BitTorrent and fountain codes: friends or foes?

Salvatore Spoto, Rossano Gaeta, Marco Grangetto, Matteo Sereno
Department of Computer Science
University of Turin
Turin, Italy
{*spoto, rossano, grangetto, matteo*}*@di.unito.it*

***Abstract*—BitTorrent is the most popular file sharing protocol on the Internet. It is proved that its performance are near-optimal for generic file distribution when the overlay is in a steady state. The two main BitTorrent strategies, tit-for-tat and local rarest fist, work at best: the former assures reciprocity in downloading and uploading rates between peers and the latter distributes the different file pieces equally among the overlay. Both assure good performance in terms of resource utilization and the practical consequence is that the peers achieve good downloading times. The best network condition for the protocol is a network characterized by roughly fixed arrival rates and no flash crowds. Nevertheless, many research works argue that the performance of the protocol quickly degrades when the peers join and leave at high rates, the network is affected by flash crowds phenomenon and the number of peer that shares the complete file is only a little fraction of the total population. This is the case of many real-world peer-to-peer applications like video-on-demand or live-streaming. In this scenario the introduction of some kind of network coding can mitigate the adverse network behavior and improve the overall performance of the protocol. In this paper we develop a modification of the BitTorrent protocol with the introduction of Luby-Transform codes, that belong to the class of the erasure rateless codes. Using a modified version of GPS (General Purpouse Simulator), we set up simulations that prove how these changes make the original protocol more robust to adverse network conditions and speed up its performance in such situations.**

***Keywords*-bittorrent; network coding; luby transform; LT; peer to peer;**

I. INTRODUCTION

A. Overview

Peer-to-peer are the most popular applications over the Internet [1]. In recent years this kind of applications has gained great popularity, mainly derived from the great diffusion of fast residential Internet connections: a great number of people use them to retrieve and share multimedia contents, generating a significant portion of the total Internet traffic. The number of Internet users is constantly growing following a trend that appears not to stop in the near future. A peer-to-peer system can be suited to support a wide range of applications: historically it was mainly oriented to share and download content for an off-line utilization, while in recent years many other different purposes arise: video-on-demand, live-streaming and distributed portals are few examples. The benefits of using the peer-to-peer paradigm w.r.t. the classica client/server is to shift the load from a central server to the peers' overlay, with many advantages: save server bandwidth, provide the service to many users with the same bandwidth utilization, obtain a more robust system due the error-resilient feature of a distributed overlay. Among all peer-to-peer applications, BitTorrent [2], [3] is the standard de facto for generic file sharing. It is designed to deliver content for an off line utilization: a user must download the whole file before it can be used. In last years the BitTorrent protocol has been subject of considerable research efforts: its strategies have been analyzed in depth, by simulation and in real-world setups, and many works in the literature argue that BitTorrent can reach near-optimal performance [4], [5], [6]. The optimality of the BitTorrent is related with the network conditions: when the network is characterized by constant arrival rates, there are no flash crowds and many peers continue to share the already downloaded file the protocol performs well. The rarest first piece selection and the tit-for-tat, are good enough in most situations. Using the first policy a peer always requests the rarest piece in his neighbourhood, contributing to replicate pieces that are difficult to found, while the latter assure reciprocity between downloading and uploading rates, in fact a peer send data only to his best uploaders. On the other hand, the performance of the protocol quickly degrade when these preconditions are violated, and the overlay is affected by high churn rates and flash crowds. The introduction of a rateless encoding can lighten some major drawbacks of the protocol in such situations. In this paper we design a modification of the BitTorrent protocol with the introduction of coding and we test its performance through simulation. We realize a simulated version of the modified protocol using GPS (General Purpouse Simulator) [16], and compare its performance with the simulated implementation of BitTorrent. From our tests we observe that while the standard protocol continue to perform better in stable network condition when sharing large files, the modified protocol has a significative gain in adverse network conditions with file small/medium size.

The research leading to these results has received funding from the EC Seventh Framework Programme under grant agreement 248036 (COAST).
978-1-4244-6534-7/10/$26.00 ©2010 IEEE

B. Related Works

The optimality of the BitTorrent protocol is analyzed in [5], where it is shown that the strategies that the protocol implements, the rarest first piece selection and the tit-for-tat, are good enough in most situations that can arise in real-world. When the overlay is in a steady state, BitTorrent shows a near-ideal behavior. This occurs most of the times in file-sharing scenarios: many peer are connected to the overlay for the time that is needed to receive the whole file; so when the files are large enough, a peer spends a lot of time connected to the overlay. In addition a peer can continue to share the file also when the download has been completed. It is clear that in this conditions the overlay is quite stable. In other peer-to-peer applications the network behavior can be very different: examples are video-on-demand or the live-streaming cases. In this scenario peers join and leave the network at high rate. In addition only few pieces are available at a time. The rarest first policy is strongly penalized due to the reduced diversity of the available file parts. The BitTorrent protocol is not designed for this kind of applications, and needs some improvements. [5] has demonstrated that in theory an improvement in protocol performance can be achieved using network coding. Many works in related literature try to adapt the BitTorrent protocol for streaming purposes. Previous modifications of the BitTorrent protocol concentrate their efforts on video-on-demand applications. All the works [9], [10], [11], [12] share very similar approaches: the intent is to modify the rarest first policy and the tit-for-tat mechanism in order to make them suitable for video-on-demand applications. [9] and [10] modify the rarest first policy to require the pieces that are necessary in the immediate future for playback, while the papers [11] and [12] extend the tit-for-tat mechanism; in [11] the peers are ranked on forwarding rates, i.e., the capability that a peer has to serve the other peers, in place of standard download and upload rates. Other approaches such as [7], [8] maintain a conventional streaming server and to assist it with the BitTorrent overlay: a request is first sent to the peer-to-peer network then, if the request cannot be satisfied, it is forwarded to the streaming server. Also [8] modifies the rarest first policy in order to give higher priority to the pieces near to the playback point. Tripler [13] is a modification of the protocol proposed for live-streaming. Similar to the previous works, it proposes a modification of the rarest first strategy reducing the downloading set to an interval around the playback time. None of the previous modifications investigates the introduction of network coding at piece level for live-streaming applications. On the other hand, some works try to introduce network coding in peer-to-peer application. The most remarkable is [20] that suggests a protocol based on exclusive use of network coding. The paper proves through simulation that the proposed approach is able to achieve better performance compared to a protocol that is similar to BitTorrent. Some other attempts have been made in the literature trying to integrate network coding in BitTorrent; in [21] a simple protocol is proposed to introduce Luby-Transform codes in BitTorrent, but only few simple scenarios are simulated so the results are not conclusive. In [22], [23] the use of coding is exploited to avoid some scenarios in which the local rarest first strategy fails, and produce as metric of the effectiveness of the proposed protocol the starvation of the peers. In [25] is proposed a protocol for streaming video content based on LT codes. The whole file is encoded and coded blocks are exchanged between peers. A peer must wait for the recover of the original data before relay packets on network. This introduce a delay in the content's spreading.In [26] some preliminary ideas on the distribution of LT coded blocks with application to the P2P video streaming are presented without the design of a complete protocol. Many of the cited works don't compare their performance with the BitTorrent protocol.

In our contribution we first introduce some modifications in BitTorrent to enhance it with a particular class of rateless codes, i.e., the LT codes. Later on, by using a simulation based study, we compare the performance of the modified BitTorrent against a standard version of the protocol. The rest of the paper is organized as follows. In Section II we give a brief description of the protocol, the used terminology and the employed strategies. In Section III we describe the proposed protocol that introduces the LT code, and give a brief description of this kind of encoding. In Section IV the simulation methodology and the used software. In Section V we describe the different scenarios that we set up to prove the effectiveness of the modified protocol, producing experimental results. Section VI draws conclusions and outlines future developments.

II. Overview of BitTorrent

Proposed by Bram Cohen, BitTorrent [3] has the purpose of improving the dissemination of generic content among a network distributing the load on the peer-to-peer overlay, in opposition to the forerunner client-server paradigm where all the cost for the transmission was on a central server. For the sake of clarity, here we summarize the terminology related to the BitTorrent protocol and give an overview of the protocol.

A. Terminology

MetaInfo file: a file that contains information related to a particular document. The most relevant information is the tracker address, the file size and piece size. Usually it has ".torrent" extension

Swarm: the set of peers that cooperate to download a file

Tracker: a central server that stores the list of peers related to a particular torrent. It replies to client's requests with a list of the addresses of the other peers in the swarm.

Seeder: a peer that has completed the download but continues to share the file.
Leecher: a peer that has not yet completed the download.
Piece: the whole file is subdivided into pieces of roughly the same size, usually a piece has dimension of 2^{18} Bytes
Block: every piece is subdivided into blocks. These blocks are exachanged between the clients and usually have dimension of 2^{14} Bytes
Choked and Unchoked: a peer can made a request to another one if this latter considers him in the unchoked state. Peers do not reply to requests from peers in the choked state. Choke and unchoke states depends on download and upload rates.

B. Overview of the protocol

The protocol adopts a multi-part download scheme: the file is divided into several pieces, usually of about 2^{18} Bytes, that are shared among the peers. These exchange blocks of 2^{14} Bytes at time. A BitTorrent system uses two protocols: the *Tracker protocol* and the *peer-wire protocol*. The first is made by HTTP requests and is used when a client needs to connect to the overlay: a request is sent to the tracker that replies with a list of peers that are actually in the swarm. The tracker address and a description of the file properties is stored in a metainfo file that the client has previously downloaded from the web. After a user has got the list, it tries to connect to a random subset of the obtained peers. And starts to use the peer-wire protocol. This protocol consists of many messages, but two are the main strategies employed: a request of pieces based on a local rarest first criterion and a tit-for-tat mechanism. The rarest first strategy assures an even distribution of the file pieces among the overlay forcing a peer to request the rarest pieces in his neighborhood. There are two exceptions to the rarest first strategy: at start up, when a peer has no piece at all, the piece to download is selected at random, and near the end, in the so called *endgame*, the requests for the missing blocks are sent to the entire neighborhood. This is necessary because it has been observed that the last pieces take a long time to download. There are two different tit-for-tat mechanisms for seeders and leechers. A seeder sorts the neighbours by their download rates, unchoking the best ones. To the contrary a leecher unchokes the peers with the higher upload rates. In this way the protocol assures a good match between the download and the upload rates among peers and tries to maximize the bandwidth utilization for data dissemination. The download and upload rates are computed every *rechoke period*. BitTorrent discovers new useful connections with a strategy named *optimistic unchoke*: every three rechoke periods one peer is unchoked at random from the neighborhood. Although quite standard, some parameters of the protocol, such as the peer list size or the length of the periods of *rechoke* and *optimistic unchoke*, can vary in different implementations, but this does not affect the overall performance of the protocol and his strategies.

III. The modified Protocol

In order to introduce the LT codes we have done some modifications to the standard BitTorrent protocol. First of all the pieces are now encoded using LT codes [14] and the generated coded blocks are exchanged between peers. Consequently some modifications to the protocol's strategies must be done. The choice of LT codes has been made due their simplicity and efficiency. Here we provide a brief introduction to LT codes, then we describe the modifications introduced.

A. The Luby-Transform codes

LT codes are a class a *fountain codes* invented by Michal Luby that use simple XOR operations to encode and decode the message. They belong to the class of rateless erasure codes, and they have the property to generate a potentially limitless sequence of encoded symbols, differently from standard erasure codes that have a fixed rate.
The coding process: first the message is divided into k blocks of the same size, then a degree d, $0 < d \leq k$ is chosen for the coded message. Then d blocks are chosen at random with uniform distribution and xored together. This is the data part of the message. At this point the message is sent to the destination. The degree d is generated by a particular distribution, named the Robust Soliton Distribution [14], that is show to be optimal in terms of coding redundancy.
The decoding process: also the decoding process uses XOR operations in order to recovery the original message. It is based on the consideration that for a block $\mathbf{M}$, $\mathbf{M} \oplus \mathbf{M} = 0$, where $\oplus$ is the XOR operator. So when a coded message is xored with a packet of degree 1, i.e. a decoded block, his degree is decreases by one. If this operation is repeated for $d - 1$ times the original message is recovered. Starting from this consideration the decoder processes a message in the subsequent way: if the packet has degree one it is saved as a decoded block and xored with all blocks in which it appears as index, reducing their degree by one; otherwise it is xored with all the blocks that are already decoded and that are present in its list of indexes, reducing its degree. This process continues until the original message is recovered. The decoding process has a certain overhead [14]: the recipient needs to receive a number of packets that exceed in percentage the original size of the message by a certain amount ε. So if k is the original number of non coded blocks, the decoding process succeeds when $k(1+\varepsilon)$ packets are received. In [14] it is shown that LT codes are asymptotically optimal, i.e., the overhead $\varepsilon \to 0$ in the limit of large block sizes $k \to \infty$.

B. Modifications to BitTorrent protocol

In order to evaluate the LT codes' effectiveness, we choose to start from BitTorrent then introduce the proper

modifications. This choice is motivated by many reasons. First of all it is impractical to encode the whole file, specially when it is of large size. Indeed the LT's decoding process requires that all received blocks must be maintained in memory until the original piece is recovered, and this can require a considerable amount of memory for large files. For this reason we choose to maintain the file's piece subdivision and encode the pieces' data. So, similar to BitTorrent, our protocol subdivides the file into pieces, and each one is encoded with LT codes. The original BitTorrent's piece size is of 2^{18} Bytes and the block size is of 2^{14} Bytes. In order to use the LT codes with efficiency the piece is now subdivided into about 1000 blocks. To obtain this ratio the block size is set to 500 Bytes with a subsequent piece size of 2^{19} Bytes, that results in roughly 1048 blocks for piece. This assures that the decoding process is accomplished with an overhead of 10% [15]. LT codes introduces some advantages with respect to the original protocol. In BitTorrent a piece can be shared only when it is completely downloaded. In fact, downloading of a partial piece with the standard protocol would require the introduction of a proper strategy for content reconciliation: a peer must share with others the piece's progress information, avoiding to receive duplicated information. This would result in a signalling overhead.

Instead, with LT codes it is possible to download the same piece from multiple peers. In fact, provided that the transmitting peers use asynchronous random generators for the creation of LT coded packets, every new packet is innovative for the receiver and coded blocks can be sent without information of the piece's progress. More importantly, LT codes allow one to share partially downloaded pieces without content reconciliation. When a piece is partially downloaded it cannot be decoded at all, so the only possible action is forwarding some of the received coded blocks. The block to be forwarded is chosen randomly, so it can happen that some duplicated blocks are collected by the receiving peers. When a peer successfully recovers a piece, new encoded blocks can be generated, avoiding duplicates. In the simulation we made, we observed a reduced number of duplicates received on average. A significant percentage of duplicated packets is experienced only in the early stage of torrent's distribution when the leechers own partial pieces. This transient behavior disappears as soon as the peers decode some pieces, starting to generate new blocks. The introduction of encoding at piece level requires a new policy to select the next piece to download. BitTorrent chooses a piece according to the rarest first strategy, selecting it only among the completely downloaded ones. The possibility to share partially downloaded pieces extends the set of possible parts to download. So we modified the rarest first in order to consider also partial pieces: a peer tries first to download the rarest complete piece in its neighborhood, but when useful pieces are not available, it selects a partial rarest one. In this way we try to minimize the negative impact of duplicates. The tit-for-tat does not require modifications, indeed it performs in the same way in both protocols.
In the following we summarize the modifications of the BitTorrent protocol we made:

- the dimensions of a block is changed from the original 2^{14} Bytes to 500 Bytes for an efficient use of LT codes. Every piece is thus composed of 1048 blocks;
- a chunk is shared to the other peers in the overlay also when it is partially downloaded. Now a peer sends the *HAVE* message, that in the original protocol inform other peers that the source own a piece, as soon as one coded block has been received. In this way a peer can start to share content sooner with respect to the standard BitTorrent.
- the local rarest strategy takes in account also the partially downloaded pieces This increase the availability of downloadable content, and reduce the start-up period.
- a piece can be downloaded from many peer at the same time.

All the other protocol parameters, such as the length of the *rechoke* period or the maximum number of unchoked peers are the same as in the original BitTorrent.

IV. Simulation metodology

To evaluate the performance of the modified protocol we use the GPS [16] simulator. GPS includes a complete implementation of the standard BitTorrent, that is, at the best of our knowledge, the most complete and accurate available. We use this implementation as a starting point to develop and test our changes. Then we generated different scenarios that implement adverse network conditions and compare the performance of both protocols.

A. Simulator description

GPS is a discrete-event simulator developed with the specific task to implement the BitTorrent protocol. The simulation is accurate, with a complete implementations of all the strategies and messages. GPS models the physical underlying network using a transit-stub model [17]. A transit-stub graph represents the topology in a hierarchical way: the peers are attached to stubs that are linked to transit nodes. The internal network of transit nodes represents the router sub-net: a message between two peers travels from a stub domain to another, across a set of transit nodes. Both stub and transit nodes are organized into domains, that are essentially random graphs [18]. The model permits to defines the property of these graphs, generating many different topologies. Usually the capacity of links between transit nodes is defined to be greater than the transit-stub ones. In order to simulate the overhead that derive from the use TCP/IP protocol employed by BitTorrent, GPS implements a macro-model [19] to compute the available bandwidth on a link. In addition to this the simulator define

Table I
DOWNLOADING TIMES IN A COMMON FILE SHARING SCENARIO.

File size	BTd [1]	LTd [2]
1 MBytes	54s	43s
2 MBytes	76s	85s
100 MBytes	1900 s	2100s
200 MBytes	3300 s	4100s

[1] BitTorrent standard protocol download's times [2] Modified BitTorrent protocol download's times

Table II
DOWNLOADING TIMES AND ACHIEVABLE BITRATE OBSERVED ON A SIMPLE TOPOLOGY OF THREE PEERS.

File size	BTd [1]	LTd [2]	BTb[1]	LTb[2]	Gain (%)
1 MBytes	38s	35s	210Kbps	228Kbps	8%
2 MBytes	50s	48s	320Kbps	333Kbps	4%
4 MBytes	77s	75s	415Kbps	426Kbps	2.6%

[1] the columns **BTd** and **BTb** show respectively the download times and achievable bitrate of the standart BitTorrent protocol
[2] the columns **LTd** and **LTb** show respectively the download times and achievable bitrate of the modified protocol that use the LT codes

a flow-model for the link usage: the bandwidth of a link is equally shared between all the active connections on that link, and the effective available bandwidth between two peers corresponds to the bottleneck on the path between the two nodes. The GPS simulator has been validated against a real implementation of BitTorrent, providing good results [16].

B. Extension of the GPS simulator

We made some changes to the simulator in order to define more complex scenarios. By modifying the simulator classes we add the possibility to generate heterogeneous networks, in which the peers can have different connection speeds. In the original simulator, it was not possible to define peer departures, but only joins. We extend GPS adding the possibility for a peer to leave the network overlay at certain instant in time, independently from the progress of its download. Then we implemented the modified protocol introducing the modification described in Section III starting from the original BitTorrent. Finally, we developed the classes to collect and analyze simulation results and statistics.

V. EXPERIMENTAL RESULTS

First we conduct some experiments in a common file sharing scenario: 30 peers join the swarm to download a file; we use in these tests different file size, from 1MByte to 200MByte. The peers do not leave the network after the download completion, but start to act as seeders. In Table I are reported the results of these simulations: the modified protocol has achieved the worst performance, except for 1MByte scenario. This is mainly due to the overhead introduced by the use of LT codes.

We can assume here that rarest first and tit-for-tat are enough to ensure diversity and a fair distribution of the content in the overlay. In this condition LT codes do not introduce any performance improvement. Then we instrumented the simulator to conduct some experiments in scenarios characterized by high rates of peer arrivals and departures, using files with low number of pieces. This conditions are similar to live-streaming, in which the pieces are spread on the overlay as they are generated, and have a short life time. In such kind of applications generic users can join the channel for a short time, and this results in high churn rates. Also flash crowds phenomenon [24] can be an issue for the peer-to-peer overlay. This corresponds, for example, to some particular moments of the channel schedule, as the start or the end of a particular TV program. So a desirable property for a protocol is to be resilient to such kind of events. In all scenarios we simulate also the possibility that BitTorrent shares and spreads a document with only a single piece. This limitation derives from the unavailability of more pieces due the real-time nature of the content distribution. All the results presented are the average of ten simulation runs.

A. A simple topology of three peers

We start our investigation by defining a simple toy topology of three peers as show in Figure 1. In this topology, using the BitTorrent protocol, the leechers can not exchange data between them until a piece is completely downloaded. We want to verify if the modified protocol can achieve better performance by relaying incomplete pieces. The arrows show the flow of data between peers. Peer 1 is the seeder, that replies to requests made from the leechers, peers 2 and 3. We show that in such a simple scenario our modified protocol performs better compared to BitTorrent whether the shared file has a single or multiple chunks. In Table II we report the results that compare the download times and the resulting achievable download rates of the standard BitTorrent protocol and the one modified with LT codes. From Table II we observe that the modified protocol achieves better performance for all the considered file size. The columns **BTd** and **BTb** show respectively the download times and achievable bitrate of the standard BitTorrent protocol while the columns **LTd** and **LTb** show respectively the download times and achievable bitrate of the modified protocol that uses LT codes The column **Gain** report the gain expressed in percentage. There is always a gain using the modified protocol, and it is particular evident for file dimensions of 1 MByte and 2 MBytes. This derive from the possibility to start to download a piece immediately, because the leechers can share data between them even if these latter do not belong to a completed piece.

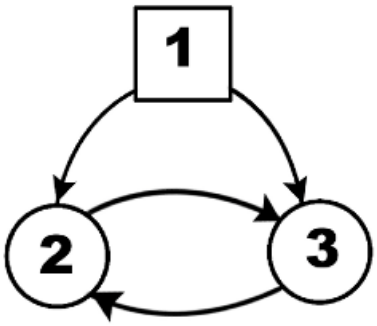

Figure 1. Simple tree peer topology, peer 1 is the seeder while peer 2 and 3 are leecher

B. A more complex scenario

After having verified that in the previous simple scenario the modified protocol has some benefit over the standard BitTorrent, we built a simulation scenario that represents a flash crowd in a network overlay: to this end we generate with GT-ITM [17] a network topology and define over it a network with a 10% of nodes with a bandwidth of 10 Mbps and a 90% of 1 Mbps for a total of 50 nodes. The peers with the larger capacity represent some institutional nodes whereas the slower ones represent standard ADSL connections. Moreover, this scenario is characterized by massive arrivals and departures of peers. These events are scheduled in this way:

- Time 0: a flash crowd of 50 peers occurs
- Time 50: 20 random selected peers leave the network, regardless of the state of the download

Also in this scenario we use a file of dimension of 1 MByte. The results are showed in Table III: the first row compares the performance in the case of a file with a single chunk and the second for a file subdivided into more chunks. Also in this case our protocol performs better, with the downloading times and the resulting achievable bitrate that show a gain of 66% if we consider files with only one piece. The gain decreases to 25% if consider the file subdivide into more pieces. When the file is divided into more parts, these are a dimension of 2^{18} Bytes for the standard protocol and of 2^{19} Bytes for the modified one. This result proves the increased resilience of the codes to flash-crowd. The resulting overlay is more robust to a single massive arrivals and departures of peers. In Figure 2 we report the normalized distributions of the download times. The distribution on the left refers to the modified protocol while the one on the right describes the downloading times of the standard BitTorrent. It is clear that the BitTorrent that uses the LT codes performs better

Table III
DOWNLOADING TIMES AND ACHIEVABLE BITRATE OBSERVED IN A SINGLE FLASH CROWD SCENARIO.

File size	PS[1]	BTd[2]	LTd[3]	BTb[2]	LTb[3]	Gain
1 MByte	S	53.1s	18.1s	151 Kbps	441 Kbps	66%
1 MByte	M	23.9s	17.1s	334 Kbps	467 Kbps	26%

[1] the file subdivision, single (**S**) or multiple (**M**) pieces
[2] the columns **BTd** and **BTb** show respectively the download times and achievable bitrate of the standart BitTorrent protocol
[3] the columns **LTd** and **LTb** show respectively the download times and achievable bitrate of the modified protocol that use the LT codes

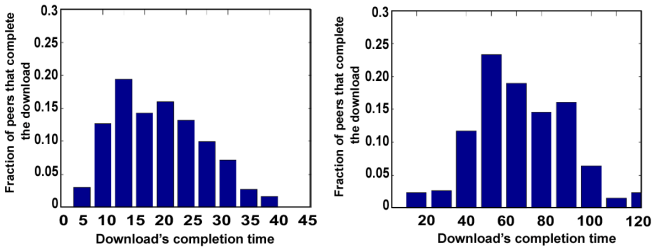

Figure 2. Normalized distributions of the download times: on the left the modified protocol, on the right the original BitTorrent protocol

in both situations, showing shorter download times. Indeed the modified protocol's time distribution is centered around 20 seconds, and show values from 5 to 40 seconds. Instead standard BitTorrent donwload's times are between 10 and 120 seconds, with a peak aroud 60 seconds.

C. Flash crowd scenario

Now we generate a more complex scenario that presents multiple flash crowd phenomenon and massive departure of peer to test the overlay robustness. To describe a flash-crowd we use three parameters: a, b, c are that are the exponents of three exponential distribution, from which the values are random sampled to model a flash-crowd. Values derived from the distribution with parameter a model the times between two consecutive flash-crowds, b is used to model the intensity, expressed as the number of the peers that join the network in a reduced interval time, and the third parameter describe the flash crowd duration. Short values sampled from the distribution with parameter c describe a more critical flash-crowd, because the peers join the network in a reduced amount of times. The total number of peers that join the network is distributed linearly among this time interval. Massive departures of peers are modeled in the same way, but using three different parameter: d, e, f In addition in the simulated scenarios arrival and departures of peers at fixed rates are simulated independently from flash-crowds. This rates are modeled using two additional exponential distribution that use parameters g and h. Table IV summarize all the used parameters.

We conducted some experiments building complex scenarios that cripple the BitTorrent performance. The experi-

Table IV
PARAMETER USED FOR GENERATE FLASH CROWD SCENARIO

Parameter	Description
a	Time between a flash-crowd and the next
b	Intensity of the flash-crowd
c	Flash crowd duration
d	Time between a massive departure of peers and the next
e	Intensity of the massive departure of peers
f	Flash crowd duration for massive departure
g	Inter-arrival times parameter distribution
h	Leave times parameter distribution

Table V
DOWNLOADING TIMES AND ACHIEVABLE BITRATE OBSERVED IN A SCENARIO AFFECTED BY MULTIPLE FLASH CRWOD AND MASSIVE DEPARTURE OF PEERS.

File size	BTd[1]	LTd[2]	BTb[1]	LTb[2]	Gain
4 MBytes	65 s	56 s	492 Kbps	571 Kbps	14%
8 MBytes	145 s	129 s	441 Kbps	496 Kbps	12%

[1] the columns **BTd** and **BTb** show respectively the download times and achievable bitrate of the standart BitTorrent protocol
[2] the columns **LTd** and **LTb** show respectively the download times and achievable bitrate of the modified protocol that use the LT codes

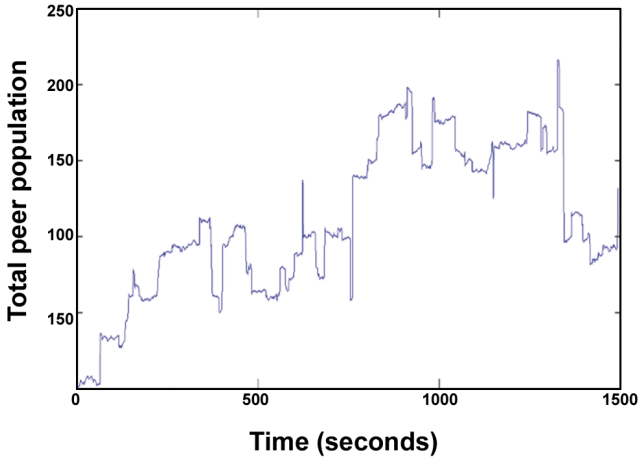

Figure 3. The evolution of overlay population in a sample scenario generated with parameters $a = 0.025$, $b = 0.05$, $c = 0.5$, $d = 0.025$, $e = 0.05$, $f = 0.5$

ment has the subsequent parameters: $a = 0.025$, $b = 0.05$, $c = 0.5$, $d = 0.025$, $e = 0.05$, $f = 0.5$, $e = 0.2$, $f = 0.2$, with a total simulation length of 1500 seconds. The results are averaged over 20 runs of 2 different randomly generated scenarios. In Figure 3 one scenario generated with the above parameters is shown. It is affected by many flash-crowds and massive departure of peers, as expected. The simulated network has 1000 nodes, with 10% of 10 Mbps of bandwidth and 90% represent ADSL connections with 1 Mbps of bandwidth. In this case we conduct two simulations using file of 4 MBytes and 8 MBytes, that are subdivided into multple pieces. In this way the original BitTorrent protocol can use his standard local rarest first strategy and tit-for-tat at his best. Table V reports the results of such simulations compared to BitTorrent in the same scenario and network conditions: there is a clear gain of 14% for the file of 4 MBytes and 12% for dimensions of 8 MBytes. Also in this case there is a clear advantage that derives from the introduction of LT codes, that can spread content with limited file size and in adverse network condition in a faster way. The gain mostly derives from the capacity to share partially downloaded pieces. This advantages, in case of file of small size and in high dynamic network conditions, exceed the drawback derived from the overhead introduced from the use of the LT codes.

VI. CONCLUSIONS

In this paper we proposed a novel modifications of the BitTorrent protocol by introducing the LT encoding at the piece level, with some modification to the standard protocol strategies, in particular the local rarest first. Then we proved by simulations that better performance can be achieved considering file of small size and in adverse network conditions. This conditions represent real-world applications like live-streaming, in which only a little fraction of the total file is available at time and peer joins and leave the network at high rates. In such situation our proposed protocol yields a gain that is above the 10% in all simulated scenarios, in spite of the decoding overhead that the LT codes introduce. In conclusion, network coding turns to be beneficial for P2P file sharing applications. We are currently working on an extention of the presented protocol. While here we develop a modification of BitTorrent, our intention is to implement a completly new peer-to-peer protocol, suitable for streaming applications. With a real-world implementation of this latter, possibly deployed on PlanetLab, we can confirm the benefits that we point out with simulation, and investigate new favorable scenarios.

REFERENCES

[1] T. Karagiannis, A. Broido, M. Faloutsos, and K. C. Claffy, *Transport layer identification of p2p traffic*, in Proc. ACM IMC04, Taormina, Sicily, Italy, October 2004.

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

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

[4] A. Legout, G. Urvoy-Keller, P. Michiardi, *Understanding bittorrent: An experimental perspective. Technical report*, CNRS : FRE2660 EURECOM

[5] A. Legout, G. Urvoy-Keller, P. Michiardi, *Rarest First and Choke Algorithms Are Enough*, IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement, Rio de Janeriro, Brazil, 2006.

[6] A. Al-Hamra, A. Legout, and C. Barakat, *Understanding the Properties of the BitTorrent Overlay*, CoRR abs/0707.1820: (2007).

[7] C. Dana, D. Li, D. Harrison, C. Chuah, *BASS: BitTorrent Assisted Streaming System for Video-on-Demand*, Multimedia Signal Processing, 2005 IEEE 7th Workshop on (2005)

[8] B. Liu, Y. Cui, B. Chang, B. Gotow, Y. Xue, *BitTube: case study of a web-based peer-assisted video on demand system,* , Multimedia, 2008. ISM 2008. Tenth IEEE International Symposium on In Multimedia, 2008

[9] A. Vlavianos, M. Iliiofotou, M. Faloutsos, *BiToS: Enhancing BitTorrent for Supporting Streaming Applications*, In 9th IEEE Global Internet Symposium 2006 (April 2006)

[10] D. Erman, K. De Vogeleer, A. Popescu, *On piece selection for streaming bittorrent*, Fifth Swedish National Computer Networking Worshop (SNCNW2008)

[11] J. J. D. Mol, J. A. Pouwelse; M. Meulpolder; D. H. J. Epema; H. J. Sips, *Give-to-Get: Free-riding-resilient Video-on-Demand in P2P Systems*, Multimedia Computing and Networking 2008

[12] P. Shah, J.F. Paris, *Peer-to-Peer Multimedia Streaming Using BitTorrent*, Performance, Computing, and Communications Conference, 2007. IPCCC 2007

[13] J. J. D. Mol, A. Bakker, J.A. Pouwelse, D.H.J. Epema, and H.J. Sips, *The Design and Deployment of a BitTorrent Live Video Streaming Solution*, The IEEE Int'l Symposium on Multimedia 2009, December 2009.

[14] M. Luby, *LT codes*, Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on (2002).

[15] E.Hyytia, T. Tirronen, J. Virtamo *Optimizing the Degree Distribution of LT Codes*, in RESIM 2006, 6th International Workshop on Rare Event Simulation, 2006, Bamberg, Germany

[16] W. Yang, N. Abu-Ghazaleh, *GPS: A General Peer-to-Peer Simulator and its Use for Modeling BitTorrent*, Proceedings of 13th Annual Meeting of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '05) , Sept 27-29, 2005, Atlanta.

[17] K. Calvert, M. Doar, E. W. Zegura. *Modeling Internet Topology*, IEEE Communications Magazine, June 1997.

[18] P. Erdős, and A. Rényi, *On Random Graphs* in Publ. Math. Debrecen 6, p. 290297 (1959).

[19] M. Mathis, J. Semke, and J. Mahdavi. *The macroscopic behavior of the tcp congestion avoidance algorithm*, SIGCOMM Comput. Commun. Rev., 27(3):6782, 1997.

[20] C. Gkantsidis, P. Rodriguez *Network Coding for Large Scale Content Distribution*, INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE, Vol. 4 (2005), pp. 2235-2245 vol. 4.

[21] D. Bickson, R. Borer *The BitCod Client: A BitTorrent Clone using Network Coding* In P2P '07: Proceedings of the Seventh IEEE International Conference on Peer-to-Peer Computing (2007), pp. 231-232

[22] H. Luan and D. H.K. Tsang *A Simulation Study of Block Management in BitTorrent* In Proceedings of the 1st international conference on Scalable information systems (InfoScale 2006) (01 June 2006)

[23] B. Dodson, D. Petkanics, Z. Ives, S. Khanna, *Combining coding and block schemes for p2p transimissions*

[24] B. Li, Y. Keung, S. Xie, F. Liu, Y. Sun, and H. Yin, *An Empirical Study of Flash Crowd Dynamics in a P2P-based Live Video* Streaming System, in Proc. of IEEE Globecom, Nov. 2008.

[25] C. Wu, B. Li, Senior Member *rStream: Resilient and Optimal Peer-to-Peer Streaming with Rateless Codes* IEEE Transactions on Parallel and Distributed Systems (2008).

[26] M. Grangetto, R. Gaeta, and M. Sereno, *Rateless codes network coding for simple and efficient p2p video streaming* in Proc. of IEEE International Conference on Multimedia and Expo, July 2009.