Annales Univ. Sci. Budapest., Sect. Comp. 37 (2012) 65–80

TOWARDS A FASTER BITTORRENT

Ádám Agócs, Zoltán Ács,
Attila Balaton and Tamás Lukovszki
(Budapest, Hungary)

Communicated by András Benczúr

(Received January 15, 2012; revised March 3, 2012;
accepted March 12, 2012)

Abstract. BitTorrent is the most popular peer-to-peer system for file sharing. In this protocol, the file is divided into pieces and the clients upload and download the pieces to each other. The tit-for-tat rule of the protocol enforces the cooperation between selfish peers. In the original BitTorrent protocol, sometimes the neighbors of a peer have no piece that the peer does not already have, which inhibits the download. Rare pieces can cause long waiting times. An elegant way of solving these problems is the extension of the original protocol by source coding. This coding method increases the diversity of the pieces in the network which accelerates tit-for-tat piece exchange and leads to faster downloads.
We propose a novel deterministic source coding method. The main advantages of our method compared to state of the art random coding methods are the reduced traffic overhead and the guarantee of decodability of the original file after downloading d different coded pieces, where d is the number of pieces of the file. We analyze the deterministic method theoretically. The theoretical results are backed up by simulations.

Key words and phrases: BitTorrent, peer-to-peer, source coding.
2010 Mathematics Subject Classification: 15A03, 12E30, 05A10.
1998 CR Categories and Descriptors: G.1.3, F.2.1, C.2.1
The Research is supported by the European Union and co-financed by the European Social Fund (grant agreement no. TÁMOP 4.2.1./B-09/1/KMR-2010-0003).

1. Introductiobn

In the past few years, lots of peer-to-peer file sharing systems have been proposed and implemented. The most successful among them is unambiguously BitTorrent. In this protocol each file is split into data pieces. A peer can share individual pieces even if it has not finished downloading the original file. BitTorrent protocol uses tit-for-tat to enforce collaboration of nodes. When a node finished downloading the file, it will become a seeder. These seeders are the most useful nodes of the network: they use their whole bandwidth to upload pieces. The nodes communicate with each other by sending `have` and `request` messages. The `have` message is an identifier of a piece, which is sent by a node to the neighborhood when the node finished downloading the piece. The `request` message is also an identifier of a piece, which is sent to an uploader to request the piece. These messages can inflict a huge communication overhead, but this overhead can be handled if piece diversity is high enough.

1.1. Motivation

The problem of rare pieces is one of the most significant defects of the original BitTorrent protocol. Imagine the sutiation when a group of pieces is unavailable in a whole neighborhood. The nodes of the neighborhood are able to exchange their pieces, but the rest of the pieces are out of the neighborhood. In the protocol, this problem is handled by nodes downloading rare pieces first. This solution works in several cases, but there is a better way to solve the problem of rare pieces. If we are able to increase piece diversity enough, then rare pieces disappear. An obvious way to increase piece diversity is applying source coding by every seeder.

1.2. Our work

We compared three types of source coding:

- Random linear source coding
- Random binary source coding
- Deterministic source coding

Random linear source coding is a well-known method, random binary source coding was proposed by Locher et al. [13], while deterministic source coding

is our novel solution, which will be presented and analysed in detail. In every method, the distributed file is split into d pieces. Every piece of the file is represented as an m-dimensional vector: $x_1, x_2, \ldots x_d \in (GF(p^k))^m$, where p is a prime number and $GF(p^k)$ is a finite field. In the simulations we used $p = 2^{31} - 1$ and $k = 1$.

1.2.1. Random source coding

Random source coding is a special type of random network coding. In this case the coding coefficients are chosen random from $GF(p^k)$. If the encoding coefficients are chosen uniformly at random, the probability that the coding matrix will be invertible depends on the size of the field. A result for network coding from Jaggi et al. presented in [10] They showed that if the network is modeled as a directed graph $G = (V, E)$ and the field size is at least $|E|/\delta$, the encoding will be invertible at any given receiver with probability at least $1 - \delta$. This means that a large network requires a large field size. Since the coding mechanism is the same that we considered as random source coding, the results can be applied to source coding as well.

1.2.2. Random binary source coding

Random binary source coding was proposed by Loecher et al. [13]. They used special coding vectors, where the coding coefficients are 0 or 1. Suppose the file is divided into d pieces. Binary source coding makes linear combinations where there are precisely $m \ll d$ ones in every coding vector, and the rest of the coding coefficients are zeros.

This technique has several interesting properties. There is no need for weights, which simplifies the system. The size of the coding vectors is reduced, since it is a simple bitmap. Finally, due to the rare coding coefficients, the matrix can be inverted quickly.

1.3. Our results

The novel deterministic source coding has several advantages compared to random coding methods. The encoding will be invertible at any receiver with the probability one and the communication overhead can be reduced in the network due to the special coding vectors. With source coding we can achieve a high piece diversity in the network. This high diversity allow us to introduce a new technique, when the `have` and `request` messages can be sent empty. This new technique can be considered a *push* protocol where the downloader is not allowed to determine the piece it wants to download.

1.4. Outline of the paper

In the next section, we present some related work on the analysis of BitTorrent and source and network coding. In Section 3, we present our novel source coding mechanism. The analysis of traffic overhead is presented in Section 4, and in Section 5, we consider the case when high diversity makes the exchange of file identifiers unnecessary. We made simulations to compare the coding methods. The details and the results can be found in Section 6.

2. Related work

A lot of theoretical and practical work investigated BitTorrent networks. Arthur and Panigrahy [2] modelled BitTorrent as a graph of nodes. In one time step, each node can upload one piece to a neighboring node and each node can download one piece from a neighboring node. They proved lower bounds for routing times using several routing policies and graph models. Another graph theoretic analysis of peer-to-peer systems can be found in [14], where Loguinov et al. proposed de-Bruijn graphs for construct topology of a peer-to-peer network.

Related works on source and network coding is very diversified. The first appearance of network coding can be found in [1], where Ahlswede et al. proposed network coding and it has been proved that the information rate from the source to a set of nodes can reach the minimum of the individual max-flow bounds through coding. Li et al. [12] showed further that it is sufficient to use only random linear functions to achieve optimal transmission rate. Jain et al. [11] proposed a scheme for building peer-to-peer overlay networks for broadcasting using network coding.

In [13] Loecher et al. proposed a novel source coding mechanism, which turned out a success in BitTorrent networks. They used special coding vectors, where the coding coefficients were 0 or 1.

3. Deterministic source coding

Now we describe our deterministic source coding method. This method guarantees that the encoding will be invertible at any receiver with the prob-

ability one. Furthermore, compared to random coding methods, the communication overhead can be reduced in the network due to the special coding vectors.

The coding algorithm generates $K > d$ coded pieces, where d is the number of the original pieces. The number of generated coded pieces strongly impacts the efficiency of the method. The choice of the K will be discussed later. Let $a_1, a_2, \ldots, a_K \in GF(p)$ where $a_i \neq a_j$ if $i \neq j$. These numbers are the *coding coefficients*. The coding algorithm generates K different linear combinations from the original vectors:

$$y_i = 1x_1 + a_i x_2 + a_i^2 x_3 + \ldots + a_i^{d-1} x_d$$

where $1 \leq i \leq K$. In the next theorem, we will show that any d of these K vectors will be independent.

Claim 3.1. *We can restore the original file from arbitrary d different y_i vectors.*

Proof. After receiving d different coded pieces, we obtain the following coding matrix:

$$A = \begin{pmatrix} 1 & a_{i_1} & a_{i_1}^2 & \ldots & a_{i_1}^{d-1} \\ 1 & a_{i_2} & a_{i_2}^2 & \ldots & a_{i_3}^{d-1} \\ 1 & a_{i_3} & a_{i_3}^2 & \ldots & a_{i_3}^{d-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & a_{i_d} & a_{i_d}^2 & \ldots & a_{i_d}^{d-1} \end{pmatrix},$$

which is a Vandermonde matrix over GF(p). It is a well-known property ([15] Chapter 4 Lemma 17) that its determinant is

$$\det(A) = \prod_{j<\ell} (a_{i_\ell} - a_{i_j}) \tag{3.1}$$

Since every a_i is different, the determinant cannot be 0. Now assume that we get d different linear combinations denoted by $y_{i(1)}, y_{i(2)}, \ldots, y_{i(d)}$, where

$$y_{i(k)} = 1x_1 + a_{i(k)} x_2 + a_{i(k)}^2 x_3 + \ldots + a_{i(k)}^{d-1} x_d$$

for every $1 \leq k \leq d$, which gives us d different vector equations. Now consider the first coordinates of these equations. This is a linear equation system with the following coefficients:

$$y_{i(1)_1} = 1x_{11} + a_{i(1)}x_{21} + a_{i(1)}^2 x_{31} + \ldots + a_{i(1)}^{d-1} x_{d1}$$

$$y_{i(2)_1} = 1x_{11} + a_{i(2)}x_{21} + a_{i(2)}^2 x_{31} + \ldots + a_{i(2)}^{d-1} x_{d1}$$

$$\vdots$$

$$y_{i(d)_1} = 1x_{11} + a_{i(d)}x_{21} + a_{i(d)}^2 x_{31} + \ldots + a_{i(d)}^{d-1} x_{d1},$$

which can be written in the following form:

$$\begin{pmatrix} y_{i(1)_1} \\ y_{i(2)_1} \\ y_{i(3)_1} \\ \vdots \\ y_{i(d)_1} \end{pmatrix} = \begin{pmatrix} 1 & a_{i_1} & a_{i_1}^2 & \ldots & a_{i_1}^{d-1} \\ 1 & a_{i_2} & a_{i_2}^2 & \ldots & a_{i_3}^{d-1} \\ 1 & a_{i_3} & a_{i_3}^2 & \ldots & a_{i_3}^{d-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & a_{i_d} & a_{i_d}^2 & \ldots & a_{i_d}^{d-1} \end{pmatrix} \begin{pmatrix} x_{11} \\ x_{21} \\ x_{31} \\ \vdots \\ x_{d1} \end{pmatrix}.$$

According to (3.1), the matrix is invertible. Hence, the solution exists and unique. Thus, the first coordinate of every piece is decoded. Applying this method for the remaining $m-1$ coordinates we can decode every piece. ■

4. Traffic overhead

In this section we compare the traffic overhead of the different coding techniques. Traffic overhead means the sent data in the network which is not directly the distributed file. There are three main categories of data which inflict the traffic overhead:

- The additional data for decoding. This can be the whole coding vector or a unique identifier of the coding vector.
- The `have` message which is sent to the neighborhood after a node finished to download a piece.
- The `request` message which is sent to a neighbor to request a piece.

Due to special coding vectors, the deterministic method produced the less traffic overhead. The tables show the amount of transmitted data when the following coding techniques were used:

- Deterministic source coding
- Random source coding
- Random binary source coding

file size	256K	512K	1M	2M	4M
4K pieces	258.7K	517.4K	1.01M	2.02M	4.04M
16K pieces	256.7K	512.4K	1.002M	2.004M	4.008M
64K pieces	256.1K	512.2K	1M	2M	4M

Table 1. The amount of transmitted data with deterministic coding

file size	256K	512K	1M	2M	4M
4K pieces	420.4K	1.14M	3.57M	12.27M	45.04M
16K pieces	266.2K	553K	1.16M	2.64M	6.57M
64K pieces	256.6K	514.6K	1.01M	2.04M	4.16M

Table 2. The amount of transmitted data with random coding

We used 32-bit numbers for coding vectors. The graph of the network is an Erdős-Rényi random graph and the average degree was set to 8.27 (simulated result). We compared the downloaded data by one node, when the size of the original file and the size of the pieces were varied.

file size	256K	512K	1M	2M	4M
4K pieces	261.1K	532.5K	1.08M	2.32M	5.28M
16K pieces	256.3K	513.3K	1.01M	2.02M	4.08M
64K pieces	256K	512K	1M	2.001M	4.004M

Table 3. The amount of transmitted data with binary random coding

It is straightforward from the tables that huge traffic overhead can occur in several cases. For example, consider the case when a 4M size file is distributed, the size of the pieces is 4K and we used random coding. In this case the communication overhead is more then 10 times the size of the file (Table 2). Most of the extra data come from the `have` and `request` messages. This observation inspired us to consider the cases when the `have` and `request` messages are sent

empty. The theoretical value of the whole transmitted data is presented in Table 4 and 5 for these cases. It is easy to see that this reducing gave a much better traffic overhead. In the next section we show that large file diversity can make `have` and `request` messages unnecessary.

file size	**256K**	**512K**	**1M**	**2M**	**4M**
4K pieces	256.3K	512.5K	1.001M	2.002M	4.004M
16K pieces	256.1K	512.1K	1M	2M	4M
64K pieces	256K	512K	1M	2M	4M

Table 4. The amount of transmitted data with deterministic coding with empty `have` and `request` messages

file size	**256K**	**512K**	**1M**	**2M**	**4M**
4K pieces	272K	576K	1.25M	3M	8M
16K pieces	257K	516K	1.016M	2.063M	4.25M
64K pieces	256.1K	512.25K	1.001M	2.004M	4.016M

Table 5. The amount of transmitted data with random coding with empty `have` and `request` messages

5. Reducing traffic overhead

Discussing the tables above it is easy to see that traffic overhead can reach a high amount in several cases. Since the coding vector or a unique identifier of the coding vector is necessary for the decoding, the only way to save on traffic overhead is to reduce the size of the `have` and `request` messages. In this section we will show theoretically that the `have` and `request` messages are not necessary if the file diversity is large enough.

Consider the case when the client does not have information about the pieces of its neighbors. This means that the `have` and `request` messages are empty. A node downloads d arbitrary pieces and checks if these pieces are different after all d pieces are downloaded. The peer is able to restore the original file from any d different pieces. Hence, the main question is the number of coded pieces that must be downloaded to get d different coded pieces.

We denote by K the number of different coded pieces. In our analysis we assume, that the coded pieces are distributed uniformly in the network, i.e. the probability of choosing a certain coded piece uniformly at random is $1/K$. We used the standard ω and o notations to describe bounds on asymptotic growth rates. The first theorem states that if $K = \omega(d^2)$, it is enough to download d pieces. Then the probability that these will be different is asymptotically 1, if $d \to \infty$.

Theorem 5.1.

(i) *Let $K = \omega(d^2)$ and download exactly d coded pieces. The probability that the original file can be restored from these pieces is asymptotically 1 when $d \to \infty$.*

(ii) *If $K = o(d^2)$, then this probability is asymptotically 0 when $d \to \infty$.*

Proof. We download d coded pieces among K, independently, uniformly at random. Denote $P(d)$ the probability that we download d different pieces:

$$P(d) = \frac{\binom{K}{d} d!}{K^d}. \tag{5.1}$$

In (5.1) the numerator $\binom{K}{d} d!$ is the number of favorable cases, i.e. choosing d different coded pieces among K. The denominator is the number of all cases.

$$P(d) = \frac{\binom{K}{d} d!}{K^d} = \frac{\frac{K!}{(K-d)!}}{K^d} = \frac{K(K-1)\dots(K-d+1)}{K^d}.$$

For an upper bound on $P(d)$, consider the inequality between the arithmetic and geometrical means for $K, (K-1), \dots, (K-d+1)$:

$$\sqrt[d]{\prod_{i=0}^{d-1}(K-i)} < \frac{\sum_{i=0}^{d-1}(K-i)}{d} = K - \frac{d-1}{2}. \tag{5.2}$$

Using (5.2) we got the following upper bound for $P(d)$:

$$P(d) < \frac{(K - \frac{d-1}{2})^d}{K^d} = \left(1 - \frac{d-1}{2K}\right)^d, \tag{5.3}$$

which is asymptotically $e^{-\frac{(d-1)d}{2K}}$.

Now we show a lower bound on $P(d)$. Instead of the the inequality between the arithmetic and geometrical means, underestimate the product with the

lowest member. Thus we got

$$P(d) = \frac{\prod_{i=0}^{d-1}(K-i)}{K^d} > \frac{(K-d+1)^d}{K^d} = \left(1 - \frac{d-1}{K}\right)^d, \tag{5.4}$$

which is asymptotically $e^{-\frac{(d-1)d}{K}}$.

Summarizing, we have

$$e^{-\frac{(d-1)d}{K}} < P(d) < e^{-\frac{(d-1)d}{2K}}. \tag{5.5}$$

Now using the $K = \omega(d^2)$ and the lower bound on $P(d)$, the proof of (i) is done:

$$\lim_{d\to\infty} P(d) \geq \lim_{d\to\infty} e^{-\frac{(d-1)d}{\omega(d^2)}} = \lim e^{-o(1)} = 1. \tag{5.6}$$

The proof of (ii) is obtained by the upper bound $P(d) < e^{-\frac{(d-1)d}{2K}}$. Then the right side of the inequality goes to 0 if $K = o(d^2)$. ■

Theorem 5.1 (i) states that downloading arbitrary d pieces are enough if $K = \omega(d^2)$. Theorem 5.1 (ii) says that if $K = o(d^2)$, then d pieces will not be enough with probability which goes to 1 if d increases. In fact, a stronger claim also can be proved. If $K = o(d^2)$, then slightly more pieces than d still will not be enough. If the number of downloaded coded pieces is $d + s$, where s is a constant which is independent from d, then the probability that the decoding is possible still goes to 0 if $d \to \infty$.

Theorem 5.2. *Let $K = o(d^2)$ and assume that we download $d + s$ coded pieces, where s is a given constant. The probability of the original file can be decoded from these $d + s$ pieces is asymptotically* 0.

Proof. We will show that for every $0 \leq i \leq s$ the probability that there are $d + i$ different among the downloaded $d + s$ goes to 0. Denote this probability by $P(d + i)$, which is overestimated in the following way:

$$P(d+i) < \frac{\binom{K}{d+i}\frac{(d+s)!}{(s-i)!}(d+i)^{s-i}}{K^{d+s}}. \tag{5.7}$$

For the estimation (5.7), we choose the $d+i$ different coded pieces, then consider the first occurrences and distribute the remaining $s - i$ coded pieces. This is obviously an upper bound because some cases counted several times; if a piece arrives repeatedly it is counted only once. Continuing the estimation:

$$\frac{\binom{K}{d+i}\frac{(d+s)!}{(s-i)!}(d+i)^{s-i}}{K^{d+s}} = \frac{\prod_{j=0}^{d+i-1}(K-j)(d+s)!(d+i)^{s-i}}{K^{d+i}(d+i)!K^{s-i}(s-i)!}.$$

Split the product into two pieces and overestimate both of them. The first part can be overestimated by the following way:

$$\frac{\prod_{j=0}^{d+i-1}(K-j)}{K^{d+i}} < \frac{(K-\frac{d+i-1}{2})^{d+i}}{K^{d+i}} < \left(1-\frac{d+i-1}{2K}\right)^{d+i}.$$

And the second part:

$$\frac{(d+s)^{s-i}(d+i)^{s-i}}{K^{s-i}(s-i)!} < \frac{(d+s)^{2(s-i)}}{K^{s-i}(s-i)!} < \frac{\left(\frac{(d+s)^2}{K}\right)^{s-i}}{(s-i)!}.$$

In the estimations we used again the inequality between the arithmetic and geometric means and overestimated $\frac{(d+s)!}{(d+i)!}$ and $(d+i)^{s-i}$ by $(d+s)^{s-i}$. Summarizing the two estimations below we got the following asymptotic result:

$$\begin{aligned}\lim_{d\to\infty} P(d+i) &= \lim_{d\to\infty}\left(1-\frac{d+i-1}{2K}\right)^{d+i}\frac{\left(\frac{(d+s)^2}{K}\right)^{s-i}}{(s-i)!} = \\ &= e^{-\frac{(d+i-1)(d+i)}{2K}} e^{(s-i)\ln\frac{(d+s)^2}{K}}\frac{1}{(s-i)!} = \\ &= \frac{1}{(s-i)!} e^{(s-i)\ln\frac{(d+s)^2}{K}-\frac{(d+i-1)(d+i)}{2K}},\end{aligned}$$

which goes to 0 if

$$\lim_{d\to\infty}\left((s-i)\ln\frac{(d+s)^2}{K}-\frac{(d+i-1)(d+i)}{2K}\right) = -\infty. \tag{5.8}$$

Since $K = o(d^2)$,

$$\lim_{d\to\infty}\left(\frac{(d+i-1)(d+i)}{2K}\right) = +\infty. \tag{5.9}$$

By (5.8) and (5.9) the following condition will be enough to $\lim_{d\to\infty}(P(d+i)) = 0$

$$(s-i)\ln\frac{(d+s)^2}{K} < \frac{1}{2}\frac{(d+i-1)(d+i)}{2K},$$

which is equivalent to

$$c\ln\frac{d^2}{K} < \frac{d^2}{K} \tag{5.10}$$

with an appropriate c constant. It stands from equation (5.9), thus we got that $\lim_{d\to\infty} P(d+i) = 0$ for every i. Now denote by $P(\geq d)$ the probability of downloading at least d different pieces. According to the previous results:

$$\lim_{d\to\infty}\big(P(\geq d)\big) = \lim_{d\to\infty}\Big(\sum_{i=0}^{s} P(d+i)\Big) = \sum_{i=0}^{s}\lim_{d\to\infty}\big(P(d+i)\big) = 0$$

since s is constant, which is independent of d. ■

6. Simulation results

In this section we investigate our source coding techniques in BitTorrent protocol to compare it with another methods such as random source coding and binary random source coding. The simulation software is written in PeerSim [16]. We extended the implementation of the BitTorrent protocol for Peersim by Frioli and Pedrolli.

In our simulations we have 100 peers, one of them is a tracker. The tracker assigns 5 to 10 random neighbors to a new peer. The bandwidth of links which create connection between two different nodes can be 512Kb, 1Mb and 4Mb. At the beginning of the simulation we have only one seeder, which has the whole file. The file consists of 128 pieces. This seeder creates source coded pieces. If a peer receives 128 linearly independent pieces it decodes them to reproduce the original file and becomes a new seeder of the network. The tracker and seeders do not leave the network before all peers finish the download. In the simulations we neglected the time of checking linear independence in the case of random coding, i.e. we assumed that it can be done in zero time. Note that in the case of deterministic coding we only have to check whether the coefficients defining the coding vectors are distinct. We have repeated our simulations 100 times.

6.1. Comparison of source coding methods in BitTorrent protocol

First we have measured the finish times of the peers. Figure 1 shows the finish times of the peers. The x-axis shows the number of peers that finished the download and the y-axis shows the corresponding time. By using deterministic and binary random coding, the nodes finish the downloads faster than in the case of using random coding. The deterministic coding is slightly faster than the binary random coding. The reason of this is the lower traffic overhead, which has been analysed in Section 4.

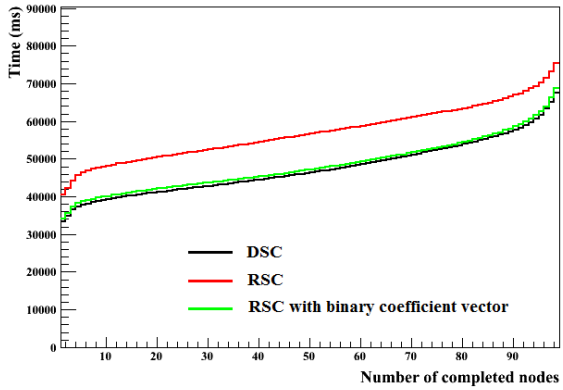

Figure 1. Finish time of the nodes

Figure 2 shows the number of pieces which are in the network, i.e. the number of pieces downloaded or being downloaded. It can be seen that deterministic source coding and binary source coding give better results because of traffic overhead.

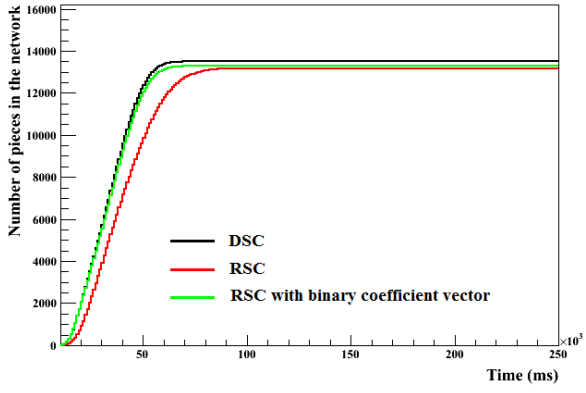

Figure 2. Number of pieces in the network

6.2. Downloading random blocks

In Section 5 we have analysed this "blind" downloading method when peers download random pieces from the neighbors, i.e. the peers send nothing as `have` and `request` messages but not the coding vectors in them. In Section 5 we have shown that in the case of appropriately high diversity of pieces, the probability of downloading the same piece more than once would be very small.

Thus, the traffic overhead could be reduced significantly. Unfortunately, the described simulation settings result in a low piece diversity. The event that a peer downloads the same piece more than once appears quite frequently. We compared the deterministic source coding and the random source coding in this scenario. Figure 3 shows the finish times of the peers. The simulation results show that the performance of the deterministic source coding is better than that of the random source coding also in this situation. Figure 4 presents the number of pieces which are in the network.

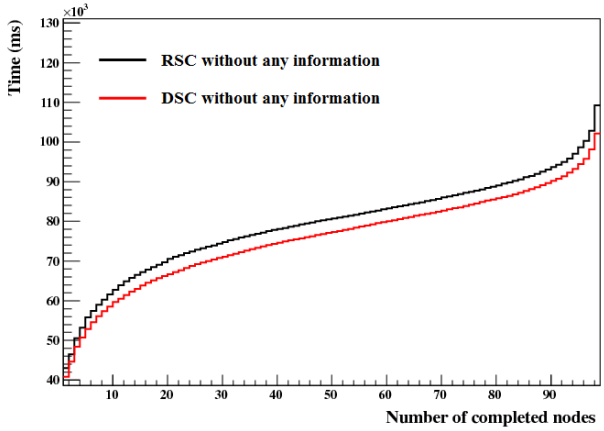

Figure 3. Finish times of the nodes

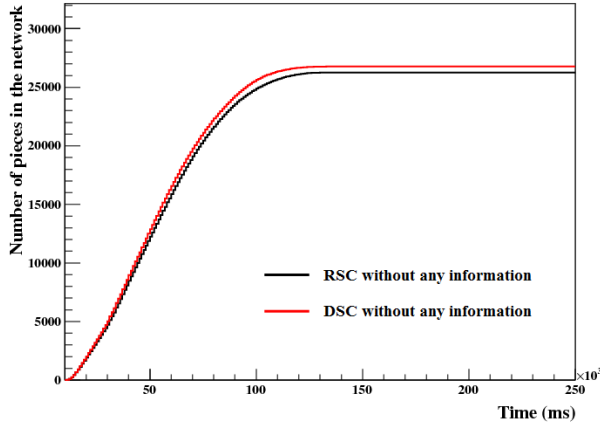

Figure 4. Number of pieces in the network

7. Summary and future work

We presented a deterministic source coding method for distributing files in peer-to-peer networks. This method is an alternative to the random source coding. The main advantages of our deterministic coding are (i) that the coding will always be invertible after receiving d different coded pieces and (ii) a significantly lower communication overhead, which results from the special coding vectors that also must be known (and transmitted) for decoding the coded pieces. Instead of the whole vector only one coefficient of the vector must be sent with a coded piece. The BitTorrent protocol can be extended relatively easily by our deterministic source coding method.

We also considered a case where the peers download a random piece from the neighbor and they can check if the pieces are different after they download them. Using the simplifying assumption that the coded pieces are distributed uniformly in the network, we have shown that (i) if there are $K = \omega(d)$ different coded pieces, the probability that a peer can reconstruct the original file after downloading d randomly chosen coded pieces is asymptotically 1 when $d \to \to \infty$. (ii) If $K = o(d)$, then this probability is asymptotically 0 even if a peer downloads a constant number of additional pieces.

We compared our deterministic source coding method with the random coding also by simulations. The results of the simulations show that by using our deterministic coding the peers finish the downloads faster than by using random coding or binary random coding.

Future work includes more exchausive simulations and the extension of our deterministic source coding algorithm to general network coding scenario, where each node of the network is allowed to generate coded pieces.

References

[1] **Ahlswede, R., N. Cai, S. Li and R. Yeung,** Network information flow, *IEEE Transactions on Information Theory*, **46(4)** (2000), 1204–1216.

[2] **Arthur, D.and R. Panigrahy,** Analyzing BitTorrent and related peer-to-peer networks, *SODA(2006)* 961–969, 2006.

[3] **Balaton, A., T. Lukovszki and Á. Agócs,** A new deterministic source coding method in peer-to-peer systems, in: *Proc. 12th IEEE International Symposium on Computational Intelligence and Informatics (CINTI)*, 2011, pp. 403–408.

[4] **Chou, P., Y. Wu and K. Jain,** Practical network coding, in: *Proc. Allerton Conference on Communication, Control, and Computing*, 2003.

[5] **Cohen B.,** Incentives build robustness in bittorrent, *1st Workshop on Economics of Peer-to-Peer Systems*, 2003.

[6] **Frioli, F. and M. Pedrolli,** (2008) A BitTorrent module for Peersim, *Peersim website*
`http://peersim.sourceforge.net/code/bittorrent.tar.gz`
[accessed 29/02/12]

[7] **Gkantsidis, C. and P. Rodriguez,** Network coding for large scale content distribution, *Proc. IEEE INFOCOM*, 2005, 2235–2245.

[8] **Harvey, N.A., D.R. Karger and K. Murota,** Deterministic network coding by matrix completion, in: *Proc. 16th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2005, pp. 489–498.

[9] **Ho, T., M. Medard, R. Koetter, D.R. Karger, M. Effros and B.L. Jun Shi,** A random linear network coding approach to multicast, *IEEE Transactions on Information Theory*, **52(10)** (2006), 4413–4430.

[10] **Jaggi, S., P. Sanders, P.A. Chou, M. Effros, S. Egner, K. Jain and L. Tolhuizen,** Polynomial time algorithms for multicast network code construction, *IEEE Transactions on Information Theory*, **51(6)** (2005), 1973–1982.

[11] **Jain, K., L. Lovász and P.A. Chou,** Building scalable and robust peer-to-peer overlay networks for broadcasting using network coding, in: *Proc. 24th ACM Symposium on Principles of Distributed Computing (PODC)*, 2005, pp. 51–59, 2005.

[12] **Li, S-Y.R., R.W. Yeung and N. Cai,** Linear network coding, *IEEE Transactions on Information Theory*, **49(2)** (2003), 371–381.

[13] **Locher, T., S. Schmid and R. Wattenhofer,** Rescuing tit-for-tat with source coding, in: *7th IEEE International Conference on Peer-to-Peer Computing (P2P)*, 2007.

[14] **Loguinov, D., A. Kumar, V. Rai and S. Ganesh,** Graph-theoretic analysis of structured peer-to-peer systems: routing distances and fault resilience, *SIGCOMM '03*, 2003, 395–406.

[15] **McWilliams, F.J. and N.J. Sloane,** *The Theory of Error-correcting Codes*, North-Holland, Amsterdam, 1977.

[16] **Montresor, A. and M. Jelasity,** (2009) PeerSim: A scalable P2P simulator, in: *Proc. of the 9th Int. Conference on Peer-to-Peer* (*P2P'09*), Seattle, WA, September 2009, pp. 99–100.

Á. Agócs, Z. Ács, A. Balaton and T. Lukovszki
Faculty of Informatics
Eötvös Loránd University
H-1117 Budapest, Pázmány P. sétány 1/C
Hungary
`agocs_a@inf.elte.hu`, `acszolta@inf.elte.hu`
`balcsi4@inf.elte.hu`, `lukovszki@inf.elte.hu`