HAL
open science

Rarest First and Choke Algorithms Are Enough

Arnaud Legout, Guillaume Urvoy-Keller, Pietro Michiardi

To cite this version:

Arnaud Legout, Guillaume Urvoy-Keller, Pietro Michiardi. Rarest First and Choke Algorithms Are Enough. [Research Report] 2006. inria-00001111v1

HAL Id: inria-00001111

https://inria.hal.science/inria-00001111v1

Submitted on 13 Feb 2006 (v1), last revised 6 Sep 2006 (v3)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Rarest First and Choke Algorithms Are Enough

Arnaud Legout
I.N.R.I.A.
Sophia Antipolis, France
Email: arnaud.legout@sophia.inria.fr
Guillaume Urvoy-Keller and Pietro Michiardi
Institut Eurecom
Sophia Antipolis, France
Email: {Guillaume.Urvoy,Pietro.Michiardi}@eurecom.fr

Technical Report

***Abstract*— The performance of peer-to-peer file replication comes from its piece and peer selection strategies. Two such strategies have been introduced by the BitTorrent protocol: the rarest first and choke algorithms. Whereas it is commonly admitted that BitTorrent performs well, recent studies have proposed the replacement of the rarest first and choke algorithms in order to improve efficiency and fairness. In this paper, we advocate that the replacement of the rarest first and choke algorithms cannot be justified in the context of peer-to-peer file replication in the Internet based on real experiments.**

We instrumented a BitTorrent client and ran experiments on real torrents with different characteristics. Our experimental evaluation is peer oriented, instead of tracker oriented, which allows us to get detailed information on all exchanged messages and protocol events. We go beyond the mere observation of the good efficiency of both algorithms. We show that the rarest first algorithm guarantees a diversity of the pieces among peers close to the ideal one. In particular, on our experiments, a replacement of the rarest first algorithm with a source or network coding solution cannot be justified. We also show that the choke algorithm in its latest version fosters reciprocation and is robust to free riders. In particular, the choke algorithm is fair and its replacement with a bit level tit-for-tat solution is not appropriate. Finally, we identify new areas of improvements for efficient peer-to-peer file replication protocols.

I. Introduction

In a few years, peer-to-peer file sharing has become the most popular application in the Internet [16], [17]. The major reasons for this success are the efficient content localization and replication. Whereas content localization has raised a lot of attention in the last years [7], [12], [22], [24], content replication has started to be the subject of active research recently. As an illustration, the most popular peer-to-peer file sharing networks [1] eDonkey2K, FastTrack, Gnutella, Overnet focus on content localization. The only widely used [16], [17], [19] peer-to-peer file sharing application focusing on content replication is BitTorrent [8].

Yang et al. [25] studied the problem of efficient content replication in a peer-to-peer network. They showed that the capacity of the network to serve a content grows exponentially with time in the case of a flash crowd, and that a key improvement on peer-to-peer file replication is to split the content into several pieces. Qiu et al. [21] proposed a refined model of BitTorrent and showed its high efficiency. These studies show that a peer-to-peer architecture for file replication is a major improvement compared to a client server architecture, whose capacity of service does not scale with the number of peers.

However, both studies assume global knowledge, which is not realistic. Indeed, they assume that each peer knows all the other peers, and that each peer can always find an interesting piece of content on any other peer. As a consequence, the results obtained with this assumption can be considered as the optimal case. In real implementations, there is no global knowledge. The challenge is then to design a peer-to-peer protocol that achieves a level of efficiency close to the one achieved in the case of global knowledge.

The two keys of efficient peer-to-peer content replication are piece and peer selection strategies. Indeed, in a peer-to-peer system, the content is split into several pieces, and each peer acts as a client and a server. Therefore, each peer can receive and give any piece to any other peer. An efficient piece selection strategy should guarantee that each peer can always find an interesting piece on any other peer. The rationale is to offer the largest choice of peers to the peer selection strategy. An efficient peer selection strategy should maximize the capacity of service of the system. In particular, the peer selection should be based on efficiency criteria, e.g., upload and download capacity, and should not be biased by the lack of available pieces in some peers.

The rarest first algorithm is a piece selection strategy that consists in selecting the rarest pieces first. This simple strategy used by BitTorrent performs better than a random piece selection strategy [5], [9]. However, Gkantsidis et al. [11] argued using simulations that the rarest first algorithm may lead to the scarcity of some pieces of content and proposed a solution based on network coding. Whereas this solution is elegant and has raised a lot of attention, it leads to several complex deployment issues such as security or computational complexity. Other solutions based on source coding [18] have also been proposed to solve the claimed deficiencies of the rarest first algorithm.

The choke algorithm is the peer selection strategy of BitTorrent. This strategy is based on the reciprocation of upload and download speeds. Several studies [5], [10], [13], [15] discussed the fairness issues of the choke algorithm. In particular, they argued that the choke algorithm is unfair and favors free riders,

i.e., peers that do not contribute. Solutions based on a bit level tit-for-tat have been proposed to give a higher level of fairness than the choke algorithm.

In this paper, we perform an experimental evaluation of the piece and peer selection strategies as implemented in BitTorrent. Specifically, we have instrumented a client and run several experiments on torrents with different characteristics in order to evaluate the properties of the rarest first and choke algorithms. We do not pretend to have reached completeness, but to cover a large variety of today real cases. Our main conclusions on real torrents are the following.

- The rarest first algorithm guarantees a high diversity of the pieces. In particular, it prevents the apparition of rare pieces and of the last pieces problem.
- We have found that torrents in a startup phase can have a poor piece diversity. The duration of this phase depends only on the upload capacity of the source of the content. In particular, the rarest first algorithm is not responsible of the poor piece diversity during this phase.
- The fairness achieved with a bit level tit-for-tat strategy is not appropriate in the context of peer-to-peer file replication. We have proposed two new fairness criteria in this context.
- The choke algorithm is fair, fosters reciprocation, and is robust to free riders in its latest version.

Our contribution is to go beyond the mere observation of the good performance of BitTorrent. We provide new insights into the role of peer and piece selection for efficient peer-to-peer file replication. We show for the first time that on real torrents, the efficiency of the rarest first and choke algorithms do no justify their replacement by more complex solutions. Also, we identify, based on our observations, new area of improvements: the replication of the first pieces and the speed of delivery of the first copy of the content. Finally, we propose two new fairness criteria in the context of peer-to-peer file replication and we present for the first time results on the new version of the choke algorithm that fixes fundamental fairness issues.

Our conclusions significantly differ from the one presented in the literature [5], [10], [11], [13], [15], [18]. There are three main reasons to this divergence. First, we are in the context of peer-to-peer file replication in the Internet. As a consequence, the peers are well connected without severe bottlenecks in the network. The problems identified in the literature with the rarest first algorithm are in the context of networks with connectivity problems or low capacity bottlenecks. Second, we evaluated for the first time the new version of the choke algorithm. The evaluation of the choke algorithm in the literature was performed on the old version. We showed that the new version solves the problems identified on the old one. Finally, we performed an experimental evaluation on real torrents. Simulating peer-to-peer protocols is hard and requires many simplifications. In particular, all the simulations of BitTorrent we are aware of consider that each peer only knows few other peers, i.e., each peer has a small peer set [5], [11]. In the case of real torrents, the peer set size is much larger. The consequence is that BitTorrent builds a random graph, connecting the peers, that has a larger diameter in simulations than in real torrents. However, the diameter has a fundamental impact on the efficiency of the rarest first algorithm.

In this study, we show that on the specific context considered, i.e., the peer-to-peer file replication in the Internet, the rarest first and choke algorithms are enough. Even if we cannot extend our conclusions to other peer-to-peer contexts, we believe this paper to shed a new light on a today important problem.

The rest of the paper is organized as follows. We present the terminology used throughout this paper in section II-A. Then, we give a short overview of the BitTorrent protocol in section II-B, and we give a description of the rarest first and choke algorithms in section II-C. We present our experimentation methodology in section III, and our detailed results in section IV. Related work is discussed in section V. We conclude the paper with a discussion of the results in section VI.

II. Background

A. Terminology

The terminology used in the peer-to-peer community and in particular in the BitTorrent community is not standardized. For the sake of clarity, we define in this section the terms used throughout this paper.

- **Pieces and Blocks** Files transfered using BitTorrent are split in *pieces*, and each piece is split in *blocks*. Blocks are the transmission unit on the network, but the protocol only accounts for transfered pieces. In particular, partially received pieces cannot be served by a peer, only complete pieces can.
- **Interested and Choked** We say that peer A is *interested* in peer B when peer B has pieces that peer A does not have. Conversely, peer A is *not interested* in peer B when peer B only has a subset of the pieces of peer A. We say that peer A *chokes* peer B when peer A decides not to send data to peer B. Conversely, peer A *unchokes* peer B when peer A decides to send data to peer B.
- **Peer Set** Each peer maintains a list of other peers it can send pieces to. We call this list the *peer set*. The notion of peer set is also known as neighbor set.
- **Local and Remote Peers** We call *local peer* the peer with the instrumented BitTorrent client, and *remote peers* the peers that are in the peer set of the local peer.
- **Active Peer Set** A peer can only send data to a subset of its peer set. We call this subset the *active peer set*. The choke algorithm (described in section II-C.2) determines the peers being part of the active peer set, i.e., which remote peers will be choked and unchoked. Only peers that are unchoked by the local peer and interested in the local peer are part of the active peer set.
- **Leecher and Seed** A peer has two states: the *leecher state*, when it is downloading a content, but does not have yet all the pieces; the *seed state* when the peer has all the pieces of the content. For short, we say that a peer is a *leecher* when it is in leecher state and a *seed* when it is in seed state.

- **Initial Seed** The *initial seed* is the peer that is the first source of the content.
- **Rarest First Algorithm** The *rarest first algorithm* is the piece selection strategy used in BitTorrent. We give a detailed description of this algorithm in section II-C.1. The rarest first algorithm is also called the local rarest first algorithm.
- **Choke Algorithm** The *choke algorithm* is the peer selection strategy used in BitTorrent. We give a detailed description of this algorithm in section II-C.2. The choke algorithm is also called the tit-for-tat algorithm, or tit-for-tat like algorithm.
- **Rare and Available Pieces** We call the pieces only present on the initial seed *rare pieces*, and we call the pieces already served at least once by the initial seed *available pieces.*
- **Rarest Pieces and Rarest Pieces Set** The *rarest pieces* are the pieces that have the least number of copies in the peer set. In the case the least replicated piece in the peer set has m copies, then all the pieces with m copies form the *rarest pieces set*. The rarest pieces can be rare pieces or available pieces, depending on the number of copies of the rarest pieces.

B. BitTorrent Overview

BitTorrent is a P2P application that capitalizes on the bandwidth of peers to efficiently replicate contents on large sets of peers. A specificity of BitTorrent is the notion of *torrent*, which defines a session of transfer of a single content to a set of peers. Torrents are independent. In particular, participating to a torrent does not bring any benefit for the participation to another torrent. A torrent is alive as long as there is at least one copy of each piece in the torrent. Peers involved in a torrent cooperate to replicate the file among each other using *swarming* techniques [23]. In particular, the file is split in pieces of typically 256 kB, and each piece is split in blocks of 16 kB. Other piece sizes are possible.

A user joins an existing torrent by downloading a *.torrent* file usually from a Web server, which contains meta-information on the file to be downloaded, e.g., the piece size and the SHA-1 hash values of each piece, and the IP address of the so-called *tracker* of the torrent. The tracker is the only centralized component of BitTorrent, but it is not involved in the actual distribution of the file. It keeps track of the peers currently involved in the torrent and collects statistics on the torrent.

When joining a torrent, a new peer asks to the tracker a list of IP addresses of peers to build its initial peer set. This list typically consists of 50 peers chosen at random in the list of peers currently involved in the torrent. The initial peer set will be augmented by peers connecting directly to this new peer. Such peers are aware of the new peer by receiving its IP address from the tracker. Each peer reports its state to the tracker every 30 minutes in steady-state regime, or when disconnecting from the torrent, indicating each time the amount of bytes it has uploaded and downloaded since it joined the torrent. A torrent can thus be viewed as a collection of interconnected peer sets. If ever the peer set size of a peer falls below a predefined threshold, typically 20 peers, this peer will contact the tracker again to obtain a new list of IP addresses of peers. By default, the maximum peer set size is 80. Moreover, a peer should not exceed a threshold of 40 initiated connections among the 80 at each time. As a consequence, the 40 remaining connections should be initiated by remote peers. This policy guarantees a good interconnection among the peer sets in the torrent.

Each peer knows the distribution of the pieces for each peer in its peer set. The consistency of this information is guaranteed by the exchange of messages. The exchange of pieces among peers is governed by two core algorithms: the rarest first and the choke algorithms. These algorithms are further detailed in section II-C.

C. BitTorrent Algorithms Description

We focus here on the two core algorithms of BitTorrent: the rarest first and choke algorithms. We do not give all the details of these algorithms, but we explain the main ideas behind them.

1) Rarest First Algorithm: The rarest first algorithm works as follows. Each peer maintains the number of copies in its peer set of each piece. It uses this information to define a rarest pieces set. Let m be the number of copies of the rarest piece, then the index of each piece with m copies in the peer set is added to the rarest pieces set. The rarest pieces set of a peer is updated each time a copy of a piece is added to or removed from its peer set. Each peer selects the next piece to download at random in its rarest pieces set.

The behavior of the rarest first algorithm can be modified by three additional policies. First, if a peer has downloaded strictly less than 4 pieces, it chooses the next piece to request at random. This is called the *random first policy*. Once it has downloaded at least 4 pieces, it switches to the rarest first algorithm. The aim of the random first policy is to permit a peer to download its first pieces faster than with the rarest first policy, as it is important to have some pieces to reciprocate for the choke algorithm. Indeed, a piece chosen at random is likely to be more replicated than the rarest pieces, thus its download time will be on average shorter.

Second, BitTorrent also applies a *strict priority policy*, which is at the block level. When at least one block of a piece has been requested, the other blocks of the same piece are requested with the highest priority. The aim of the strict priority policy is to complete the download of a piece as fast as possible. As only complete pieces can be sent, it is important to minimize the number of partially received pieces.

Finally, the last policy is the *end game mode* [8]. This mode starts once a peer has requested all blocks, i.e., blocks are either requested or already received. While in this mode, the peer requests all blocks not yet received to all the peers in its peer set that have the corresponding blocks. Each time a block is received, it cancels the request for the received block to all the peers in its peer set that have the corresponding pending request. As a peer has a small buffer of pending requests, all blocks are effectively requested close to the end of the

download. Therefore, the *end game mode* is used at the very end of the download, thus it has little impact on the overall performance. We discuss the impact of the *end game mode* in section IV-A.3.

2) Choke Algorithm: The choke algorithm was introduced to guarantee a reasonable level of upload and download reciprocation. As a consequence, free riders, i.e., peers that never upload, should be penalized. For the sake of clarity, we describe without loss of generality the choke algorithm from the point of view of the local peer. In this section, *interested* always means interested in the local peer, and *choked* always means choked by the local peer.

The choke algorithm differs in leecher and seed states. We describe first the choke algorithm in leecher state. At most 4 remote peers can be unchoked and interested at the same time. Peers are unchoked using the following policy.

1) Every 10 seconds, the interested remote peers are ordered according to their download rate to the local peer and the 3 fastest peers are unchoked.
2) Every 30 seconds, one additional interested remote peer is unchoked at random. We call this random unchoke the optimistic unchoke.

In the following, we call the three peers unchoked in step 1 the regular unchoked (RU) peers, and the peer unchoked in step 2 the optimistic unchoked (OU) peer. The optimistic unchoke peer selection has two purposes. It allows to evaluate the download capacity of new peers in the peer set, and it allows to bootstrap new peers that do not have any piece to share by giving them their first piece.

We describe now the choke algorithm in seed state. In previous versions of the BitTorrent protocol, the choke algorithm was the same in leecher state and in seed state except that in seed state the ordering performed in step 1 was based on upload rates from the local peer. With this algorithm, peers with a high download rate are favored independently of their contribution to the torrent.

Starting with the version 4.0.0, the *mainline* client [2] introduced an entirely new algorithm in seed state. We are not aware of a documentation on this new algorithm and of an implementation of it apart from the *mainline* client.

We describe this new algorithm in seed state in the following. At most 4 remote peers can be unchoked and interested at the same time. Peers are unchoked using the following policy.

1) Every 10 seconds, the unchoked and interested remote peers are ordered according to the time they were last unchoked, most recently unchoked peers first.
2) For two consecutive periods of 10 seconds, the 3 first peers are kept unchoked and an additional 4th peer that is choked and interested is selected at random and unchoked.
3) For the third period of 10 seconds, the 4 first peers are kept unchoked.

In the following, we call the three or four peers that are kept unchoked according to the time they were last unchoked the seed kept unchoked (SKU) peers, and the unchoked peer selected at random the seed random unchoked (SRU) peer. With this new algorithm, peers are no more unchoked according to their upload rate from the local peer, but according to the time of their last unchoke. As a consequence, the peers in the active peer set are changed regularly, each new SRU peer taking an unchoke slot off the oldest SKU peer.

We will show in section IV-B.1 why the new choke algorithm in seed state is fundamental to the fairness of the choke algorithm.

III. Experimentation Methodology

A. Choice of the BitTorrent client

Several BitTorrent clients are available. The first BitTorrent client has been developed by Bram Cohen, the inventor of the protocol. This client is open source and is called *mainline*. As there is no well maintained and official specification of the BitTorrent protocol, the *mainline* client is considered as the reference of the BitTorrent protocol. It should be noted that, up to now, each improvement of Bram Cohen to the BitTorrent protocol has been replicated to all the other clients.

The other clients differ from the *mainline* client on two points. First, the *mainline* client has a basic user interface. Other clients have a more sophisticated interface with a nice look and feel, realtime statistics, many configuration options, etc. Second, as the *mainline* client defines the BitTorrent protocol, it is de facto a reference implementation of the BitTorrent protocol. Other clients offer experimental extensions to the protocol.

As our intent is an evaluation of the strict BitTorrent protocol, we have decided to restrict ourselves to the *mainline* client. This client is very popular as it is the second most downloaded BitTorrent client at SourceForge with more than 52 million downloads. We instrumented the version 4.0.2 of the *mainline* client released at the end of May 2005[1].

B. Experimentations

We performed a complete instrumentation of the *mainline* client. The instrumentation comprises: a log of each BitTorrent message sent or received with the detailed content of the message, a log of each state change in the choke algorithm, a log of the rate estimation used by the choke algorithm, and a log of important events (end game mode, seed state).

All our experimentations were performed with the default parameters of the *mainline* client. It is outside of the scope of this study to evaluate the impact of each BitTorrent parameters variation. The main default parameters are: the maximum upload rate (default to 20 kB/s), the minimum number of peers in the peer set before requesting more peers to the tracker (default to 20), the maximum number of connections the local peer can initiate (default to 40), the maximum number of peers in the peer set (default to 80), the number of peers in the active peer set including the optimistic unchoke (default to 4), the block size (default to 2^{14} Bytes), the number of pieces downloaded before switching from random to rarest first piece selection (default to 4).

[1]The latest branch of development is 4.4.x. In this branch, there is no new functionality to the core protocol, but a new tracker-less functionality and some improvements to the client. As the evaluation of the tracker functionality was outside the scope of this study we focused on version 4.0.2.

In our experiments, we uniquely identify a peer by its IP address and peer ID. The peer ID, which is 20 bytes, is a string composed of the client ID and a randomly generated string. This random string is regenerated each time the client is restarted. The client ID is a string composed of the client name and version number, e.g., M4-0-2 for the *mainline* client in version 4.0.2. We are aware of around 20 different BitTorrent clients, each client existing in several different versions. When in a given experiment, we see several peer IDs corresponding to the same IP address[2], we compare the client ID of the different peer IDs. In the case the client ID is the same for all the peer IDs on a same IP address, we deem that this is the same peer. We cannot rely on the peer ID comparison, as each time a client crashes or restarts, the random string is regenerated. The pair (IP, client ID) does not guarantee that each peer can be uniquely identified, because several peers beyond a NAT can use the same client in the same version. However, considering the large number of client IDs, it is common in our experiments to observe 15 different client IDs, the probability of collision is reasonably low for our purposes. Unlike what was reported by Bhagwan et al. [4] for the Overnet file sharing network, we did not see any problem of peer identification due to NATs. In fact, BitTorrent has an option, activated by default, to prevent accepting multiple concurrent incoming connections from the same IP address. The idea is to prevent peers to increase their share of the torrent, by opening multiple clients from the same machine. Therefore, even if we found in our traces different peers with the same IP address at different moment in time, two different peers with the same IP address cannot be connected to the local peer during overlapping periods.

We did all our experimentations from a machine connected to a high speed backbone. However, the upload capacity is limited by default by the client to 20 kB/s. There is no limit to the download capacity. We obtained effective maximum download speed ranging from 20 kB/s up to 1500 kB/s depending on the experiments.

We ran between 1 and 3 experiments on 20 different torrents. We considered copyrighted and free contents, which are TV shows, movies, cartoons, music albums, live concert recordings, and softwares. Each experiment lasted for 8 hours in order to make sure that each client became a seed and to have a representative trace in seed state.

We give the characteristic of each torrent in Table I. The number of seeds and leechers is given at the beginning of the experiment. Therefore, these numbers can be very different at the end of the experiment.

TABLE I
TORRENT CHARACTERISTICS.

Torrent	# of Seeds	# of Leechers	Size (MB)
1	50	18	600
2	1	40	800
3	1	2	580
4	115	19	430
5	160	5	6
6	102	342	200
7	9	30	350
8	1	29	350
9	12612	7052	140
10	462	180	2600
11	1	130	820
12	30	230	820
13	0	66	700
14	3	612	1413
15	3697	7341	349
16	1	50	1419
17	11641	5418	350
18	11975	4151	350
19	514	1703	349
20	20	126	184

C. Limitations

We took during this work two decisions that restrict the scope of this study. We have chosen to focus on the behavior of a single client in a real torrent. Whereas it may be argued that a larger number of instrumented peers would have given a better understanding of the torrents, we took the decision to be as unobtrusive as possible. Increasing the number of instrumented clients would have required to either control those clients ourselves, or to ask some peers to use our instrumented client. In both cases, the choice of the instrumented peer set would have been biased, and the behavior of the torrent impacted. On the contrary, our decision was to understand how a new peer (our instrumented peer) joining a real torrent behaves.

A second decision was to evaluate only real torrents. In such a context it is not possible to reproduce an experiment, and thus to gain statistical information because each experiment depends on the behavior of peers, the number of seeds and leechers in the torrent, and the subset of peers randomly returned by the tracker. However, studying the dynamic of the protocol is as important as studying its statistical properties. Also, as we considered torrents with different characteristics and observed a consistent behavior on these torrents, we believe our observations to be representative of the rarest first and choke algorithms.

IV. EXPERIMENTATION RESULTS

A. Rarest First Algorithm

We define the entropy as the repartition of pieces among peers. We say that there is ideal entropy when each leecher[3] is always interested in any other leecher. We do not claim that ideal entropy can be always achieved, but it should be the objective of any efficient piece selection strategy. Indeed, a poor entropy may adversely impact the service capacity of the torrent by biasing the peer selection strategy

We evaluated the rarest first algorithm on a representative set of real torrents. We showed that the rarest first algorithm achieves an entropy close to the ideal one, and that its replacement by more complex solutions cannot be justified. Then, we evaluated the dynamics of the rarest first algorithm

[2]Between 0% to 26% of the IP addresses, depending on the experiments, are associated in our traces to more than one peer ID. The mean is around 9%.

[3]Only the case of leechers is relevant for the entropy characterization, as seeds are always interesting for leechers and never interested in leechers.

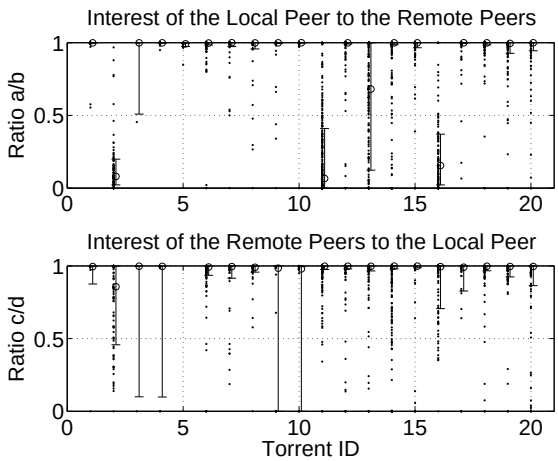

Fig. 1. *Entropy characterization.* ***Top subplot:*** *For each remote leecher peer for a given torrent, a dot represents the ratio $\frac{a}{b}$ where a is the time the local peer in leecher state is interested in this remote peer and b is the time this remote peer spent in the peer set when the local peer is in leecher state.* ***Bottom subplot:*** *For each remote leecher peer for a given torrent, a dot represents the ratio $\frac{c}{d}$ where c is the time this remote peer is interested in the local peer in leecher state and d is the time this remote peer spent in the peer set when the local peer is in leecher state.* ***For both subplots:*** *Each vertical solid lines represent the 20th percentile (bottom of the line), the median (identified with a circle), and the 80th percentile (top of the line) of the ratios for a given torrent.*

to understand the reasons for this good entropy. Finally, we focused on a specific problem called the last pieces problem, which is presented [11], [18] as a major weakness of the rarest first strategy. We showed that the last pieces problem is overestimated. In contrast, we identified a first blocks problem, which is a major area of improvement for BitTorrent.

1) Entropy Characterization: The major finding of this section is that the rarest first algorithm achieves an entropy close to the ideal one for real torrents. According to our definition of ideal entropy, each leecher is always interested in any other leecher. As we do not have global knowledge of the torrent, we characterize the entropy from the point of view of the local peer with two ratios. For each remote peer we compute:

- the ratio $\frac{a}{b}$ where a is the time the local peer in leecher state is interested in this remote peer and b is the time this remote peer spent in the peer set when the local peer is in leecher state;
- the ratio $\frac{c}{d}$ where c is the time this remote peer is interested in the local peer in leecher state and d is the time this remote peer spent in the peer set when the local peer is in leecher state.

In the case of ideal entropy the above ratios should be one. Fig. 1 gives a characterization of the entropy for the torrents considered in this study.

For most of our torrents, we see in Fig. 1 that the ratios are close to 1, thus an entropy close to the ideal one. For the top subplot, 75% of the torrents have the 80th percentile close to one. For the bottom subplot, 70% of the torrents have a 80th percentile close to one, and 90% of the torrents have a median close to one. We discuss below the case of the torrents with a poor entropy.

First, we discuss why the local peer is often not interested in the remote peers for torrents 2, 3, 11, 13, and 16, see Fig. 1, top subplot. These torrents have a poor entropy because they are in a startup phase. This means that the initial seed has not yet served all the pieces of the content. We remind that the pieces only present on the initial seed are the *rare pieces*, and that the pieces already served at least once by the initial seed are the *available pieces*, see section II-A. The reason for the poor observed entropy is that during a torrent startup, available pieces are replicated with an exponential capacity of service [25], but rare pieces are served by the initial seed at a constant rate. Thus, available pieces are replicated faster than rare pieces. This leads to two problems. First, the probability to have peers in a peer set with the same subset of pieces is higher during the torrent startup than when there is no rare piece in the torrent. Second, when there is no rare piece, a peer with all the available pieces becomes a seed. But, when there are rare pieces, a peer with all the available pieces remains a leecher because it does not have the rare pieces. However, these leechers cannot be interested in any other peer as they have all the available pieces at this point of time, but they stay in the peer set of the local peer. Thus a poor ratio for these leechers in Fig. 1. In conclusion, the poor entropy we observed is not due to a deficiency of the rarest first algorithm, but to the startup phase of the torrent whose duration depends only on the upload capacity of the initial seed. We discuss further this point in section IV-A.2.a.

Now, we discuss why the remote peers are often not interested in the local peer for torrents 2, 3, 4, 5, 9, and 10, see Fig. 1, bottom subplot. No dot is displayed for torrent 5 because due to the small number of leechers in this torrent, the local peer in leecher state had no leecher in its peer set. Four torrents have a 80th percentile close to 0. The percentile for these torrents is computed on a small number of ratios: 3, 8, 15, and 12 for torrents 3, 4, 9 and 10 respectively. Therefore, the 80th percentile is not representative as it is not computed on a set large enough. Additionally, the reason for the low 80th percentile is peers with a ratio of 0. We identified two reasons for a ratio of 0. First, some peers join the peer set with almost all pieces. They are therefore unlikely to be interested in the local peer. Second, some peers with no or few pieces never sent an interested message to the local peer. This can be explained by a client behavior changed with a plugin or an option activation. The super seeding option [3] available in several BitTorrent clients has this effect. In conclusion, the poor entropy of some peers is either a measurement artifact due to modified or misbehaving clients, or the result of the inability of the rarest first algorithm to reach ideal entropy in some extreme cases.

We have seen that peers that join the torrent with almost all pieces may not be interested in the local peer. In this scenario, the rarest first algorithm does guarantee ideal entropy. However, we argue that this case does not justify the replacement of the rarest first algorithm for two reasons. First, this case appears seldom and does not significantly impact the overall entropy of the torrent. Second, the peers with a poor entropy are peers that join the peer set with only few missing pieces. In the case of torrent startup, it is not clear whether

a solution based, for instance, on source or network coding would have proposed interesting pieces to such peers. Indeed, when a content is split into k pieces, there is no solution based on coding that can reconstruct the content in less than k pieces. For this reason, when the initial seed has not yet sent at least one copy of each piece, there is no way to reconstruct the content, so no way to have interesting pieces for all the peers.

For the computation of the ratios on Fig. 1, we did not consider peers that spent less than 10 seconds in the peer set. Our motivation was to evaluate the entropy of pieces in a torrent. However, due to several misbehaving clients, there is a permanent noise created by peers that join and leave the peer set frequently. Such peers stay typically less than few seconds in the peer set, and they do not take part in any active upload or download. Therefore, these misbehaving peers adversely bias our entropy characterization. Filtering all peers that stay less than 10 seconds remove the bias.

In summary, we have seen that the rarest first algorithm enforces an entropy close to the ideal one. We have identified torrent with a poor entropy and showed that the rarest first algorithm is not responsible for this poor entropy. We have also identified seldom cases where the rarest first algorithm does not perform optimally, but we have explained that these cases do not justify a replacement with a more complex solution. In the following, we evaluate how the rarest first piece selection strategy achieves a high entropy.

2) Rarest First Algorithm Dynamics: We classify a torrent in two states: the transient state and the steady state[4]. In transient state, there is only one seed in the torrent. In particular, there are some pieces that are rare, i.e., present only at the seed. This state corresponds to the beginning of the torrent, when the initial seed has not yet uploaded all the pieces of the content. All the torrents with a poor entropy in Fig. 1, top subplot, are in a transient state. A good piece replication algorithm should minimize the time spent in the transient state because a poor entropy may adversely impact the service capacity of a torrent by biasing the peer selection strategy. In steady state, there is no rare piece, and the piece replication strategy should prevent the torrent to enter again a transient state. All the torrents with a high entropy are in steady state.

In the following, we evaluate how the rarest first algorithm performs in transient and steady state. We show that the poor entropy of torrents experienced in transient state is due to the limited upload capacity of the initial seed, and that the rarest first algorithm minimizes the time spent in this state. We also show that the rarest first algorithm is efficient at keeping a torrent in steady state, thus guaranteeing a high entropy.

a) Transient State: In order to understand the dynamics of the rarest first algorithm in transient state, we focus on torrent 11. The file distributed in this torrent is split in 1657 pieces. We run this experiment during 32828 seconds.

Torrent 11 was in transient state for most of the experiment. We probed the tracker to get statistics on the number of seeds and leechers during this experiment. We found that this torrent

[4]Our definition of transient and steady state differs from the one given by Yang et al. [25].

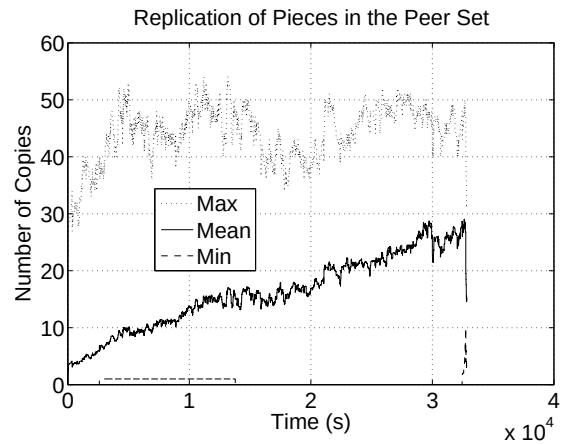

Fig. 2. *Evolution of the number of copies of pieces in the peer set with time for torrent 11.* ***Legend:*** *The dotted line represents the number of copies of the most replicated piece in the peer set at each instant. The solid line represents the mean number of copies over all the pieces in the peer set at each instant. The dashed line represents the number of copies of the least replicated piece in the peer set at each instant.*

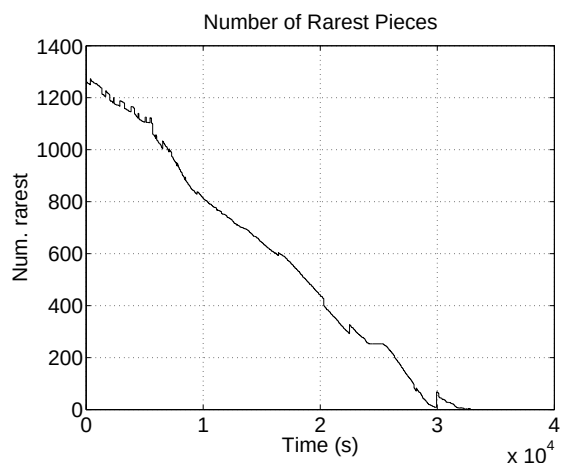

Fig. 3. *Evolution of the number of rarest pieces in the peer set for torrent 11. The rarest pieces set is formed by the pieces that are equally the rarest, i.e., the pieces that have the least number of copies in the peer set.*

had only one seed for the duration of most of our experiment. Moreover, in the peer set of the local peer, there was no seed in the intervals [0,2594] seconds and [13783,32448] seconds. This is confirmed by Fig. 2 that shows that the least replicated piece (min curve) has either a single copy in the peer set when the seed is in the peer set, or is a missing piece when the seed leaves the peer set.

We see in Fig. 1, top subplot, that torrent 11 has a poor entropy. This poor entropy is due to the limited upload capacity of the initial seed. Indeed, when a torrent is in transient state, available pieces are replicated with an exponential capacity of service [25], but rare pieces are served by the initial seed at a constant rate. We see in Fig. 3 that at the beginning of the experiment around 1250 over 1657 pieces are rare. The number of rarest pieces, i.e., the set size of the pieces that are equally rarest, decreases linearly with time. As the size of each piece in this torrent is 512 kB, a rapid calculation shows that the rarest pieces are duplicated in the peer set at a rate

close to 20 kB/s. We do not have a direct proof that this rate is the one of the initial seed, because we do not have global knowledge of the torrent. However, we have two reasons to believe it is reasonable. First, the default upload rate on most of the BitTorrent clients is set to 20 kB/s. Thus this upload rate is likely to correspond to a single client. Second, the torrent is in its startup phase and most of the pieces are only available on the initial seed. Therefore, only the initial seed can serve the rare pieces shown in Fig. 3. In conclusion, the upload capacity of the initial seed is the bottleneck for the replication of the rare pieces, and the time spent in transient state only depends on the upload capacity of the initial seed.

The rarest first algorithm minimizes the time spent in transient state and replicates fast available pieces. Indeed, leechers download first the rare pieces. As the rare pieces are only present on the initial seed, the upload capacity of the initial seed will be fully utilized and no or few duplicate rare pieces will be served by the initial seed. We see in Fig. 3 that there is no significant slope change when the seed leaves the peer set. Therefore, pieces missing in the peer set are served by peers outside the peer set at the same rate as pieces present in the peer set. Once served by the initial seed, a rare piece becomes available and is served in the torrent with an increasing capacity of service. As rare pieces are served at a constant rate, most of the capacity of service of the torrent is used to replicate the available pieces on the leechers. Indeed, Fig. 2 shows that once a piece is served by the initial seed, the rarest first algorithm will start to replicate it fast as shown by the continuous increase in the mean number of copies over all the peers.

In summary, the poor entropy observed for some torrents is due to the transient phase. The duration of this phase cannot be shorter than the time for the initial seed to send one copy of each piece, which is constrained by the upload capacity of the initial seed. Thus, the time spent in this phase cannot be shorten further by the piece replication strategy. The rarest first algorithm minimizes the time spent in transient state. Once a piece is served by the initial seed, the rarest first algorithm replicates it fast. Therefore, a replacement of the rarest first algorithm by another algorithm cannot be justified based on the real torrents we have monitored in transient state.

b) Steady State: In order to understand the dynamics of the rarest first algorithm in steady state, we focus on torrent 7. We have seen on Fig. 1 that torrent 7 has a high entropy. Fig. 4 shows that the least replicated piece (min curve) has always more than 1 copy in the peer set. Thus, torrent 7 is in steady state. The content distributed in this torrent is split in 1395 pieces.

In the following, we present the dynamics of the rarest first algorithm in steady state, and explain how this algorithm prevents the torrent to return in transient state. Fig. 4 shows that the mean number of copies remains well bounded over time by the number of copies of the most and least replicated pieces. In particular, the number of copies of the least replicated piece remains close to the mean. The variation observed in the number of copies are explained by the variation of the peer set size, see Fig. 5. The decrease in the number of copies 13680 seconds after the beginning of the experiment corresponds to

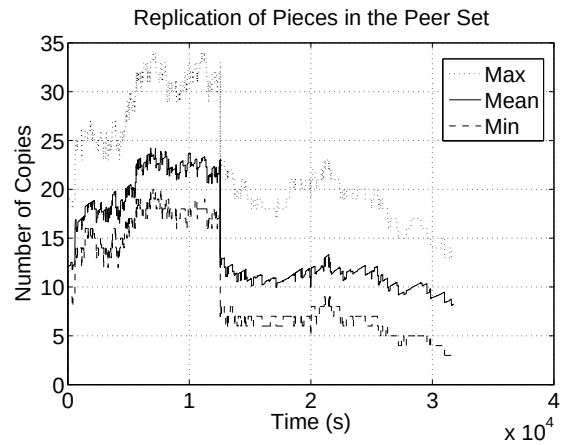

Fig. 4. *Evolution of the number of copies of pieces in the peer set with time for torrent 7.* ***Legend:*** *The dotted line represents the number of copies of the most replicated piece in the peer set at each instant. The solid line represents the mean number of copies over all the pieces in the peer set at each instant. The dashed line represents the number of copies of the least replicated piece in the peer set at each instant.*

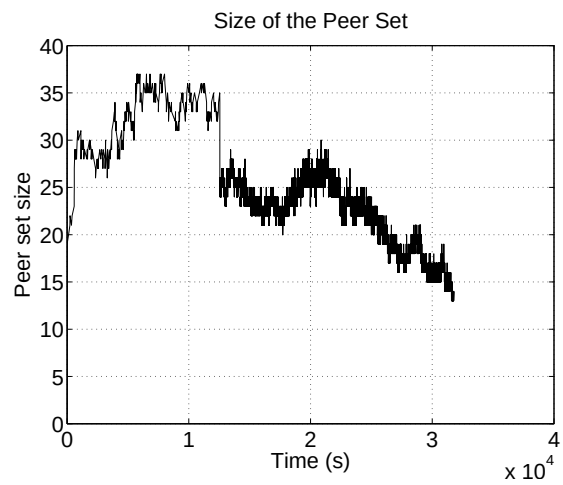

Fig. 5. *Evolution of the peer set size for torrent 7.*

the local peer switching to seed state. Indeed, when a leecher becomes a seed, it closes its connections to all the seeds.

The rarest first algorithm does a very good job at increasing the number of copies of the rarest pieces. Fig. 4 shows that the number of copies of the least replicated piece (min curve) closely follows the mean, but does not significantly get closer. However, we see in Fig. 6 that the number of rarest pieces, i.e., the set size of the pieces that are equally rarest, follow a sawtooth behavior. Each peer joining or leaving the peer set can alter the set of rarest pieces. But, as soon as a new set of pieces becomes rarest, the rarest first algorithm quickly duplicates them as shown by a consistent drop in the number of rarest pieces in Fig.6. Finally, we never observed in any of our torrents a steady state followed by a transient state.

In summary, the rarest first algorithm in steady state ensures a good replication of the pieces in real torrents. It also replicates fast the rarest pieces in order to prevent the apparition of a transient state. We conclude that on real torrents in steady state, the rarest first algorithm is enough to guarantee a high

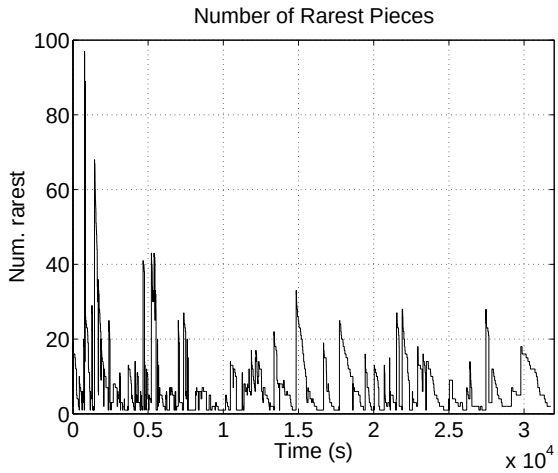

Fig. 6. *Evolution of the number of rarest pieces in the peer set for torrent 7. The rarest pieces set is formed by the pieces that are equally the rarest, i.e., the pieces that have the least number of copies in the peer set.*

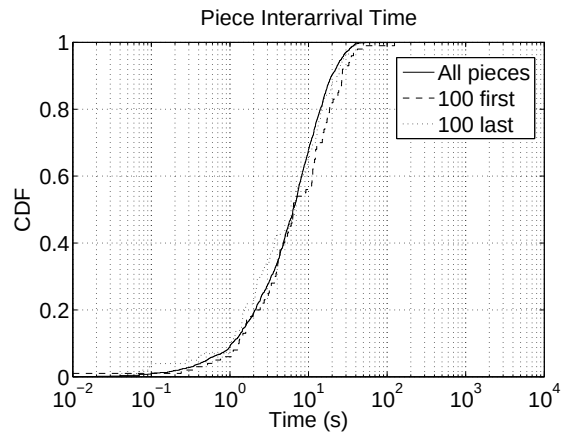

Fig. 7. *CDF of the piece interarrival time for torrent 7.* ***Legend:*** *The solid line represents the CDF for all pieces, the dashed line represents the CDF for the 100 first downloaded pieces, and the dotted line represents the CDF for the 100 last downloaded pieces.*

entropy.

3) Last Pieces Problem: We say that there is a last pieces[5] problem when the download speed suffers a significant slow down for the last pieces. This problem is due to some pieces replicated on few overloaded peers, i.e., peers that receive more requests than they can serve. This problem is detected by a peer only at the end of the content download. Indeed, a peer always seeks for fast peers to download from. Thus, it is likely that if some pieces are available on only few overloaded peers, these peers will be chosen only at the end of the content download when there is no other pieces to download.

We have performed all our experiments with the end game mode enabled as it does not hide a last pieces problem. Indeed, the end game mode intent is mistakenly considered to suppress the last pieces problem. This mode was first proposed by Rodriguez et al. [23] to solve the termination idle time during a parallel download. The termination idle time is not related to the rarity of a piece, but to a decrease in capacity of service when there are fewer pieces to request than peers to serve them. In this case, some peers remain idle. Rodriguez's solution is to request such idle peers with pieces already requested to other peers. This way, the perceived capacity of service is at least the one of the fastest active peer. However, in the case the last pieces are on few overloaded peers, the end game mode will not speed up significantly the end of the download. Thus, a last pieces problem can be detected even with the end game mode enabled.

In the following, we show that the last pieces problem is overstated, but the first blocks problem is underestimated and an important possibility of performance improvement.

Due to space limitation, we only present plots for torrent 7 that is in steady state, but we discuss the results for the other torrents.

Fig. 7 shows that there is no last pieces problem for torrent 7. The 100 first and 100 last pieces have roughly the same interarrival time distribution than all the pieces. We observed the same result in all our experiments for torrents in steady state.

For two torrents in transient state, we observed a slow down for the 100 last pieces. This slow down is not due to the rarest first algorithm, but to the limited upload capacity of the initial seed. Indeed, during the transient phase available pieces are replicated with an exponential capacity of service [25], but rare pieces are served by the initial seed at a constant rate. Therefore, when a peer enters a torrent in transient state, the first pieces it receives are available pieces, i.e., pieces that can be served by several different peers. But, once the peer has all the available pieces, the remaining pieces are rare and can be received at most at the upload capacity of the initial seed. The download speed of the rare pieces will be lower than the one of the available pieces. Thus, a download slow down for the last pieces. In conclusion, the last pieces problem is seldom and may appear only for torrents in transient state. Moreover, the slow down for the last pieces does not depend on the number of peers in the torrent, but only on the upload capacity of the initial seed.

It is important to study the piece interarrival time, because partially received pieces cannot be retransmitted by a BitTorrent client, only complete pieces can. However, pieces are split in blocks, which are the BitTorrent unit of data transfer. For this reason we have also evaluated the block interarrival time.

We see in Fig. 8 that there is no last blocks problem, but a first blocks problem. The curve for the last 100 blocks is very close to the one for all blocks. But, the interarrival time for the 100 first blocks is larger than for the 100 last blocks. We have never observed a last blocks problem in all our experiments for torrents in steady state. As the interarrival time for the last 100 blocks did not increase, the local peer did not suffer from a slow down at the end of the download.

However, we found several times a first blocks problem. This is due to the startup phase of the local peer, which depends on the set of peers returned by the tracker and the moment at which the remote peers decide to *optimistically unchoke* or *seed random unchoke* the local peer, see section II-

[5]This problem is usually referenced as the last piece (singular) problem. However, there is no reason why this problem affects only a single piece.

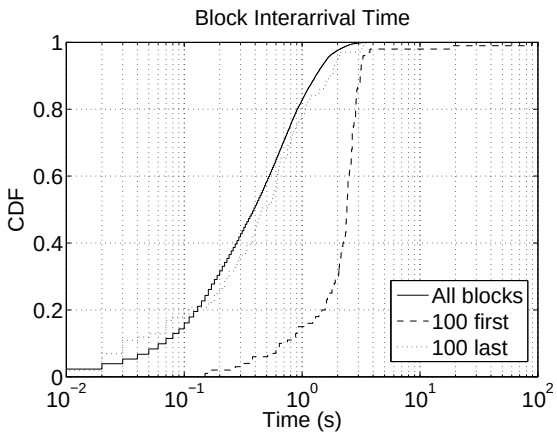

Fig. 8. *CDF of the block interarrival time for torrent 7.* ***Legend:*** *The solid line represents the CDF for all blocks, the dashed line represents the CDF for the 100 first downloaded blocks, and the dotted line represents the CDF for the 100 last downloaded blocks.*

C.2. We have observed seldom last blocks problem on torrents in transient state. The explanation is the same as for the last pieces problem for torrents in transient state.

In summary, a last pieces problem appears seldom on torrent in transient state only. This problem is inherent to the transient state of the torrent, and is not due to the rarest first algorithm. Moreover, the rarest first algorithm is efficient at mitigating this problem by replicating fast rare pieces once they become available. However, we observed a first blocks problem. This first blocks problem results in a slow startup of the torrent, which is an area of improvement for BitTorrent.

B. Choke Algorithm

1) Fairness Issue: Several recent studies [5], [10], [13], [15] challenge the fairness properties of the choke algorithm because it does not implement a bit level tit-for-tat, but a coarse approximation based on short term download estimations. Moreover, it is believed that a fair peer selection strategy must enforce a byte level reciprocation. For instance, a peer A refuses to upload data to a peer B if the amount of bytes uploaded by A to B minus the amount of bytes downloaded from B to A is higher than a given threshold [5], [10], [15]. The rationale behind this notion of fairness is that free riders should be penalized, and reciprocation should be enforced. We call this notion of fairness, tit-for-tat fairness.

We argue in the following that tit-for-tat fairness is not appropriate in the context of peer-to-peer file replication. A peer-to-peer session consists of seeds, leechers, and free riders, i.e., leechers that never upload data. We consider the free riders as a subset of the leechers. With tit-for-tat fairness, when there is more capacity of service in the torrent than request for this capacity, the excess capacity will be lost even if slow leechers or free riders could benefit from it. Excess capacity is not rare as it is a fundamental property of peer-to-peer applications. Indeed, there are two important characteristics of peer-to-peer applications that tit-for-tat fairness does not take into account. First, leechers can have an asymmetrical network connectivity, the upload capacity being lower than the download capacity. In the case of tit-for-tat fairness, a leecher will never be able to use its full download capacity even if there is excess capacity in the peer-to-peer session. Second, a seed cannot evaluate the reciprocation of a leecher, because a seed does not need any piece. As a consequence, there is no way for a seed to enforce tit-for-tat fairness. But, seeds can represent an important part of a peer-to-peer session, see Table I. For this reason, it is fundamental to have a notion of fairness that take into account seeds.

In the following, we present two fairness criteria that take into account the characteristics of leechers and seeds and the notion of excess capacity:

- Any leecher i with an upload speed U_i should get a lower download speed than any other leecher j with an upload speed $U_j > U_i$.
- A seed should give the same service time to each leecher.

With these two simple criteria, leechers are allowed to use the excess capacity, but not at the expense of leechers with a higher level of contribution. Reciprocation is fostered and free riders are penalized. Seeds do not make a distinction between contributing leechers and free riders. However, free riders cannot compromise the stability of the system because the more there are contributing leechers, the less the free riders receive from the seeds.

To summarize the above discussion, tit-for-tat fairness is not appropriate in the context of peer-to-peer file replication protocols like BitTorrent. For this reason, we proposed two new criteria of fairness, one for leechers and one for seeds. It is beyond the scope of this study to perform a detailed discussion of the fairness issues for peer-to-peer protocols. Our intent is to give a good intuition on how a peer-to-peer protocol should behave in order to achieve a reasonable level of fairness.

In the following, we show on real torrents that the choke algorithm in leecher state fosters reciprocation, and that the choke algorithm in seed state gives the same service time to each leecher. We conclude that the choke algorithm is fair.

2) Leecher State: The choke algorithm in leecher state fosters reciprocation. We see in Fig. 9 that peers that receive the most from the local peer (top subplot) are also peers from which the local peer downloaded the most (bottom subplot). Indeed, the same color in the top and bottom subplots represents the same set of peers. All seeds are removed from the data used for the bottom plot, as it is not possible to reciprocate data to seeds. This way, a ratio of 1 in the bottom subplot represents the total amount of bytes downloaded from leechers.

Two torrents present a different characteristic. The local peer for torrent 5 does not upload any byte in leecher state because due to the small number of leechers in this torrent, the local peer in leecher state had no leecher in its peer set. Torrents 16, which is in transient state, has a poor level of reciprocation. This is explained by a single leecher that gave to the local peer half of the pieces, but who received few pieces from the local peer. The reason is that this remote leecher was almost never interested in the local peer. This problem is due to the poor entropy of the torrent in transient state.

We now focus on torrent 7. The local peer stayed 228 minutes in leecher state and 334 minutes in seed state. Because

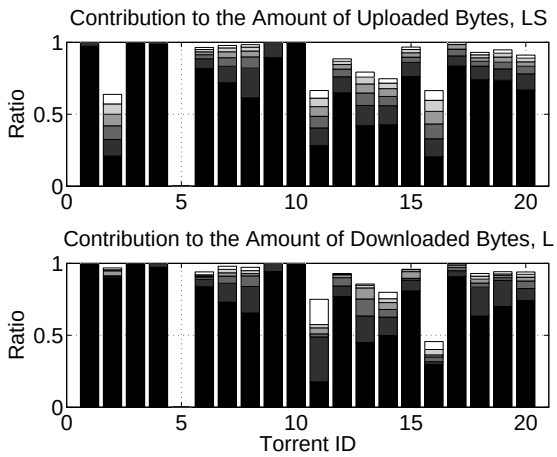

Fig. 9. *Fairness characterization of the choke algorithm in leecher state for each torrent.* ***Top subplot:*** *Amount of bytes uploaded from the local peer to remote peers. We created 6 sets of 5 remote peers each, the first set (in black) contains the 5 remote peers that receive the most bytes from the local peer. Each next set contains the next 5 remote peers. The sets representation goes from black for the set containing the 5 best remote downloaders, to white for the set containing the 25 to 30 best downloaders.* ***Bottom subplot:*** *Amount of bytes downloaded from remote peers to the local peer. The same set construction is kept. Thus, this plot shows how much each set of downloaders, as defined in the top subplot, uploaded to the local peer.*

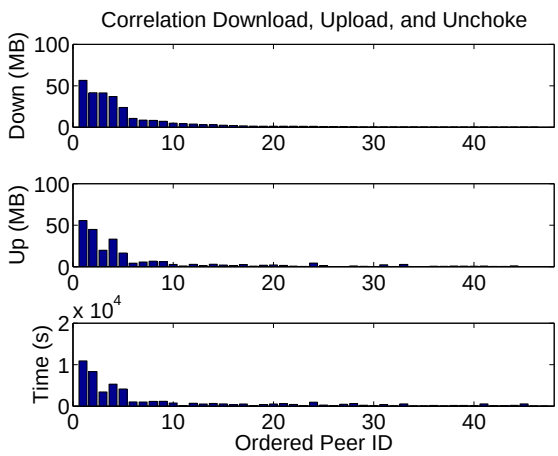

Fig. 10. *Reciprocation characterization of the choke algorithm in leecher state for torrent 7.* ***Top subplot:*** *Amount of bytes downloaded from the remote peers.* ***Middle subplot:*** *Amount of bytes uploaded to the remote peers.* ***Bottom subplot:*** *Total unchoked time of the remote peers.* ***All subplots:*** *Peers are ordered according to the amount of bytes downloaded (top subplot), the same order is kept for the two other subplots.*

the choke algorithm takes its decisions based on the current download rate of the remote peers, it does not achieve a perfect reciprocation of the amount of bytes downloaded and uploaded. However, Fig. 10 shows that the peers from which the local peer downloads the most are also the peers the most frequently unchoked and the peers that receive the most uploaded bytes. Thus the level of reciprocation is good.

The above results show that with a simple distributed algorithm and without any stringent reciprocation requirements, unlike tit-for-tat fairness, one can achieve a good reciprocation. More importantly, the choke algorithm in leecher state allows leechers to benefit from the excess capacity. It is important

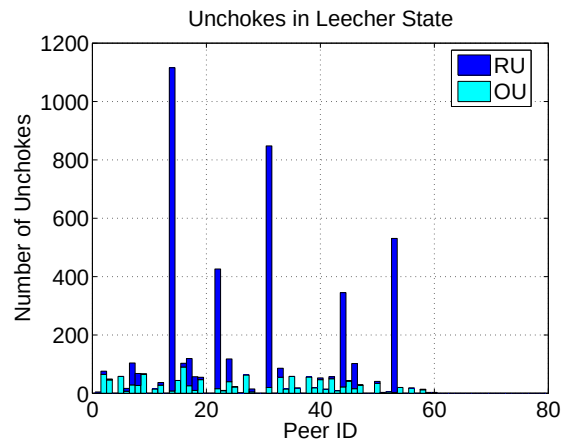

Fig. 11. *Number of times each peer is unchoked when the local peer is in the leecher state for torrent 7.* ***Legend:*** *RU is for regular unchoke, OU is for optimistic unchoke. Peer IDs are ordered according to the time each peer first entered the peer set. All peers IDs for the entire experiment are given.*

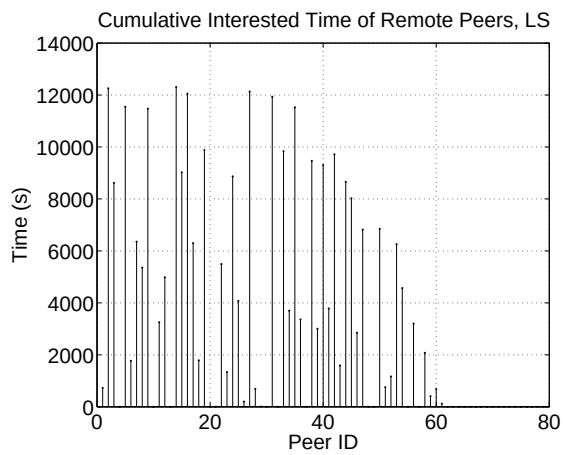

Fig. 12. *Cumulative interested time of the remote peers in the pieces of the local peer, when the local peer is in leecher state for torrent 7.*

to understand why the choke algorithm achieves this good reciprocation. One reason is the way the active peer set is built. In the following, we focus on how the local peer selects the remote peers to upload blocks to.

The choke algorithm in leecher state selects a small subset of peers to upload blocks to. We see in Fig. 9, top subplot, that the 5 peers that receive the most data from the local peer (in black) represents a large part of the total amount of uploaded bytes. At first sight, this behavior is expected from the choke algorithm because a local peer selects the three fastest downloading peers to upload to, see section II-C.2. However, there is no guarantee that these three peers will continue to send data to the local peer. In the case they stop sending data to the local peer, the local peer will also stop reciprocating to them.

We focus again on torrent 7 in order to understand how this subset of peers is selected. Fig. 11 shows that most of the peers are optimistically unchoked, and few peers are regularly unchoked a lot of time. The optimistic unchoke acts as a peer discovery mechanism. The peers that are not unchoked

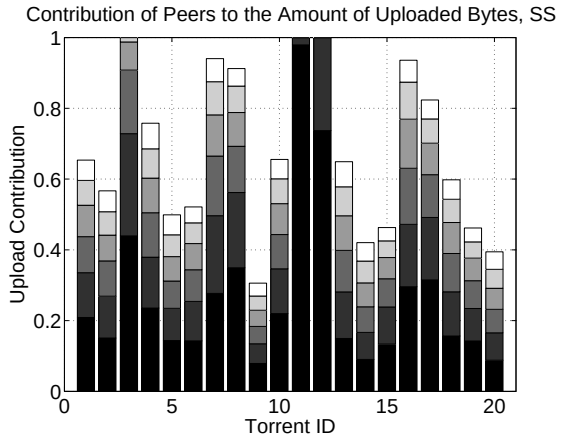

Fig. 13. *Fairness characterization of the choke algorithm in seed state for each torrent.* ***Legend:*** *We created 6 sets of 5 remote peers each, the first set (in black) contains the 5 remote peers that receive the most bytes from the local peer. Each next set contains the next 5 remote peers. The set representation goes from black from the set containing the 5 best remote downloaders, to white for the set containing the 25 to 30 best downloaders.*

at all are either initial seeds, or peers that do not stay in the peer set long enough to be optimistically unchoked. After 600 seconds of experiment and up to the end of the leecher state, a minimum of 18 and a maximum of 28 peers are interested in the local peer. In leecher state, the local peer is interested in a minimum of 19 and a maximum of 37 remote peers. Therefore, the result is not biased due to a lack of peers, or to a lack of interest. We see in Fig. 12 that there is no correlation between the cumulative interested time and the number of times a peer is regularly unchoked. Thus, peers to unchoke are selected based on their reciprocation level only.

Fig. 9 shows that for three torrents in transient state, torrents 2, 11 and 16, the amount of bytes uploaded by the 30 best remote peers is lower than for the other torrents. Torrents in transient state have a poor entropy. Therefore, the peers are no more selected based only on their reciprocation level, but also on the pieces available. For this reason, a larger set of peers receives pieces from the local peer. Thus, a lower fraction of bytes uploaded to the best remote peers.

In summary, we have seen that the choke algorithm guarantees a good level of reciprocation. Thus, it fosters reciprocation. One important reason is that each peer elects a small subset of peers to upload data to. This stability improves the level of reciprocation. We have seen that this stability is not due to a lack of interest. Our guess is that the choke algorithm leads to an equilibrium in the peer selection. The exploration of this equilibrium is fundamental to the understanding of the choke algorithm efficiency. It is beyond the scope of this study to do this analysis, but it is an important area of future research.

3) Seed State: The new choke algorithm in seed state gives the same service time to each remote peer. We see in Fig. 13 that each peer receives roughly the same amount of bytes from the local peer. The differences among the peers are due to the time spent in the peer set. The more time spent in the peer set, the more time a peer is unchoked. For torrents 11 and 12 the five best downloaders receive most of the bytes, because for both torrents there were less than 10 remote peers that received bytes from the local peer.

This new version of the choke algorithm in seed state is the only one to give the same service time to each leecher. This has three fundamental benefits compared to the old version. First, as each leecher receives a small and equivalent service time from the seeds, the entropy of the pieces is improved. In contrast, with the old choke algorithm, a few fast leechers can receive most of the pieces, which decreases the diversity of the pieces. Second, free riders cannot receive more than contributing leechers. In contrast, with the old choke algorithm, a fast free rider can monopolize a seed. Third, the resilience in transient phase is improved. Indeed, the initial seed does not favor any leecher. Thus, if a leecher leaves the peer set, it will only remove a small subset of the pieces from the torrent. In contrast, with the old choke algorithm, the initial seed can send most of the pieces to a single leecher. If this leecher leaves the torrent, that will adversely impact the torrent and increase the time in transient state.

In summary, the new choke algorithm in seed state gives the same service to time to each leecher. This new algorithm is a significant improvement over the old one. In particular, whereas the old choke algorithm can be unfair and sensible to free riders, the new choke algorithm is fair and robust to free riders.

V. Related Work

Whereas BitTorrent can be considered as one of the most successful peer-to-peer protocol, there are few studies on it.

Several analytical studies of BitTorrent-like protocols exist [6], [21], [25]. Whereas they provide a good insight into the behavior of such protocols, the assumption of global knowledge limits the scope of their conclusions. Biersack et al. [6] propose an analysis of three content distribution models: a linear chain, a tree, and a forest of trees. They discuss the impact of the number of chunks (what we call pieces) and of the number of simultaneous uploads (what we call the active peer set) for each model. They show that the number of chunks should be large and that the number of simultaneous uploads should be between 3 and 5. Yang et al. [25] study the service capacity of BitTorrent-like protocols. They show that the service capacity increases exponentially at the beginning of the torrent and then scale well with the number of peers. They also present traces obtained from a tracker. Such traces are very different from ours, as they do not allow to study the dynamics of a peer. Both studies presented in [6] and [25] are orthogonal to ours as they do not consider the dynamics induced by the choke and rarest first algorithms. Qiu and Srikant [21] extend the initial work presented in [25] by providing an analytical solution to a fluid model of BitTorrent. Their results show the high efficiency in terms of system capacity utilization of BitTorrent, both in a steady state and in a transient regime. Furthermore, the authors concentrate on a game-theoretical analysis of the choke and rarest first algorithms. However, a major limitation of this analytical model is the assumption of global knowledge of all peers to make the peer selection. Indeed, in a real system,

each peer has only a limited view of the other peers, which is defined by its peer set. As a consequence, a peer cannot find the best suited peers to send data to in all the peers in the torrent (global optimization assumption), but in its own peer set (local and distributed optimization). Also, the authors do not evaluate the rarest first algorithm, but assume a uniform distribution of pieces. Our study is complementary, as it provides an experimental evaluation of algorithms with limited knowledge. In particular, we show that the efficiency on real torrents is close to the one predicted by the models.

Felber et al. [9] compare different peer and piece selection strategies in static scenarios using simulations. Bharambe et al. [5] present a simulation-based study of BitTorrent using a discrete-event simulator that supports up to 5000 peers. The authors concentrate on the evaluation of the BitTorrent performance by looking at the upload capacity of the nodes and at the fairness defined in terms of the volume of data served by each node. They varied various parameters of the simulation as the peer set and active peer set size. They provide important insights into the behavior of BitTorrent. However, they do not evaluate a peer set larger than 15 peers, whereas the real implementation of BitTorrent has a default value of 80 peers. This restriction may have an important impact on the behavior of the protocol as the piece selection strategy is impacted by the peer set size. The validation of a simulator is always hard to perform, and the simulator restrictions may biased the results. Our study provides real word results that can be used to validate simulated scenarios. Moreover, our study is different because we do not modify the default parameters of BitTorrent, but we observed its default behavior on a large variety of real torrents. Finally, we provide new insights into the rarest first piece selection and on the choke algorithm peer selection. In particular, we argue that the choke algorithm in its latest version is fair.

Pouwelse et al. [20] study the file popularity, file availability, download performance, content lifetime and pollution level on a popular BitTorrent tracker site. This work is orthogonal to ours as they do not study the core algorithms of BitTorrent, but rather focus on the contents distributed using BitTorrent and on the users behavior. The work that is the most closely related to our study was done by Izal et al. [14]. In this paper, the authors provide seminal insights into BitTorrent based on data collected from a *tracker* log for a *single* yet popular torrent, even if a sketch of a local vision from a local peer perspective is presented. Their results provide information on peers behavior, and show a correlation between uploaded and downloaded amount of data. Our work differs from [14] in that we provide a thorough measurement-based analysis of the rarest first and choke algorithms. We also study a large variety of torrents, which allows us not to be biased toward a particular type of torrent. Moreover, without pretending to answer all possible questions that arise from a simple yet powerful protocol as BitTorrent, we provide new insights into the rarest first and choke algorithms.

VI. Discussion

In this paper we go beyond the common wisdom that BitTorrent performs well. We have performed a detailed experimental evaluation of the rarest first and choke algorithms on real torrents with varying characteristics in terms of number of leechers, number of seeds, and content sizes. Whereas we do not pretend to have reached completeness, our evaluation gives a reasonable understanding of the behavior of both algorithms on a large variety of real cases.

Our main results are the following.

- The rarest first algorithm guarantees an entropy close to the ideal one. In particular, it prevents the apparition of rare pieces and of the last pieces problem.
- We have found that torrents in a startup phase can have a poor entropy. The duration of this phase depends only on the upload capacity of the source of the content. In particular, the rarest first algorithm is not responsible of the poor entropy during this phase.
- The fairness achieved with a bit level tit-for-tat strategy is not appropriate in the context of peer-to-peer file replication. We have proposed two new fairness criteria in this context.
- The choke algorithm is fair, fosters reciprocation, and is robust to free riders in its latest version.

Our main contribution is to show that on real torrents the rarest first and choke algorithms are enough to have an efficient and viable file replication protocol in the Internet. In particular, we discussed the benefits of the new choke algorithm in seed state. This new algorithm outperforms the old one and should replace it. We also identified two new areas of improvement: the downloading speed of the first blocks, and the duration of the transient phase.

The rarest first algorithm is simple. It does not require global knowledge or important computational resources. Yet, it guarantees a peer availability, for the peer selection, close to the ideal one. We do not see any striking argument in favor of a more complex solution.

We do not claim that the choke algorithm is optimal. The understanding of its equilibrium is an area of future research. However, it achieves a reasonable level of efficiency, and most importantly it guarantees a viable system by fostering reciprocation, preventing free riders to attack the stability of the system, and using the excess capacity. Solutions based on a bit level tit-for-tat are not appropriate.

Our conclusions only hold in the context we explored, i.e., peer-to-peer file replication in the Internet. There are many different contexts where peer-to-peer file replication can be used: small files, small group of peers, dynamic groups in ad-hoc networks, peers with partial connectivity, etc. All these contexts are beyond the scope of this paper, but are interesting areas for future research.

We also identified two areas of improvement. The time to deliver the first blocks of data should be reduced. In the case of large contents, this delivery time will marginally increase the overall download time. But, in the case of small contents, the penalty is significant. Also, the duration of the transient phase should be minimized as the poor entropy may results in a performance penalty. The way to solve these problems is beyond the scope of this study, but is an interesting area of future research.

We believe that this work sheds a new light on two new algorithms that enrich previous content distribution techniques in the Internet. BitTorrent is the only existing peer-to-peer replication protocol that exploits these two promising algorithms in order to improve system capacity utilization. We deem that the understanding of these two algorithms is of fundamental importance for the design of future peer-to-peer content distribution applications.

REFERENCES

[1] http://www.slyck.com.

[2] http://www.bittorrent.com/.

[3] Bittorrent protocol specification v1.0. http://wiki.theory.org/BitTorrentSpecification, June 2005.

[4] R. Bhagwan, S. Savagen, and G. Voelker. Understanding availability. In *International Workshop on Peer-to-Peer Systems*, Berkeley, CA, USA, February 2003.

[5] A. R. Bharambe, C. Herley, and V. N. Padmanabhan. Analysing and improving bittorrent performance. Technical Report MSR-TR-2005-03, Microsoft Research, Microsoft Corporation One Microsoft Way Redmond, WA 98052, USA, February 2005.

[6] E. W. Biersack, P. Rodriguez, and P. Felber. Performance analysis of peer-to-peer networks for file distribution. In *Proc. Fifth International Workshop on Quality of Future Internet Services (QofIS'04)*, Barcelona, Spain, September 2004.

[7] Y. Chawathe, S. Ratnasamy, L. Breslau, and S. Shenker. Making gnutella-like p2p systems scalable. In *Proc. ACM SIGCOMM'03*, Karlsruhe, Germany, August 25-29 2003.

[8] B. Cohen. Incentives build robustness in bittorrent. In *Proc. First Workshop on Economics of Peer-to-Peer Systems*, Berkeley, USA, June 2003.

[9] P. Felber and E. W. Biersack. Self-scaling networks for content distribution. In *Proc. International Workshop on Self-* Properties in Complex Information Systems*, Bertinoro, Italy, May-June 2004.

[10] P. Ganesan and M. Seshadri. On cooperative content distribution and the price of barter. In *IEEE ICDCS'05*, Columbus, Ohio, USA, June 2005.

[11] C. Gkantsidis and P. Rodriguez. Network coding for large scale content distribution. In *Proc. IEEE Infocom'2005*, Miami, USA, March 2005.

[12] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of dht routing geometry on resilience and proximity. In *Proc. ACM SIGCOMM'03*, Karlsruhe, Germany, August 25-29 2003.

[13] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang. Measurements, analysis, and modeling of bittorrent-like systems. In *Proc. ACM IMC'2005*, Berkeley, CA, USA, October 2005.

[14] M. Izal, G. Urvoy-Keller, E. W. Biersack, P. Felber, A. A. Hamra, and L. Garcés-Erice. Dissecting bittorrent: Five months in a torrent's lifetime. In *Proc. PAM'04*, Antibes Juan-les-Pins, France, April 2004.

[15] S. Jun and M. Ahamad. Incentives in bittorrent induce free riding. In *Proc. SIGCOMM'05 Workshops*, Philadelphia, PA, USA, August 2005.

[16] T. Karagiannis, A. Broido, N. Brownlee, and K. C. Claffy. Is p2p dying or just hiding? In *Proc. IEEE Globecom'04*, Dalla, Texas, USA, Nov. 29-Dec. 3 2004.

[17] T. Karagiannis, A. Broido, M. Faloutsos, and K. C. Claffy. Transport layer identification of p2p traffic. In *Proc. ACM IMC'04*, Taormina, Sicily, Italy, October 2004.

[18] D. Kostić, R. Braud, C. Killian, E. Vandekieft, J. W. Anderson, A. C. Snoeren, and A. Vahdat. Maintaining high bandwidth under dynamic network conditions. In *Proc. USENIX'05*, Anaheim, CA, USA, April 2005.

[19] A. Parker. The true picture of peer-to-peer filesharing. http://www.cachelogic.com/, July 2004.

[20] J. A. Pouwelse, P. Garbacki, D. H. J. Epema, and H. J. Sips. The bittorrent p2p file-sharing system: Measurements and analysis. In *Proc. 4th International Workshop on Peer-to-Peer Systems (IPTPS'05)*, Ithaca, New York, USA, February 2005.

[21] D. Qiu and R. Srikant. Modeling and performance analysis of bittorrent-like peer-to-peer networks. In *Proc. ACM SIGCOMM'04*, Portland, Oregon, USA, Aug. 30–Sept. 3 2004.

[22] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. ACM SIGCOMM'01*, San Diego, California, USA, August 27-31 2001.

[23] P. Rodriguez and E. W. Biersack. Dynamic parallel-access to replicated content in the internet. *IEEE/ACM Transactions on Networking*, 10(4), August 2002.

[24] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. ACM SIGCOMM'01*, San Diego, California, USA, August 27-31 2001.

[25] X. Yang and G. de Veciana. Service capacity in peer-to-peer networks. In *Proc. IEEE Infocom'04*, pages 1–11, Hong Kong, China, March 2004.