be altered providing the capacity to subvert the whole reputation system.

Collusion is a critical issue in peer-to-peer applications. If a large portion of peers in a peer-to-peer system are colluders, the resource in the system will be over-exploited and the system could collapse. Note that a set of colluders can actually be one user who has created several colluding accounts.

The effect of collusion is magnified in systems with cheap pseudonyms, where users can engage in the sybil attack, create fake identities and collude with their own multiple identities.

CHAPTER 2. INCENTIVES SCHEMES STUDY

2.1 Introduction

Having explained the context of the study, in this chapter is analysed the detail of the state of the art in terms of incentive mechanisms. The goal of this chapter is to identify the strengths and weakness of each algorithm.

This survey analyses the main existing incentive algorithms classified about their characteristics at the present. The classification described is currently the most widely accepted classification within the research communities in incentive mechanisms.

There is a number of reasons for users to contribute in a peer-to-peer system. Some of them offer resources in exchange for receiving other resources in the present or future time. Others may do so under threat of retaliation such as the expulsion of the community. Still others do so only out of altruism.

In order to present the incentive mechanism, there has been classified depending on how they work. This classification consist on three main categories of schemes for addressing the free-rider issues. These are inherent generosity, reciprocity-based schemes and monetary payment schemes.

2.2 Inherent Generosity

The first category in which the incentive mechanisms are classified is the inherent generosity. This category include all the peer-to-peer systems in each peer contribute based only on its generosity.

The file sharing networks such as eDonkey and Pruna are uploading and downloading inside established limits, but users can alter their applications to use the network more than they have been allowed.

In order to analyse the behaviour of this kind of systems it exists a model framework that studies the phenomenon of free-riding in peer-to-peer systems [23]. The model is based on the insight that some users gain utility from the mere act of giving. It analytically determines the resulting percentage of free-riders in the system based on the distribution of generosity in the population.

They find that if the societal generosity is below a certain threshold, then there are too many selfish peers around and the system collapses. But if it exceeds the threshold, the contribution level increases in the societal generosity.

2.3 Reciprocity-Based Schemes

This category includes the mechanisms based on reciprocity, meaning that each peer can download if he has uploaded previously. Each one maintain histories of past behaviour of other users and use this information in their decision making processes.

This category is divided into two subcategories: direct reciprocity and indirect reciprocity. The first one, user A, decides how to serve user B only based on the service provided by B to A in the past. Instead of that, in indirect reciprocity schemes, the decision of A also depends on the service provided by B to other users on the system.

2.3.1 Direct-reciprocity

In direct-reciprocity schemes, each user maintains histories of past behaviour of other users and provides a service based on the service provided by each remote peer. There is no history shared among the users in the swarm.

Direct-reciprocity schemes are suitable for applications with long lasting session, providing ample opportunities for reciprocity between pairs of users.

Direct-reciprocity schemes do not need a central entity to control the incentive operation of the whole network. Each user maintains the history and competes in the system like in a market. It has to be able to determine which peers are the best to provide service in order to obtain the best service in return.

In a decentralized incentive mechanism, sybil attacks can have direct consequences for peer-to-peer systems and are difficult to prevent. On the other hand, the decentralisation of the incentive mechanism reduces the costs and problems related to maintain a central entity.

Literature in peer-to-peer incentives usually uses the Iterated Prisoner's Dilemma as a model for understanding cooperation. In the classical Prisoner's Dilemma, two players choose simultaneously whether or not to cooperate. Each of the peers is rewarded if both cooperate, but at a lower rate than the penalty received if one cooperates and the other does not. Hence the dilemma: the rational choice of not cooperating leaves both worse off than if both had cooperated.

The classical Prisoner's Dilemma has to be adapted to the peer-to-peer environment. The model must be extended to more than two participants. It also has to be iterative, that is to play over and over during the session.

The Prisoner's Dilemma can be generalised as a payoff matrix in Table 1, in which depending on cooperation or defection of each peer, it assigns a payoff: R for reward, T for temptation to defect, S for sucker's payoff, and P for

punishment for mutual defection. This payoff reflects the utility values earned after the exchange.

	Cooperate	Defect
Cooperate	R=3, R=3	S=0, T=5
Defect	T=5, S=0	P=1, P=1

Table 2.1: Payoff matrix for the Prisoner's Dilemma

Using the Prisoner's Dilemma is possible to model players with different strategies on deciding how to act. In order to test and design new strategies computer tournaments are used in which every pair in a pool of players is subjected to repeated exchanges.

Here it is important to define what is a Nash equilibrium: In game theory, a Nash equilibrium is a solution concept of a game between two or more players, in which each player is supposed to know the equilibrium strategies of the other players, and no player has anything to gain if only unilaterally changing its own strategy.

In a Nash equilibrium each player has chosen a strategy and no player can benefit by changing its strategy and the other players keep their unchanged set of strategic options and the corresponding gains.

It is important to note that a Nash equilibrium does not imply the achievement of the best overall result for the participants, but only the best result for each considered individually. It is perfectly possible that the outcome would be better for everyone if, somehow, the players coordinate their action.

Within the peer-to-peer scope, therefore the Nash equilibrium is a point in the time where all the peers already know each other's strategies and no one has anything to gain if only unilaterally change its own strategy.

In a simulation environment with many repeated games, persistent identities, and no collusion, the Tit-for-Tat strategy dominates when the goal is the best result among all users. Instead, in a real network, with free-rider users using attacks, the issue is more complex. In Tit-for-Tat strategy, a peer will cooperate unless the remote peer defect. In case of defection in one round by the remote peer, the local peer will defect in the next round.

Direct-reciprocity schemes are the incentive schemes in which more research efforts have been done. The most important mechanisms developed up to date are showed below.

2.3.1.1 Chocking and unchocking mechanism

The chocking and unchocking mechanism is the incentive used in the official BitTorrent client implementation proposed by Bram Cohen [12]. The goal of this mechanism is to bring fairness to all users in the swarm and avoid free-rider users.

The mechanism is based in two possible states of each connection with a remote peer: chocking (not uploading) and unchocking (uploading). A peer maintains the current download rates from all its links. Based on this information, it unchocks the b links with the highest download rates (b defaults to 7 or smaller). All the other links are choked except for one that is allowed by a mechanism called the optimistic unchocking, the purpose of which is to find a better link. The period of the optimistic unchocking should be sufficiently long (30 seconds in BitTorrent 4.0.0) so that this link may be put on the unchocking list of the other peer. If it downloads from this link at a higher rate than some of the b links, this new link replaces the link with the b-th highest rate. Otherwise, another link is chosen for the optimistic unchocking in a round-robin fashion.

Some studies have shown that BitTorrent's Tit-For-Tat heuristic does not result in fair bandwidth exchange. Because it only identifies and exchanges data with a small number of peers at a time, a BitTorrent client may waste much time and bandwidth while discovering peers with similar upload rates in a large network [18].

In addition to the bad bandwidth allocation, it is vulnerable to some free-rider user attacks. Experimental test results show that the mechanism is susceptible to free riding [3]. Some studies show that bitTorrent is vulnerable to the Large view exploit [2]. This mechanism is also vulnerable to sybil attack. A peer can use multiple identities each of one asking for a block without upload nothing in back exploiting the optimistic unchocking mechanism.

2.3.1.2 Tit-for-tat from the Iterated prisoner's dilemma tournaments

Seung Jun and Mustaque Ahamad investigated the incentive mechanism of BitTorrent and proposed a new incentive algorithm based on their experience on Iterated prisioner's dilemma tournaments [3]. The algorithm proposed is the winning entry tit-for-tat of that tournaments. The goal of this mechanism is to bring fairness to all users in the swarm and avoid free-rider users.

In this mechanism, peers maintain the upload amount u and the download amount d for each link. We can define the deficit of a link as u−d. If the constant c denotes the size of a fragment, a peer ensures that the deficit of every link is restricted up to a certain bound at any time:

$$u - d \leq f \cdot c \qquad \textbf{(2.1)}$$

where f (≥ 1) is called a nice factor. Within this condition and the maximum upload rate allowed, the peer uploads evenly to all links as much as it can. This factor determines the amount that a peer is willing to risk for a chance to establish cooperation. Although neighbours may be tempted to take advantage of this nice peer, they will benefit more through the repeated exchange of fragments if they cooperate.

Although this algorithm improves the incentive mechanism provided by the original BitTorrent implementation, it still has some drawbacks. It requires long round durations to estimate bandwidth contribution of the neighbouring peers, and wastes much bandwidth each round before discovering other contributing peers. This results in a bandwidth underutilization and it is also vulnerable to strategic clients.

2.3.1.3 Strategic client: BitTyrant

Piatek, Isdal, Anderson, Krishnamurthy and Venkataramani show that the incentive mechanism of the standard BitTorrent implementation is not robust to strategic clients. Through performance modelling parameterized by real world traces, they demonstrate that all peers contribute resources that do not directly improve their performance [18].

They have modelled and analysed the BitTorrent's current incentive mechanism. They found that although the original algorithm of BitTorrent discourages free-riding, the dominant performance effect, in practice, is an altruistic contribution on the part of a small minority of high capacity peers and this is not a consequence of his chocking and unchcoking TFT algorithm. Selfish peers can significantly reduce their contribution and yet improve their download performance.

They concluded that incentives in original BitTorrent algorithm do not build robustness. Instead of this, BitTorrent works well today simply because most people use client software as-is without trying to cheat the system.

They use these results to drive the design and implementation of BitTyrant. BitTyrant can improve performance only due to more effective use of altruistic contribution of the other peers. So one BitTyrant client in a swarm full of original BitTorrent clients can improve its performance and harm the whole performance of the swarm.

BitTyrant is a strategic BitTorrent client meaning his goal is not to provide fairness to the whole swarm but only to a client be a free-rider.

2.3.1.4 TFT with Proportional Response algorithm

Levin, LaCurts Spring and Bhattacharjee view BitTorrent as an auction instead of a tit-for-tat. With this point of view They show that the unchocking algorithm

of the BitTorrent standard implementation does not yield the fairness and robustness guarantees desired from such a system. With the goal of “the more you give the more you get", they investigate the use of a proportional share mechanism as a replacement to BitTorrent’s unchoker and creates the PropShare implementation [8].

The algorithm works as follows: PropShare runs an auction for a peer i's bandwidth and accepts the bandwidth offer from peer j as j's bid. Then peer i sends to j the proportional upload bandwidth taking into account all the remote peers and the total available upload bandwidth. It also uses a mechanism that reveals strategic blocks to the neighbours only enough to keep neighbours interested.

The proportional Response algorithm provides several benefits in front of the previous algorithms. PropShare is Sybil attack proof and more collusion resistant and therefore it is more free-riding proof.

2.3.1.5 Treat-Before-Trick

Shin, Reeves and Rhee propose a method of preventing free-riding in peer-to-peer systems based on cryptography [15]. This method, called Treat-Before-Trick, is based on secret sharing. The goal of that mechanism is to provide fairness to the whole swarm and avoid free-ride users.

The steps of Treat-Before-Trick are the same as those of the BitTorrent standard implementation, with the exception of key management. Peers are still assumed to use TFT and optimistic unchocking when determining how much resources to share.

Treat-Before-Trick adds the secret sharing to BitTorrent: A file is divided into pieces, encrypted with a symmetric secret key, and then distributed to requesting peers along with subkeys generated. Peers must then swap file pieces for subkeys, which are needed in order to decrypt the file pieces.

The use of secret sharing effectively counters known free-riding techniques, such as the sybil attack, while download time for compliant peers is heavily reduced. The computational cost and the extra bandwidth required for subkey exchange is not very high. However Treat-Before-Trick is subject to attack by malicious peers, who potentially disclose keys but do not profit themselves and collusion between free-riders (exchanging subkeys) is not prevented in current Treat-Before-Trick.

2.3.1.6 Fair Torrent

Sherman, Nieh and Stein presented a deficit-based distributed algorithm called FairTorrent [1]. The goal of this mechanism is to provide fairness to the whole swarm.

They realised that all the mechanisms based on direct reciprocity are rate-based and suffer from a fundamental flaw. First, they require long round durations to estimate bandwidth contribution of the neighbouring peers and waste much bandwidth in that. Second, they assume that a peer's allocation measured in a given round is an accurate estimate of the future contribution from that peer and this assumption is problematic as each peer can change its allocation or even stop uploading to a given peer.

A FairTorrent client uploads a data block to the peer it owes the most data and automatically converges to the individual reciprocation rates of its peers, without measuring or predicting these rates. FairTorrent runs locally at each peer and maintains a deficit counter for each neighbour who represents the difference between bytes sent and bytes received from that neighbour. When it is ready to upload a data block, it sends the block to the peer with the lowest deficit. FairTorrent does not require an estimate of neighbouring peers' rate allocation. Therefore, it does not require rounds for discovering favourable peer sets.

FairTorrent creators compared FairTorrent against BitTorrent, Azureus, PropShare and BitTyrant. They show that FairTorrent provides better degree of fairness, compared with other peer-to-peer systems. They also prove that FairTorrent is the algorithm with a better performance among the others compared with. Up to now FairTorrent is resilient to free-riders, low contributors and strategic peers.

2.3.2 Indirect reciprocity

In indirect-reciprocity schemes, users maintain a shared history of past behaviour of all the users. They provide a service based on the service provided in the past by each remote peer to the whole swarm. Indirect-reciprocity schemes are also called reputation-based schemes in the literature.

The difference from the direct-reciprocity is the computation of reputation scores for each peer. These schemes have the ability to map the scores to strategies applied by the whole swarm.

Indirect-reciprocity schemes are more scalable than direct-reciprocity schemes [22]. However, indirect-reciprocity schemes rely on second-hand observations and thus must confront trust issues that do not arise in direct-reciprocity schemes. Collusion can be very harmful to these peer-to-peer systems if they are not well handled. Colluded peers could alter the reputation of peers in the shared history of the swarm.

2.3.2.1 EigenTrust

EigenTrust is not only an incentive mechanism but a complete reputation mechanism. Each peer is assigned a unique global trust value that reflects the experiences of all peers in the network with peer. In EigenTrust all peers in the network participate in computing these values in a distributed manner.

The reputation value of each peer is not only based in the uploading and downloading rates. It includes other parameters such as if the peer has been sharing inauthentic files. The whole system takes actions against the malicious peers based on the reputation values such as isolate them from the network.

EigenTrust tries to provide the fairness to the swarm. However EigenTrust has many drawbacks. It is vulnerable to collusion and to the sybil attack. A group of peers can lie about the reputation of a peer. Moreover, the distributed computation and management of the reputation is very complex and it causes computation and bandwidth overheads.

2.3.2.2 FOX

FOX is a file-sharing protocol which incentive mechanism is based on an overlay structure [14]. This overlay structure build a logical topology in which the peers are placed. Each peer will only be able to share information with their immediate neighbours.

FOX's structured topology provides a means for peer to punish nodes both upstream and downstream from it. The FOX topology is restructured in a cyclic way. When a peer is no sharing as it should, it will be punished moving it on the topology.

FOX provides optimal download times when everyone cooperates and punishes free-riders. However it has some drawbacks: reforming the structure causes overheads and peers must wait for a new restructuring to even have the possibility of punishing free-riders. Moreover, FOX is unnecessarily strict, as there may be highly provisioned nodes that are willing to give much more to the system as long as they can download more, but FOX does not provision for this.

2.4 Monetary-based schemes

The last category of methods to prevent free-riders from downloading is the Monetary-based scheme. In this category each downloader should pay the downloading fees for resources they consume and there must be a public key infrastructure to add an economic system to the network.

Monetary schemes provide a mechanism to exchange tokens by blocks of information. It allows individual users to make a profit by uploading more than they download and the other way around it allows an individual to pay for download without upload requirements. However, they have a notable drawback: It is high complex since they require an infrastructure for accounting and micropayments.

2.4.1 Dandelion

Dandelion is a monetary-based file distribution protocol that uses currency and key exchanges through a centralized server to provide incentive for sharing across different downloads [13].

Dandelion system is reminiscent of BitTorrent. However, Dandelion uses a different incentive mechanism. It employs a cryptographic scheme for the fair exchange of content uploads for credit, the content provider is able to redeem a client's credit for monetary rewards. Thus, it provides strong incentives for clients to seed content.

Cryptographic scheme provides Dandelion with free-rider robustness and makes the system proof to most of the attacks such as sibyl attack and collusion. However, it has the drawbacks associated to a monetary scheme: a centralised infrastructure for accounting and micropayments and the complexity because of the cryptographic scheme.

CHAPTER 3: USED TECHNOLOGIES AND SCENARIO

3.1 Introduction

Having realised a deep analysis of the state of the art of the algorithms of incentives existing today, now it is pretended to build an scenario that allow to compare experimentally different algorithms. In this chapter, the scenario of tests developed and the technologies used are described.

The scenario used is based on virtualization. In a physical machine, a test scenario has been created. In this scenario, the peer-to-peer network elements are virtual, intending to look as much as possible to reality by performing the same operations than a real peer-to-peer network.

In order to obtain the results in equal conditions and be able to use different algorithms together, the tests have been conducted using the same peer-to-peer protocol. In this case, BitTorrent has been used due to the fact that its specification is free to use and it exists a large number of clients which implement different algorithms of incentives.

3.2 BitTorrent architecture

BitTorrent is a protocol supporting the practice of peer-to-peer file sharing and is used to distribute large amounts of data. BitTorrent is one of the most common protocols for transferring large files in Internet. Programmer Bram Cohen, a former graduate student in Computer Science Major by the University at Buffalo, designed the protocol in April 2001 and released the first available version on July 2nd, 2001. BitTorrent clients are available for a variety of computing platforms and operating systems.

In order to be able to mount the peer-to-peer testbeds, BitTorrent requires the presence of a tracker. A tracker is a server that assists in the communication between peers using the BitTorrent protocol. A tracker maintains all the peers belonging to the swarm identified. Clients are required to communicate with the tracker to initiate downloads. After that, when clients have already begun downloading, they also communicate with the tracker periodically to negotiate with newer peers and provide statistics. However, after the initial reception of peer data, peer communication can continue without a tracker.

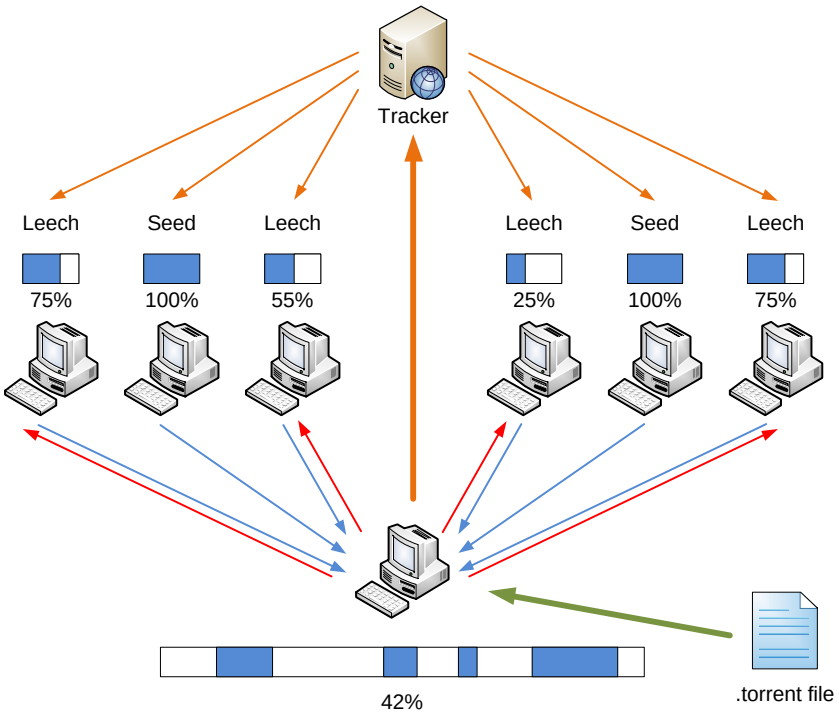

Figure 3.3: Tracker operation

In particular for these tests, the BitTornado tracker has been used. BitTornado has been run in a machine dedicated to this role in all the testbeds done. In the testsbeds, when a machine starts the BitTorrent client, it loads the torrent file containing the information about the file and the tracker. With this information, the client contacts with the tracker and registers itself as a peer in the swarm. The trackers gives to the client the list of the other peers already registered in the swarm.

3.3 Netkit

Netkit [29] is a self-contained environment developed by the Roma Tre University that makes it easy and costless to emulate complex network configurations on a single host machine in order to perform networking experiments. It allows creating several virtual network components that can be easily interconnected in order to form a network on a single PC. Interconnected machines may be organized to form a laboratory, which can be used to emulate the behaviour of a particular service or protocol.

Netkit, in itself, is an open source project aiming at integrating different other open source products. It is heavily based on the User Mode Linux (UML) variant of the Linux kernel. The purpose of this Netkit is to solve many of the difficulties and technicalities that a user could have in using UML for networking.

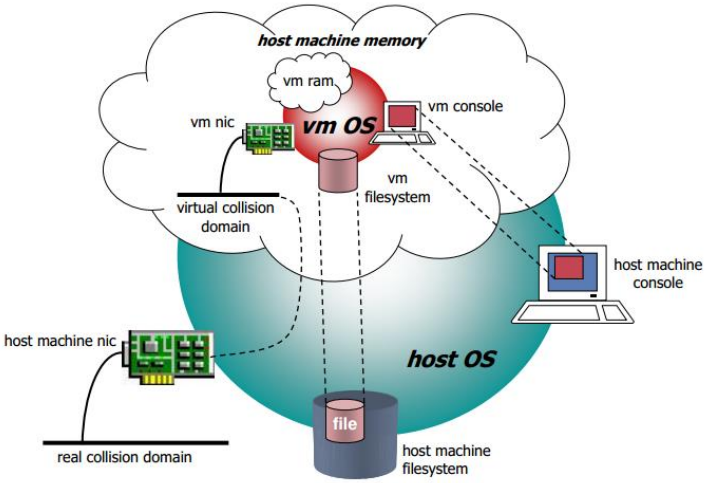

Figure 3.4: Netkit architecture (Roma Tre University)

Netkit virtual nodes can be interconnected between them using virtual hubs. This virtual hubs are a colision domain that allows to connect the network interfaces of the virtual machines. This virtual hubs can be also connected to the real world using virtual interfaces "tap" witch connects the virtual hub with a virtual network interface of the host.

In order to carry out the experimental tests, several virtualization technologies have been compared in terms of features, performance, licensing and simplicity of use among others. The virtualization products proposed were Vmware, Virtual Box, Xen and Netkit with UML. The easy to use and the open source license of use of Netkit with UML tipped the balance in favor of Netkit with UML. It was also important the scriptable way to build and boot an entire scenario used in Netkit.

Regardless the benefits explained above, for the purpose of simulating peer-to-peer networks, Netkit suffers from a significant limitation. Netkit does not allow running applications with graphical user interfaces. This limitation prevents from using some peer-to-peer clients in which the graphical user interface is needed for running. Therefore, all the peer-to-peer systems tested here are able to run without the graphical user interface.

The Netkit version used in all the test beds is the version 2.8, the latest version available as of publication time.

A very interesting feature of Netkit performing the tests has been the Netkit labs. A Netkit lab is a set of preconfigured virtual machines that can be started and halted together. This feature of Netkit has allowed to preconfigure the scenarios and then re-run the experiment as many times as needed. The way in which these scenarios are defined is by using configuration files for the elements in the scenario. In annex 3 are detailed the configuration of the scenarios created for the experiments.

3.4 User-mode Linux

User-mode Linux allows to run several guest Linux kernels as a process within the normal Linux kernel (host). As each guest is just a normal application running as a process in user space, this approach provides the user with a way of running multiple virtual Linux machines on a single piece of hardware.

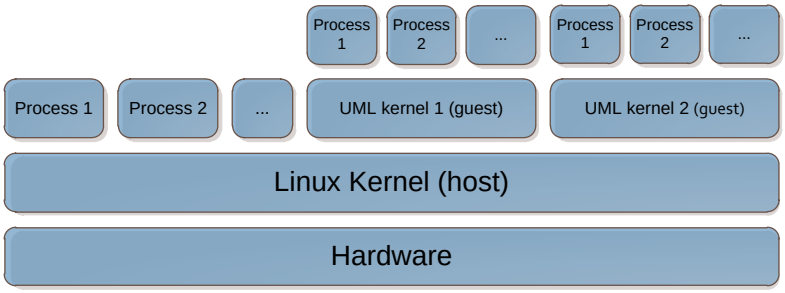

Figure 3.5: Execution levels using UML

Virtualization plays a fundamental role in development and testing applications. In User-mode Linux each guest machine runs as application in user space giving to the user the total control of that machine. That is why User-mode Linux is very useful to test and debug new software, as well as in teaching and research.

User-mode Linux is the technology in which Netkit is based. Netkit takes advantage of this powerful technology improving it to offer a very useful framework for a teaching and researching in networks.

When a User-mode Linux machine is booted, it starts to use a filesystem. This filesystem is contained in a file on the host machine and it is a block device. UML block devices can be layered, with a read-only device having a copy-on-write (COW) read-write device on top of it. This acts as a single read-write device, with the modifications to the read-only layer being recorded in the read-write COW layer. This allows multiple machines to share a filesystem, allowing a large saving in disk space on the host.

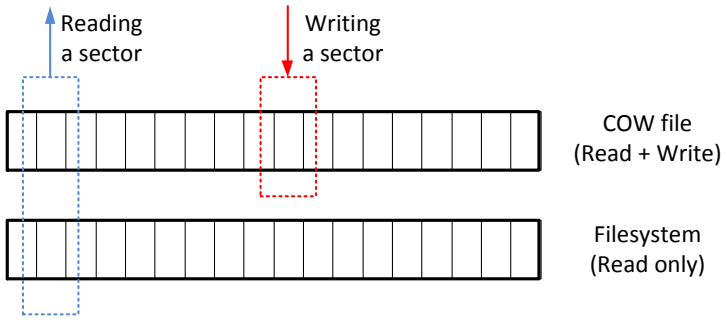

Figure 3.6: Figure Input / Output Operations on a Block Device with COW

In order to perform the testbeds, a unique filesystem for all the machines has been created. This filesystem is based on the filesystem provided with Netkit on its version 5.2 and it has been modified as it is described in the ANNEX 2. It contains all the software needed: The peer-to-peer clients, tracker, tools, etc.

3.5 Hardware

The hardware used to run the laboratories has been a personal computer with the following characteristics:

Processor	Intel Core i5-2500 Quadcore 3,30 GHz
Memory	8 GB of DDR3 SDRAM
S.O.	Ubuntu Linux 12.04 LTS – Linux 3.2.0-37 x86_64
Storage	140GB

Table 3.2: Hardware specifications

It is important to see here that the hardware characteristics of the infrastructure in which Netkit will run are a determinant limitation. Although the system used here is not a low performance desktop, this hardware will limit the number of virtual machines that can be running at the same time on the testbed.

In order to be sure that the performance of each virtual machine does not affect to the results of the testsbeds, some performance tests with virtual machines running peer-to-peer software have been performed. Due to the results obtained in these tests, a limitation of 15 machines running at the same time has been imposed on the testbeds.

3.6 Other tools used

Tc [31] and Wondershaper [32] have been used in order to limit the bandwidth capacities of the peers to perform the different tests. Tc is used to configure Traffic Control in the Linux kernel. Traffic Control consists of shaping, scheduling, policing and dropping traffic. Wondershaper provides an easy way to configure the traffic parameters of Tc using scripts.

Tcpdump [33] has been used to capture the traffic in the network. Tcpdump is a packet analyser that runs under the command line. It allows the user to intercept and display TCP/IP and other packets being transmitted or received over a network to which the computer is attached. Distributed under the BSD license, tcpdump is free software. With Tcpdump in the “Analysis tools” machine, all the traffic has been captured and stored in a .cap file.

Wireshark [34] has been used to analyse the dumps stored in the .cap files and generated with tcpdump. Wireshark is a packet analyser used for network troubleshooting, analysis, software and communications protocol development, and education. Wireshark is very similar to tcpdump, but has a graphical front-end, plus some integrated sorting and filtering options. It also allows generating statistics information about the traffic captured.

CHAPTER 4: EXPERIMENTAL TESTS RESULTS

4.1. Introduction

In order to evaluate and compare the peer-to-peer algorithms some tests have been performed. These tests intend to compare the algorithms in terms of fairness, resilience to strategic peers and convergence time. In this chapter the results of the experimental tests are presented. In order to obtain very easy to see conclusions, the results are presented in charts.

4.2. Fairness Evaluation

The aim of the first experiment is to compare the different algorithms in terms of fairness. For a one peer, we could define fairness as the relation between the quantity of information uploaded and downloaded [14]. As much as this relation approaches to one, the protocol is fairer. Azureus (Standard BitTorrent implementation) (section 2.3.1.1), BitTyrant (section 2.3.1.3), FairTorrent (section 2.3.1.6) and PropShare (section 2.3.1.4) have been evaluated in this first experiment.

In order to compare the fairness, a laboratory for each studied protocol has been set up. Each one is comprised by a set of machines located in the same network with different roles:

- 2 peers with the whole file acting as seeders
- 8 peers without the file acting as leechers
- 1 machine with tracker role
- 1 machine where the analysis tools runs

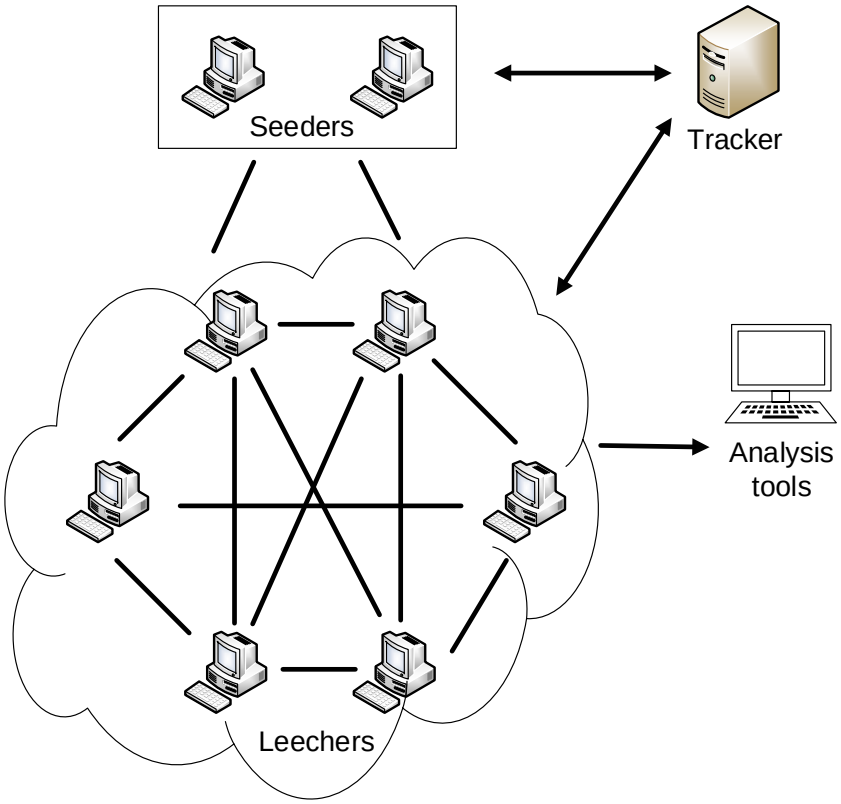

Figure 4.7: Fairness evaluation scenario

The seeders have a bandwidth capacity of 100KiB/s of uploading. Each leecher has a download capacity of 100KiB/s and the upload capacity is uniformly distributed between 10KiB/s and 60 KiB/s. These bandwidth rates reflect a typical scenario of users with asymmetric Internet connections with a bigger download capacity than upload capacity.

In this laboratory, leechers begin downloading simultaneously and remain as seeds in the system upon download completions. For each leecher, the average uploading rate has been compared with the average downloading rate from others leechers. This relation is presented below in very easy to see charts. In order to obtain the minimum sample results, the experiment has been run several times. The bandwidth usage measurements have been done using tcpdump and Wireshark.

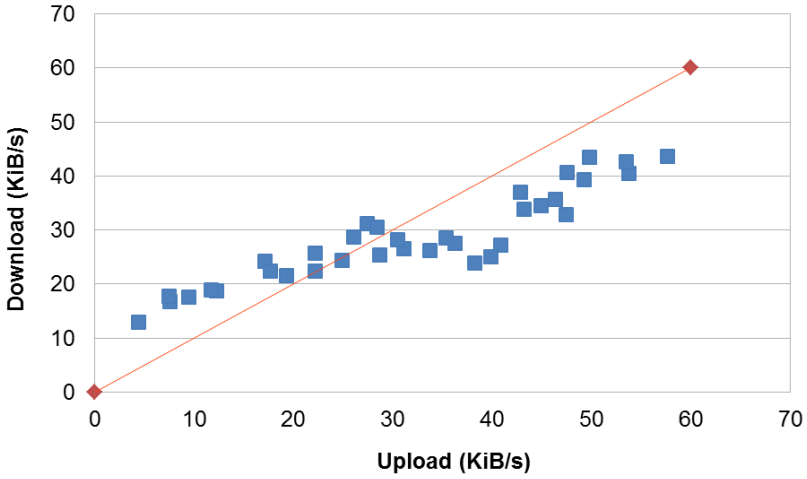

Figure 4.8: Azureus fairness evaluation chart

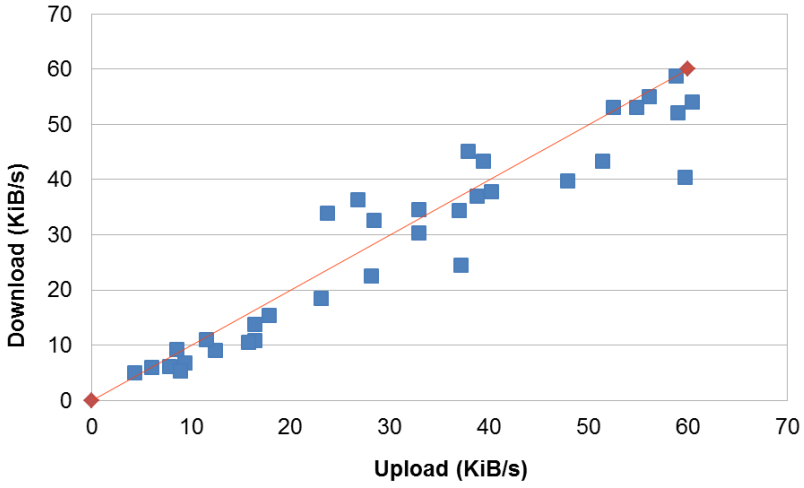

Figure 4.9: BitTyrant fairness evaluation chart

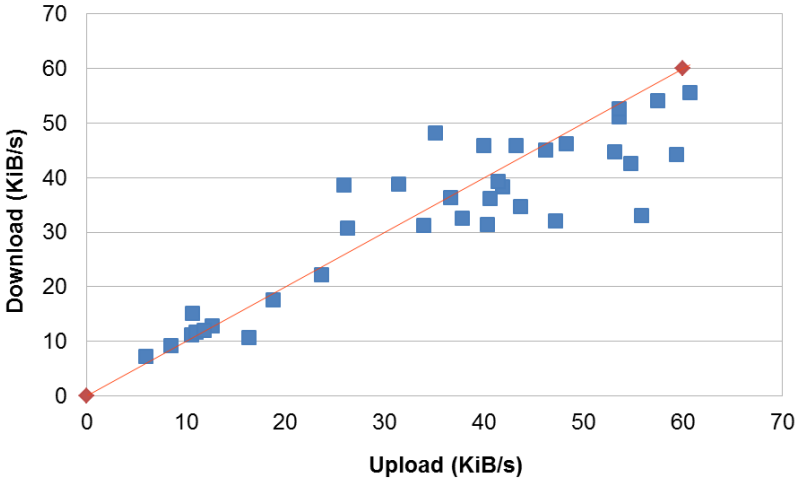

Figure 4.10: PropShare fairness evaluation chart

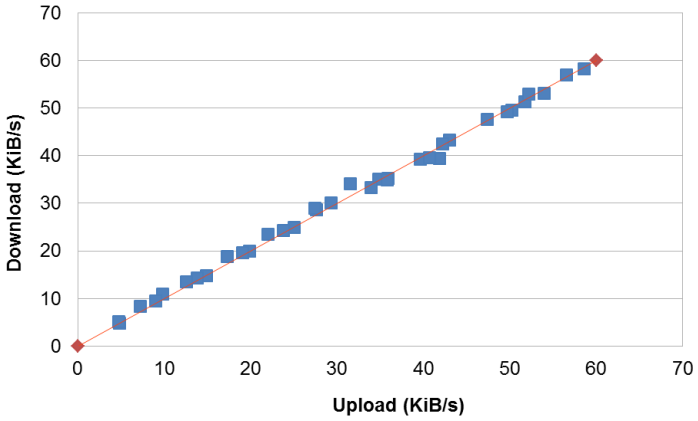

Figure 4.11: FairTorrent fairness evaluation chart

As defined before, fairness is the relation between the quantity of information uploaded and downloaded. As this relation is closest to one, we can say that is fairer. So, in the charts: the more close to the line y=x the results are, the fairer is the algorithm.

In the charts the red line is the line y=x so it is the ideal. The points marked in blue are the results of the different measurements done.

In terms of fairness, another interesting point would be to check with which other peers a peer collaborates, and to what extent is he collaborating. In order to be able to do so, a series of matrices has been built. These matrices show the total number of bytes uploaded by peers to each other, averaged over all runs. Darker squares represent more data.

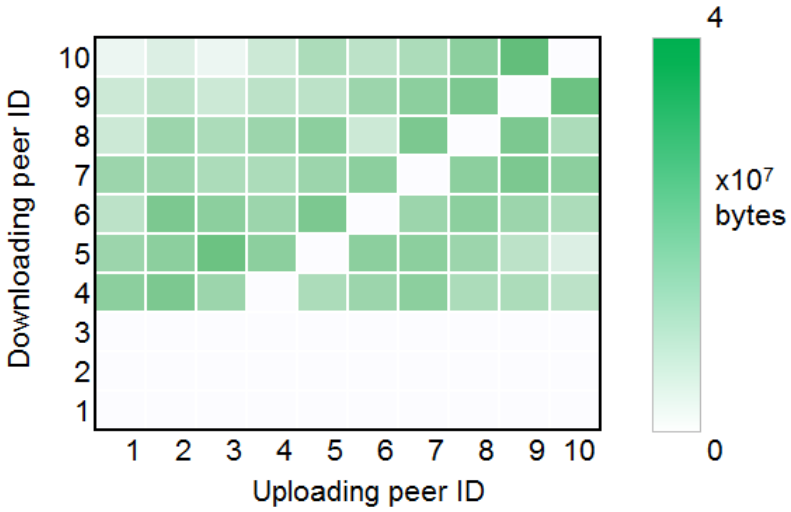

Figure 4.12: Azureus fairness evaluation matrix

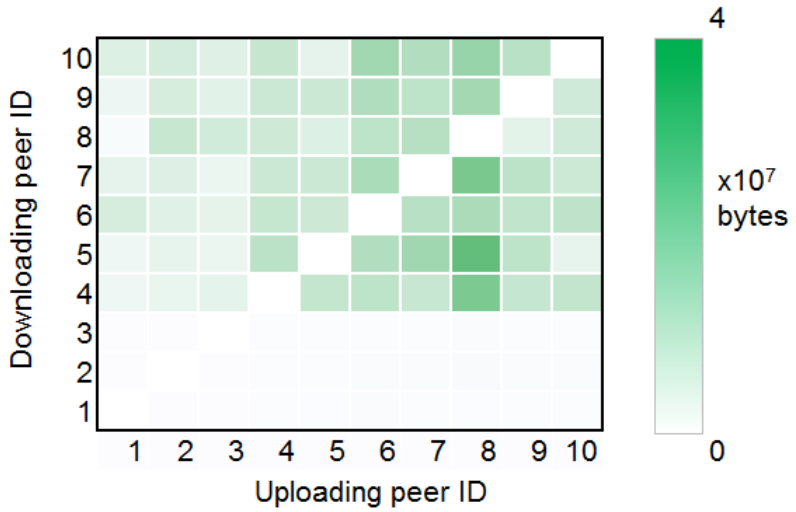

Figure 4.13: BitTyrant fairness evaluation matrix

Figure 4.14: PropShare fairness evaluation matrix

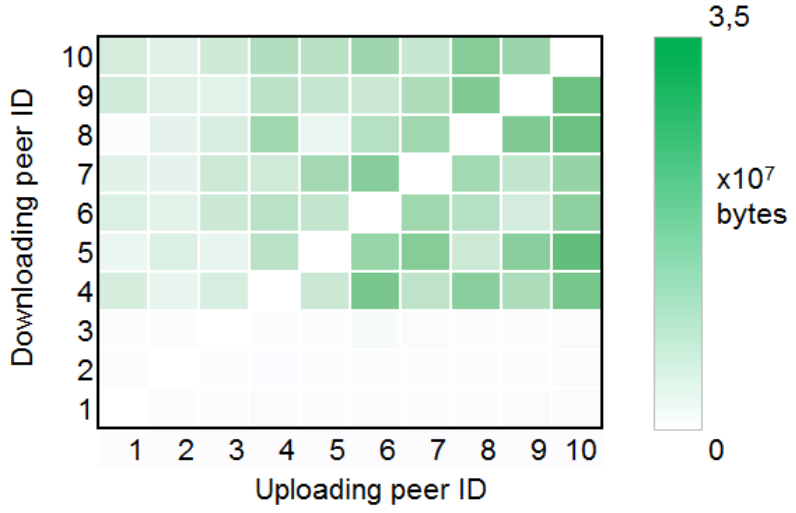

Figure 4.15: FairTorrent fairness evaluation matrix

Thanks to the resulting charts and matrix, is easy to get some interesting conclusions. We can say that the FairTorrent is the algorithm fairer as it has almost all the points over the ideal line. In the case of Azureus, it's showed that the algorithm benefits those that share less in detriment of those who share more.

4.3. Resilience to strategic peers

The aim is to compare the response of the different algorithms against a free-rider user that uses a strategic client. The goal of this testbed is to compare the resilience of the Azureus (section 2.3.1.1), FairTorrent (section 2.3.1.6) and PropShare (section 2.3.1.4) clients.

In order to do so, a scenario similar to the fairness experiment is used. This scenario is comprised of a swarm in which all nodes are sharing a file. All nodes use the same peer-to-peer algorithm except a node in which the strategic client BitTyrant is used. The download times for free-riders users with strategic node are compared with the download times of the rest of peers. Thereby it can be seen how the algorithm performs in a swarm in which free-riders nodes are present.

In order to compare the resilience, a laboratory for each studied protocol has been set up. Each one is comprised by a set of machines located in the same network with different roles:

- 2 peers with the whole file acting as seeders
- 7 peers without the file acting as leechers
- 1 peer acting as leecher running the BitTyrant client
- 1 machine with tracker role
- 1 machine where the analysis tools run

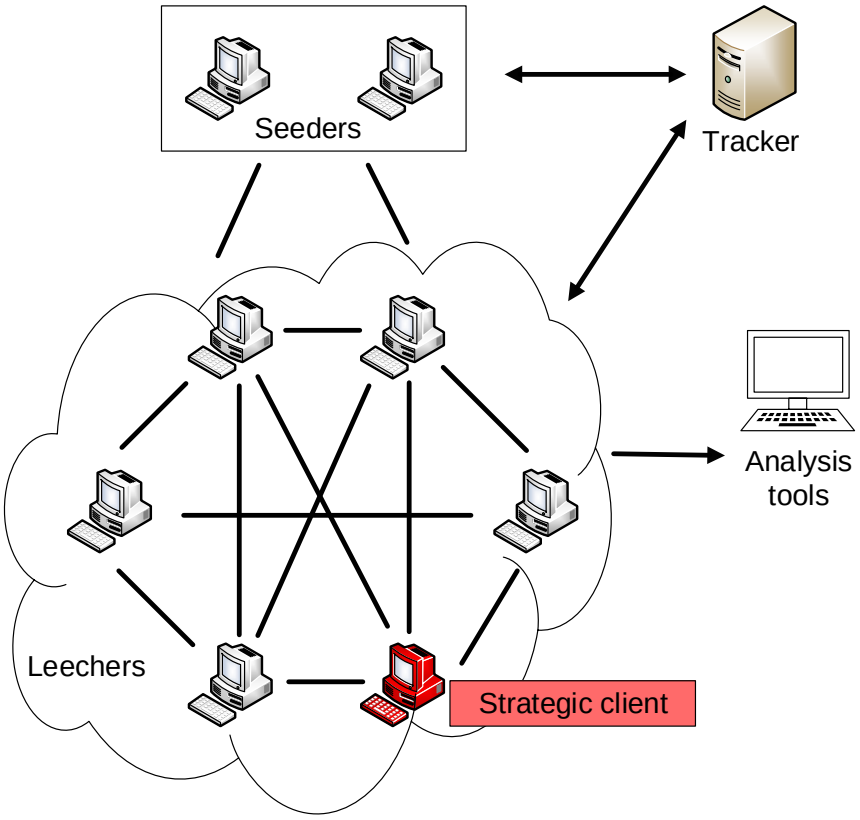

Figure 4.16: Resilience evaluation architecture

The seeders have a bandwidth capacity of 100KiB/s of upload. Each leecher have a download capacity of 100KiB/s and a upload capacity of 20 KiB/s. The strategic client has the same network capacities than the other leechers. This bandwidth rates reflects a typical scenario of users with asymmetric Internet connections with a bigger download capacity than upload capacity.

In this scenario, leechers begin downloading simultaneously and remain as seeds in the system upon download completions. The experiment has been run several times and for each leecher, the download time has been computed using tcpdump and Wireshark.

The download times are presented below in a very easy to see chart. This chart shows for each experiment the average and the range of the download times for both the strategic client and the algorithm tested.

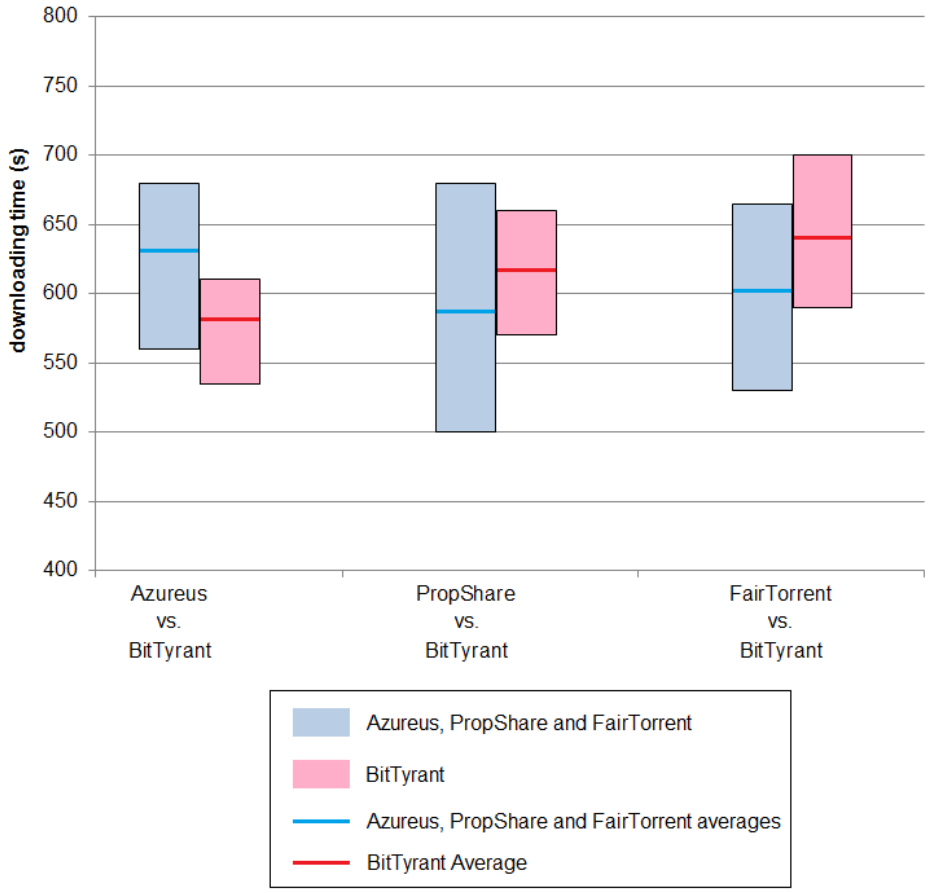

Figure 4.17: Average and range of the download times of BitTorrent and other algorithms competing

Another interesting point of view are the sharing data matrices. These matrices show the total number of bytes peers uploaded to each other, averaged over all runs. Darker squares represent more data. Peers 1 to 3 are seeds, peer 4 is the strategic peer and the other peers are leechers.

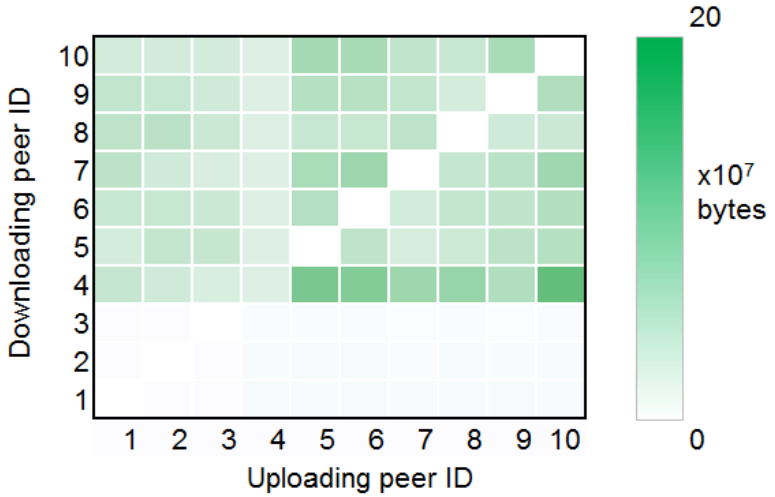

Figure 4.18: Azureus resilience evaluation matrix

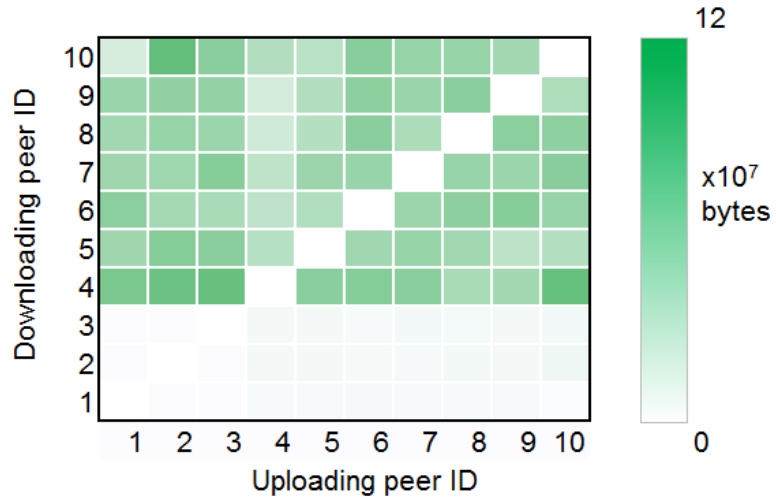

Figure 4.19: PropShare resilience evaluation matrix

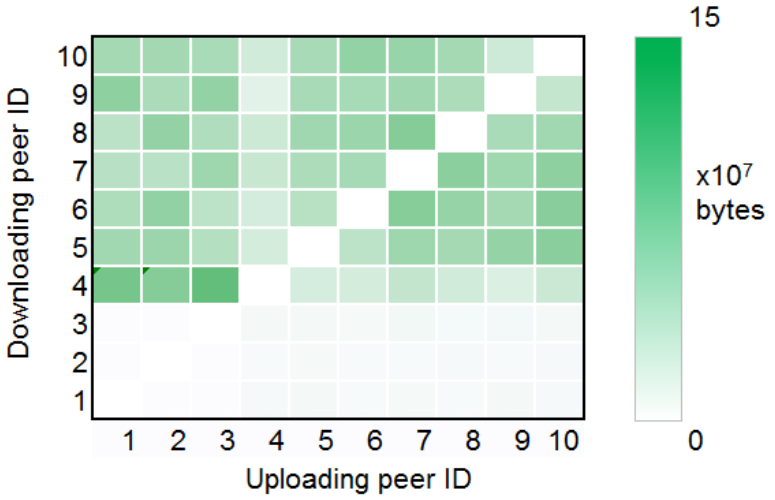

Figure 4.20: FairTorrent resilience evaluation matrix

Although there is not a huge difference, because the number of peers in the swarm is low, the chart shows clearly different behaviours among the three algorithms compared here. In the first case, the BitTyrant peer achieves better download times than the other users who are using Azureus. Furthermore, the Azureus sharing matrix shows how the peer 4 obtains more data from other leechers than the other peers do. This matrix also shows how this peer contributes with less information to the other peers. In this case we can say that Azureus is vulnerable to the free-rider strategies of the bitTyrant client. These results are in line with the conclusions of the paper made by the authors of BitTyrant as discussed in chapter 2 of this report, in the BitTyrant section.

On the other hand, in the PropShare and FairTorrent scenarios, the results are different. Both of them achieve better download times than BitTyrant. The PropShare sharing matrix shows that the strategic peer 4 has been penalised by the other leechers and therefore it has needed to obtain more data from the seeds. In the FairTorrent sharing matrix, this behaviour is more prominent. In both cases, PropShare and FairTorrent scenarios, the strategic peer BitTyrant has being punished by the algorithm on the other leechers because of his free-rider behaviour. These results are also in line with the respective conclusions from chapter 2 of this report.

4.4. Convergence time of each algorithm

As we have seen before in the direct-reciprocity section on chapter 2, the Nash equilibrium is a point in the time where all the peers already know each other's strategies and no one has anything to gain if they only change their own strategy unilaterally. In the Nash equilibrium the best result for each peer considered individually is achieved.

The goal of this testbed is to compare among the algorithms the time needed by the algorithm to achieve the equilibrium. We are looking for the convergence time of the algorithm. Azureus (Standard BitTorrent implementation) (section 2.3.1.1), BitTyrant (section 2.3.1.3), FairTorrent (section 2.3.1.6) and PropShare (section 2.3.1.4) have been evaluated in this experiment.

In order to do so, each algorithm has been executed in a testbed several times. For each peer in the swarm, we will consider that the peer has arrived to the equilibrium when it reaches a download rate of at least 90% of his maximum upload rate configured. Therefore, we will consider the convergence time as the time between the moment when the node first contacts the tracker and when the node arrives to the equilibrium.

The scenario built to compare the converge time of the different algorithms is comprised by a set of machines located in the same network:

- 3 peers with the whole file acting as seeders
- 8 peers without the file acting as leechers
- 1 machine with tracker role
- 1 machine where the analysis tools runs

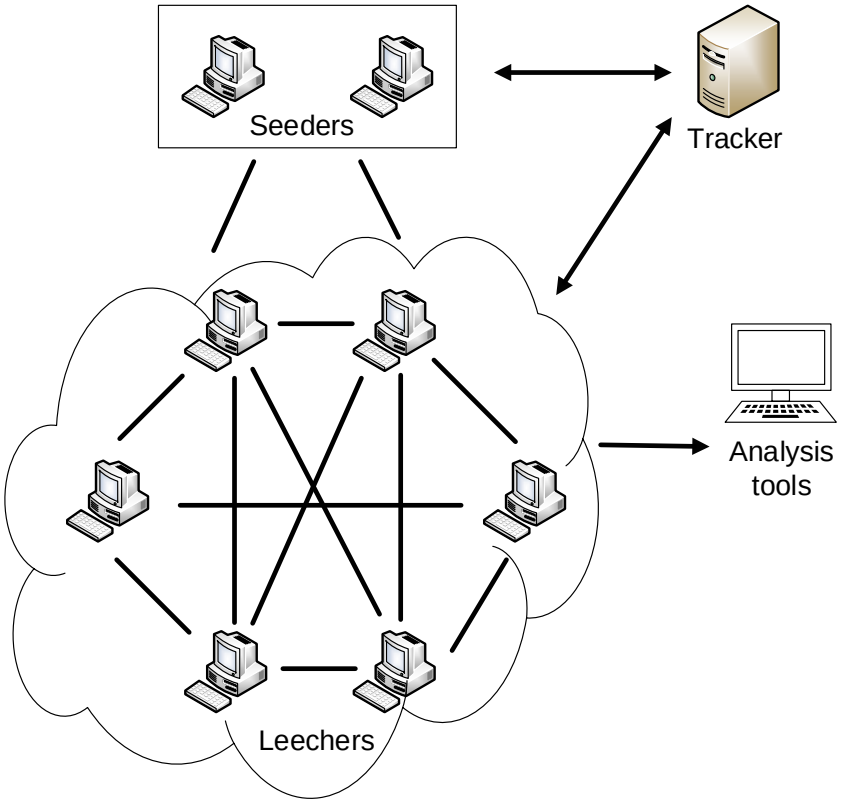

Figure 4.21: Convergence time evaluation scenario

The seeders have a bandwidth capacity of 100 KiB/s of uploading. Each leecher has a download capacity of 100 KiB/s and the upload capacity is uniformly distributed between 10 KiB/s and 60 KiB/s. These bandwidth rates reflect a typical scenario of users with asymmetric Internet connections with a bigger download capacity than upload capacity.

In this laboratory, leechers begin downloading simultaneously and remain as seeds in the system upon download completions. In order to obtain the minimum sample results, the experiment has been run several times. The bandwidth usage measurements have been done using tcpdump and Wireshark.

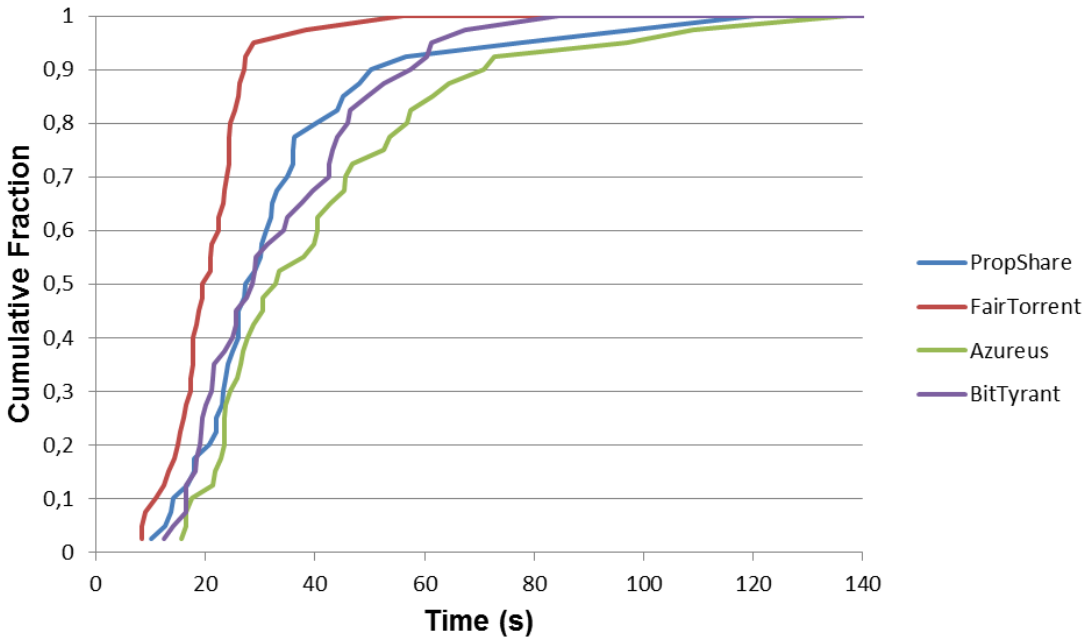

Figure 4.22: Figure Rate convergence

As defined before, the convergence time is the time that a peer needs to reach a downloading rate at least 90% of its maximum download rate configured.
The figure shows the cumulative frequency analysis of the time needed by the peers to obtain the equilibrium.

Thanks to the resulting charts, we can conclude that FairTorrent is the algorithm which needs less time to converge among the algorithms compared. The differences showed in this chart are not too large. That is because the swarms are very small due to the limitations imposed by the testbed infrastructure in which we cannot run testbeds with a very large number of peers due to performance issues. In real swarms where the number of peers are ten times larger than the used here, the differences between the algorithms would be larger.

CHAPTER 5: CONCLUSIONS

As they are presented in the introduction of this report, there were two main goals to be accomplished in this thesis. The first one was to analyse the state of the art related to the different existing algorithms detailing their characteristics and profits according to their authors, and provide a classification by characteristics of the algorithms.

The second important goal was to build an scenario where different peer-to-peer algorithms could be experimentally tested and compared. Those scenarios were supposed to be capable to obtain experimental results allowing to compare several algorithms among others and confirm the results obtained in the theoretic comparative. The experimental results had to allow verify or deny the presumed profits provided by each algorithm.

At this point we can affirm that all the goals presented at the beginning of the thesis have been accomplished throughout the chapters. A strict analysis of the state of the art has been provided. The different algorithms have been classified taking into consideration the methods to inventive the peers.

After the theoretical analysis, a functional scenario to test some algorithms has been built. The scenario built has allowed to test different BitTorrent clients implementing different incentives algorithms in order to be able to confirm or deny the most important features. The presented scenario is not a very realistic swarm, since it is comprised by very few nodes. However, the tests are real clients running in a virtualization environment and not only simulations, what means that the results obtained are not perfectly strict, but the results that we would obtain performing the same test in the real world swarms will be in the same line that the ones obtained here.

In the experiments, several algorithms of the type Direct-reciprocity have been tested under a BitTorrent network. Those algorithms are the most interesting type due to their simple way of working and thus their interoperability with other clients. In the experiments, fairness, resilience to strategic peers and convergence time has been evaluated. These test results provide a good vision of the incentive algorithms quality and allow to compare the algorithms between them.

Both in the study of the literature and in the experimental results, FairTorrent is the algorithm with better results: compared to other peer-to-peer systems, FairTorrent's deficit-based algorithm provides a high degree of fairness, a better performance and a fast convergence rate. It was also seen that FairTorrent is resilient to free-riders, low contributors and strategic peers. FairTorrent does not require a centralized system, peer reputation, or third-party credit-keeping services, making it very simple.

Both, the comprehensive study of the state of the art and the experimental tests, have allowed us to identify not only real but also supposed benefits, which ended up not being as important as their creators considered. The results

obtained in the experimental part have been in line with the results of the literature analysis.

5.1. Environmental impact

This thesis is related to improvements in the peer-to-peer networks by improving its incentives algorithms. The environmental impact of this changes in the algorithms by itself is minimum.

Despite the foregoing, the impact of the infrastructures bellow the peer-to-peer networks is large. Peer-to-peer networks usually run over Internet. Here we have to consider all the hardware across the world to support Internet. This large quantity of hardware has an important impact in terms of energy consumption. Servers need energy to be build, they need energy to run and they need energy to be refrigerated inside the datacentres, among other energy consumption throughout its life. Therefore the carbon footprint related to this energy consumption is high. Some studies indicate that in 2012, between 2% and 3% of total world's emissions were due to Internet [25].

The fraction of the total Internet bandwidth used by peer-to-peer networks is so large. Besides that, the use of peer-to-peer traffic is growing dramatically, particularly for sharing large video/audio files and software. This is because of the great success of the peer-to-peer networks on Internet.

The success of the peer-to-peer networks is because they work well. That means that improving this networks incentivize people to use more and more these networks. This implies more bandwidth consumption and therefore more hardware needed consuming energy.

Related to the environmental impact, we can conclude that maybe at first sight to improve an algorithm has a minimum impact. But an analysis a little bit far shows us that a little change in an algorithm can have a large environmental impact. But not all is bad, we also should notice here that the use of data networks minimizes the impact the technology it replaces, some of them with a larger footprint.

5.2. Future work

To carry out experimental tests, a virtualization platform called Net-kit has been used. In the future, those tests could be extrapolated into a real platform of simulations of peer-to-peer and even data could be extracted from peer-to-peer swarms. This way, it could be possible to carry out a much more strict experimental evaluation.

In terms of the development of new algorithms, we can continue researching on the presented methods of incentives. Algorithms can be improved so that they

improve on performance, fairness, security and computational cost. It is also possible to keep investigating in combining different methods at the same time and see if those combinations can bring even more improvements. Finally, we can also combine incentive methods with other methods to different purposes such as peer authentication, to check if combining them would bring benefits.

In the more generic field of peer-to-peer, it is necessary to continue working on improving the security of the system, preventing the spread of viruses and malware and avoiding the bad content identification within the network. Another field with a lot of possible improvement is the search of content algorithms within the peer-to-peer network.

5.3. Personal approach

The execution of this thesis has provided me with a lot of knowledge in different fields and technologies. First of all, the initial study made me acquire knowledge about the peer-to-peer technology and the different algorithms of incentives.

Apart from the knowledge related to the peer-to-peer technology, the experiments gave me new knowledge of a range of fields like virtualisation technologies for Linux, the Linux operative system, the tools for traffic analysis, etc. During the execution of the experimental tests several problems appeared, whose identification and resolution where crucial during this learning.

The use of Net-kit with the Linux User-Mode technology has allowed me to build up a test scenario that could be repeated as many times as it was needed, thanks to the management layer of automatization of laboratories included with Net-kit. But Net-kit has also some drawbacks. Netkit does not allow running applications with graphical user interfaces. Such a limitation prevents from using peer-to-peer clients in which the graphical user interface is needed for running. Another point has been the difficulties encountered to build the scenario. Net-kit has some problems related to working with those applications that have a high use of memory like the Java virtual Machine. That has affected during the filesystem building phase and the running of the virtual machines. Some fine tuning configuration related to this issue was performed to overcome those problems. These configuration parameters, detailed in the ANNEX 2, were necessary to be able to build the test scenario.

TC is a very powerful tool to control traffic. This tool, combined with the simplicity of use provided by Wondershaper, has been very useful and effective to limit the bandwidth of the peers.

The powerful tool Wireshark in combination with tcpdump has been very useful to obtain a detailed view of the network traffic. Wireshark includes a lot of appropriate tools to extract stats to generate the experimental results.

Apart from the knowledge related to technologies, the execution of this thesis has provided me with a lot of new knowledge regarding to methodologies, processes, work organization and a long etcetera.

BIBLIOGRAPHY

[1] A. Sherman, J. Nieh and C. Stein. FairTorrent bringing fairness to peer-to-peer systems. Proceedings of the 5th ACM Conference on the emerging Networking Experiments and Technologies (CoNEXT 2009), Rome, Italy, December 1-4, 2009.

[2] M. Sirivianos, J. H. Park, R. Chen, and X. Yang. Free-riding in BitTorrent with the large view exploit. In 6th International Workshop on Peer-to-Peer Systems (IPTPS ’07), Bellevue, WA, Feb. 2007.

[3] S. Jun and M. Ahamad. Incentives in BitTorrent Induce Free-riding. In Proceedings of the 3nd Workshop on the Economics of Peer-to-Peer Systems, Aug. 2005.

[4] K. Ranganathan, M. Ripeanu, A. Sarin and I. Foster. To Share or Not to Share, An Analysis of Incentives to Contribute in Collaborative File Sharing Environments. In Workshop on Economics of Peer-to-Peer Systems (June 2003).

[5] K. Lai, M. Feldman, I. Stoica, and J. Chuang. Incentives for cooperation in peer-to-peer networks. In Workshop on Economics of Peer-to-Peer Systems, 2003.

[7] M. Sirivianos, J. H. Park, R. Chen, and X. Yang. Free Riding in BitTorrent Networks with the Large View Exploit. In Proceedings of the 6th International Workshop on Peer-to-Peer Systems (IPTPS ’07), Feb. 2007.

[8] D. Levin, K. LaCurts, N. Spring, and B. Bhattacharjee. BitTorrent is an Auction: Analyzing and Improving BitTorrent’s Incentives. In Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, (SIGCOMM ’08), Aug. 2008.

[10] Z. Liu, P. Dhungel, Di Wu, C. Zhang, and K.W. Ross, “Understanding and Improving Incentives in Private P2P Communities”, ICDCS, Genoa, Italy, 2010.

[11] F. Wu and L. Zhang. Proportional Response Dynamics Leads to Market Equilibrium. In Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC ’07), June 2007.

[12] B. Cohen. Incentives Build Robustness in BitTorrent. In Proceedings of the 1st Workshop on the Economics of Peer-to-Peer Systems, June 2003.

[13] M. Sirivianos, X. Yang, and S. Jarecski. Dandelion: Cooperative Content Distribution with Robust Incentives. In Proceedings of the 2007 USENIX Annual Technical Conference (USENIX ’07), June 2007.

[14] D. Levin, R. Sherwood, and B. Bhattacharjee. Fair file swarming with FOX. In IPTPS, 2006.

[15] K. Shin, D. Reeves, I. Rhee, "Treat-Before-Trick : Free-riding Prevention for BitTorrent-like Peer-to-Peer Networks", Proc. Of Intl. Parallel and Distributed Processing Symposium (IPDPS 2009), May 2009.

[16] D. Qiu and R.Srikant. Modeling and Performance Analysis of BitTorrent-Like Peet-to-Peer Networks. In Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, (SIGCOMM ’04), Sept. 2004.

[17] S. Kamvar, M. Schlosser, and H. Garcia-Molina. The EigenTrust Algorithm for Reputation Management in P2P Networks. In Proceedings of the 12th International World Wide Web Conference (WWW’03), May 2003.

[18] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani. Do Incentives Build Robustness in BitTorrent. In Proceedings of the 4th USENIX Symposium on Networked Systems Design and Implementation (NSDI ’07), Apr. 2007.

[19] M. Feldman and J. Chuang, “Overcoming free-riding behavior in peer-to-peer systems,” in ACM Sigecom Exchanges, vol. 5, July 2005.

[20] S. J. Nielson, C. E. Spare and D. S. Wallach: Building Better Incentives for Robustness in BitTorrent. CoRR abs/1108.2716, 2011.

[21] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In Proceedings of Multimedia Computing and Networking (MMCN) 2002, January 2002.

[22] M. Feldman, K. Lai, I. Stoica, and J. Chuang, “Robust incentive techniques for peer-to-peer networks,” in EC, 2004.

[23] Feldman, M., Papadimitriou, C., Stoica, I., and Chuang, J.: Free-Riding and White-washing in Peer-to-Peer Systems. In Proc. SIGCOMM workshop on Practice and Theory of Incentives and Game Theory in Networked Systems, 2004.

[24] Golle, P., Leyton-Brown, K., Mironov, I., and Lillibridge, M.: Incentives For Sharing in Peer-to-Peer Networks. In Proceedings of the 3rd ACM conference on Electronic Commerce, October 2001.

[25] CEET Annual report 2012. Centre for Energy-Efficient Telecommunications (CEET), Friday 31 August 2012.

[26] R. Schollmeier, A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications, Proceedings of the First International Conference on Peer-to-Peer Computing, IEEE, 2002.

[27] Ralf Steinmetz, Klaus Wehrle. Peer-to-Peer Systems and Applications, Springer-Verlag Berlin Heidelberg, 2005.

[29] Netkit, Computer Networks Laboratory, Università degli studi Roma Tre: http://wiki.netkit.org

[28] eMule. http://www.emule-project.net/.

[29] Azureus. http://www.azureus.com/.

[30] Kazaa. http://www.kazaa.com/.

[31] Linux Advanced Routing & Traffic Control. http://http://www.lartc.org/.

[32] The Wonder Shaper. http://lartc.org/wondershaper/.

[33] Tcpdump. http://www.tcpdump.org/

[34] WireShark. http://www.wireshark.org/

ANNEX 1: NETKIT SET-UP

Starting from Netkit version 2, consists of three different packages:

1. The Netkit "core", which contains commands, documentation and other stuff which is necessary for Netkit to work.
2. The Netkit filesystem, which contains the filesystem for virtual machines.
3. The Netkit kernel, which contains the kernel used by virtual machines.

Netkit is installed in 3 steps:

- Step 1: Download and unpack

  Download all the files to a directory of your choice. Then unpack them by using the following commands:

```
#tar -xjSf netkit-x.y.tar.bz2
#tar -xjSf netkit-filesystem-Fx.y.tar.bz2
#tar -xjSf netkit-kernel-Kx.y.tar.bz2
```

  Once Netkit has been unpacked, no root privileges are required to configure it and start working.

- Step 2: Configuration

  The first step is to set the environment variable NETKIT_HOME to the name of the directory Netkit has been installed into. In order to access the Netkit man pages, the MANPATH variable must be set to ":$NETKIT_HOME/man". For example, assuming that you have installed Netkit to /home/foo/netkit and that your shell is bash, you would use the following commands:

```
#export NETKIT_HOME=/home/foo/netkit
#export MANPATH=:$NETKIT_HOME/man
```

  It may also be useful to put these lines inside your shell initialization file (`.bashrc' in case you are using the bash shell).

  After doing this, you need to update your PATH environment variable to include the path to the standard Netkit commands. This is required in order to make Netkit work properly. The entry you need to add to the PATH is "$NETKIT_HOME/bin". For example, assuming Netkit is (still) installed into /home/foo/netkit and that your shell is (still) bash, you would type:

```
#export PATH=$NETKIT_HOME/bin:$PATH
```

Again, it may be convenient to put this line inside your shell initialization file.

- Step 3: Checking the configuration

  At this point, change the current directory to the Netkit directory:

```
#cd netkit
```

  Now, run the `check_configuration.sh' script by typing:

```
#./check_configuration.sh
```

  This script takes care of checking whether your system is configured properly to make Netkit run. Any misconfigurations are signalled and instructions for fixing them are reported as well. If the script exits with success, then Netkit is ready for use.

After the installation of the Netkit, in order to test whether Netkit is working properly, you can start a simple virtual machine by issuing the command:

```
#vstart pc1
```

If everything is in place, you should see a new virtual machine starting up (eventually popping up an Xterm window) and the command `vlist' on the host machine should show an output similar to the following:

```
#vlist

USER  VHOST     PID   UPTIME       SIZE  INTERFACES
foo     pc1   24102   00:03       12376

Total virtual machines:    1    (you),     1    (all users)
Total consumed memory: 12376 KB (you), 12376 KB (all users)
```

You can stop the virtual machine by typing the following command on the host machine console:

```
#vhalt -r pc1
```

You can now delete the file pc1.log.

As an additional feature, users of the bash shell can take advantage of command line auto completion for Netkit commands. In order to activate it, first of all make sure your shell is bash:

```
#readlink -f $SHELL
```

If it is, then you can safely add the following line at the end of your ~/.bashrc file:

```
#. $NETKIT_HOME/bin/netkit_bash_completion
```

ANNEX 2: MODIFYING THE FILESYSTEM IN NETKIT

The Netkit filesystem is an image of an installed Debian GNU/Linux distribution including several packages that can profitably be used within a network emulation.

A virtual machine filesystem is a special file on the host machine. There exists a single model filesystem that is shared by all the virtual machines and provides the full suite of tools.

To prepare the different laboratory scenarios, the filesystem provided in Netkit has been modified in order to include all required packages. Below are detailed all the modifications done in the standard filesystem provided by Netkit its version 5.2.

To modify the filesystem, a virtual machine booted with the option -- no-cow has been used. A virtual machine over Netkit uses a base filesystem which is not modified during the execution and another file for each virtual machine in which all the file modifications are being done. The --no-cow option allow being able to modify directly the base filesystem. These modifications will affect all the virtual machines booted in the future.

In order to have access to Internet, a tunnel to the host machine with the option --eth0=tap,X,X has been used.

In order that some commands work properly, the virtual machine memory has to be extended to 512 MBytes. The option used to extend the virtual memory for a virtual machine is --mem=512.

Finally, the command to boot a machine to modify the filesystem has been the one showed below:

```
#echo "nameserver 8.8.8.8" >> /etc/resolv.conf
```

After that, the apt packages have to be updated and it is required to add the key in order to authenticate the source of the packages:

```
#apt-get update
#gpg --keyserver pgpkeys.mit.edu --recv-key
AED4B06F473041FA
#gpg -a --export AED4B06F473041FA | apt-key add -
#apt-get update
```

The Java Virtual Machine and the Ant packages are required for run the Azureus Client:

```
#apt-get install openjdk-6-jdk
#apt-get install ant
```

In order to avoid some errors, the java heap space has been increased:

```
#export ANT_OPTS="-Xmx512M"
#apt-get install zip
```

In order to limit the bandwidth and simulate a real network environment between the servers, Wondershaper is used. Wondershaper is a script to limit the ethernet connection bandwidth that uses the Linux tc command and makes it easier to use.

```
#apt-get install wondershaper
#ln -s /sbin/orig-tc /sbin/tc
```

Clients installation in the Netkit filesystem:

The first client installed was FairTorrent which is based on Azureus and it also includes the possibility to run the standard BitTorrent algorithm using the standard Azureus implementation.

To install the FairTorrent client is has to be downloaded from the official webpage. After the download, it has been checked the md5 hash and then decompressed:

```
#wget
http://www.cs.columbia.edu/~asherman/fairtorrent/downloads/FairTorrent_1.1.1.tgz
#tar -zxvf FairTorrent_1.1.1.tgz
#ant
```

To install the BitTyrant client, it has to be downloaded from the official webpage. After the download, it has been checked the md5 hash and then decompressed:

```
#wget
http://coblitz.codeen.org:3125/bittyrant.cs.washington.edu/
#dist_010807/BitTyrant-src.zip
#apt-get install unzip
#unzip BitTyrant-src.zip
#ant
```

Bitthieft

```
#wget http://bitthief.ethz.ch/dist/linux/BitThief.tgz
#tar -zxvf BitThief.tgz
```

PropShare

```
#wget
```

```
#http://www.cs.umd.edu/projects/propshare/propshare_src.tar.gz
#tar -zxvf propshare_src.tar.gz
#ant
```

Tracker Installation in the Netkit filesystem:

Finally a bitTorrent tracker has been installed. A tracker provides with remote peers to all the clients of the swarm. The tracker chosen was bittornado due to its simplicity and its console mode.

To install bitTornado:

```
#apt-get install bittornado
```

ANNEX 3: NETKIT LABORATORIES CONFIGURATIONS

Netkit with its labs feature, allows to build scenarios and to re-run the experiments as many times as needed. A Netkit lab is a set of preconfigured virtual machines that can be started and halted together. This annex contains an explanation about how a Netkit lab is built and the specific configurations of the scenarios used in each experiment.

Netkit allows to build scenarios in two ways:

- As a single script using the commands to execute virtual Machines
- By using laboratory configuration files.

In this case the second option has been used. In this option, Netkit requires a directory tree with the above elements:

- a lab.conf file describing the network topology.
- a set of subdirectories that contain the configuration settings for each virtual Machine.
- .startup and .shutdown files that describe actions performed by virtual machines when they are started or halted.
- [optional] a lab.dep file describing dependency relationships on the startup order of virtual Machines.
- [optional] a _test directory containing scripts for testing that the lab is working correctly.

When a laboratory directory tree is created, it can be easily booted by the command lstart.

Some examples of this configuration files used in the experiments are included above:

- Lab.conf file: In this file, several machines connected to the same virtual hub “A” have been defined. The mem parameter is also used in order to create the virtual machines with more memory than the standard. This is because using the standard amount of memory, some pieces of software produced errors due to a lack of memory resources. Finally, it also includes some descriptive information about the laboratory:

```
LAB_DESCRIPTION="Fairness LAB"
LAB_VERSION=1
LAB_AUTHOR="D. Serra"

pc1[0]="A"
pc2[0]="A"
pc3[0]="A"
pc4[0]="A"
pc5[0]="A"
pc6[0]="A"
pc7[0]="A"
pc8[0]="A"
```

```
pc9[0]="A"
pc10[0]="A"
pc11[0]="A"
pc12[0]="A"
pc13[0]="A"
pc14[0]="A"
pc15[0]="A"
pc100[0]="A"
pc101[0]="A"

pc1[mem]=512
pc2[mem]=512
pc3[mem]=512
pc4[mem]=512
pc5[mem]=512
pc6[mem]=512
pc7[mem]=512
pc8[mem]=512
pc9[mem]=512
pc10[mem]=512
pc11[mem]=512
pc12[mem]=512
pc13[mem]=512
pc14[mem]=512
pc15[mem]=512
pc100[mem]=512
pc101[mem]=512
```

- startup.conf file: This file defines the commands to be executed after the virtual machine booting. Here are some examples depending on the machine role:

    - Tracker: The first thing to do is to configure the network interface by ifconfig command. After that, it executes the tracker with the proper parameters to listen in the 6969 standard BitTorrent protocol among others:

```
ifconfig eth0 10.0.0.100 netmask 255.255.255.0 broadcast 10.0.0.255 up

bttrack --port 6969 --dfile ~/.bttrack/dstate --logfile
~/.bttrack/tracker.log --nat_check 0 --scrape_allowed full &
```

    - Seed peer (in this case, using the FairTorrent client):
        - Using ifconfig command the network interface is configured.
        - With the mkdir command, a directory that will contain the data file to be shared is created.
        - With the command ln, a link to the data file source is created: /hosthome/10MBfile.txt. In Netkit virtual machine the /hosthome directory is where the /home/ directory of the host machine is mounted. That allows to access the host files from the virtual machine. A link is created in order to avoid to copy the whole file to the virtual machine.

- The torrent file containing the hash of the file to be downloaded has been copied from the host using the /hosthome/ directory by the cp command.
- Using wondershaper, the maximum upload and download rates have been established.
- Finally the peer-to-peer client has been executed with the torrent file as a parameter.

```
ifconfig eth0 10.0.0.1 netmask 255.255.255.0 broadcast 10.0.0.255 up

mkdir /root/FairTorrent/dist/my_dir/
ln -s /hosthome/10MBfile.txt /root/FairTorrent/dist/my_dir/10MBfile.txt
cp /hosthome/10MBfile.txt.torrent /root/FairTorrent/dist/my_dir/

wondershaper eth0 8200 100

cd /root/FairTorrent/dist/
java -jar /root/FairTorrent/dist/Azureus2.jar --ui=console
/root/FairTorrent/dist/my_dir/10MBfile.txt.torrent </dev/null >&/dev/null &
```

- Leecher peer (in this case, using the FairTorrent client): This file is the same as the seed one except by the command ln. A leecher peer does not have the data file to be shared at the beginning:

```
ifconfig eth0 10.0.0.10 netmask 255.255.255.0 broadcast 10.0.0.255 up

mkdir /root/FairTorrent/dist/my_dir/
cp /hosthome/10MBfile.txt.torrent /root/FairTorrent/dist/my_dir/

wondershaper eth0 8200 100

cd /root/FairTorrent/dist/

java -jar /root/FairTorrent/dist/Azureus2.jar --ui=console
/root/FairTorrent/dist/my_dir/10MBfile.txt.torrent </dev/null >&/dev/null &
```

- Analysis tool: This machine role is in charge to analyses all the communications during the sharing process. At the beginning, the network interface is configured using ifconfig. Then, the network capture is launched using tcpdump tool:

```
ifconfig eth0 10.0.0.101 netmask 255.255.255.0 broadcast 10.0.0.255 up

tcpdump -i eth0 -s 65535 -w /hosthome/lab25.cap
```