A practical Random Network Coding scheme for data distribution on peer-to-peer networks using rateless codes

Valerio Bioglio[a], Marco Grangetto[b], Rossano Gaeta[b], Matteo Sereno[b]

[a]*Dipartimento di Elettronica e Telecomunicazioni*
Politecnico di Torino
C.so Duca Degli Abruzzi, 24 - 10129 Torino - Italy
Ph.: +39 011 0904230 - Fax: +39 011 0904099
[b]*Dipartimento di Informatica*
Università degli Studi di Torino
C.so Svizzera, 185 - 10149 Torino - Italy
Ph.: +39-011-6706711 - Fax: +39-011-751603

Abstract

In this paper we propose a practical Random Network Coding (RNC) scheme for data distribution in a peer-to-peer (P2P) overlay network. The use of RNC incurs a significant computational cost that, till present, has limited its deployment in practical applications. In this study it is shown that RNC complexity can be lowered by using Luby Transform (LT) codes to pre-encode the data and by letting intermediate nodes use RNC in a low-order Galois Field, i.e. GF(2). Moreover, we exploit a recently proposed variant of the Gaussian Elimination algorithm (OFG) to improve further both the creation of random combinations for RNC and the final decoding of the content.

Our analysis is based on both analytical modeling and simulations over P2P overlay networks generated from random graphs and real snapshots of the PPLive streaming application. The results point out that using LT codes and RNC in GF(2) one is able to significantly improve the overall

[☆]This work was partially supported by Università degli Studi di Torino and Compagnia di San Paolo under project AMALFI (ORTO119W8J).

Email addresses: valerio.bioglio@polito.it (Valerio Bioglio), grangetto@di.unito.it (Marco Grangetto), rossano@di.unito.it (Rossano Gaeta), matteo@di.unito.it (Matteo Sereno)

performance in terms of both delay and bandwidth utilization at a reasonable computational cost. Finally, the RNC strategies we propose do not require any prior knowledge of the overlay network topology thus making them very general.

Keywords: LT codes, Random Network Coding, Peer-to-Peer.

1. Introduction

Several recent results pointed out that the use of coding techniques increase the efficiency of content distribution applications such as reliable distribution of bulk data, application level multicast, P2P streaming applications [1, 2, 3, 4] or efficient broadcasting in ad hoc wireless networks [5, 6], just to mention a few. Most of the cited results have been catalyzed by the seminal promises of network coding [7, 8], where nodes in the network are allowed to combine information packets instead of simply forward them. In particular, in Random network Coding (RNC) [9] each peer transmits linear combinations of incoming packets, where the coefficients are chosen randomly over some finite field. The deployment of network coding at application level, e.g., in the field of P2P file sharing or video streaming, has been limited primarily because of the bewared added computational cost due to linear coding. Nowadays, such complexity issue must be carefully reconsidered; indeed, a novel class of erasure codes called *rateless codes* [10, 11], designed for application level coding, is turning out as a practical tool for efficient coded data dissemination.

Rateless codes [10, 11] are a family of erasure codes where the rate, i.e. the number of coded and transmitted symbols, can be adjusted on the fly. In other words, as opposed to standard channel codes, characterized by a rate, which is selected in the design phase a rateless encoder can generate an arbitrary number of coded symbols. The approach used to transmit such codes is called Digital Fountain (DF), since the transmitter can be viewed as a fountain emitting coded symbols till all the interested receivers (the sinks) have received the number of symbols required for successful decoding. It is evident that such paradigm is well suited for wireless broadcasting applications where a single source is serving many recipients experiencing different channel conditions. From the point of view of the added computational cost, rateless codes require to perform simple xor operations among the original packets on the encoder side, and to solve a sparse linear system in a Galois

field of order 2 on the decoder side.

In most of the deployed P2P applications a peer concurrently downloads contents from multiple peers and uploads towards multiple peers. Although this improves the bandwidth utilization and allows to counteract network dynamics, content reconciliation policies are required. However, the potential advantage of coding is the simplification of the content reconciliation problem, since every piece of coded information is equally useful regardless of the peer who has contributed it. If one uses coding, in principle, there is no limit to the number of uniquely coded packets generated from the original set of packets, thus relaxing the content reconciliation issue. Therefore, a simpler push approach can be adopted to let propagate the information. Nonetheless, coding poses novel issues, as well. In particular the information flow has to be divided into coding blocks; the computational complexity needed to encode end decode such blocks could make this approach unfeasible in practice.

For these motivations, in this paper we propose a practical Random Network Coding strategy in GF(2) using rateless codes. Our goal is to design a more efficient exploitation of DF principle for both fast propagation of the information and low communication overhead due to the transmission of an excessive amount of redundant coded packets. We use Luby Transform (LT) codes [10] to pre-encode the source information. In general, when a node that has not decoded yet has the possibility to send a packet, it sends a linear combination of the previous received packets, calculated in GF(2).

The major contributions of the paper are the following:

- analytical modeling of the strategies that one can use to forward and combine coded packets;
- exploitation of the previous analytical results to design novel approaches aiming at filling the performance gap between LT based and general RNC solutions; in particular, we show that throttling the upload bandwidth of the peers in the initial phase one significantly decreases the number of duplicated packets;
- besides the computational savings yielded by the usage of LT codes we show that the previously proposed OFG decoding algorithm [12] can be modified to obtain a linear combination of the received coded packets at the cost of a single XOR operation;

- the results section reports an extensive set of experiments comparing the proposed approaches versus the literature in terms of both delivery times and computational costs.

The major achievement of the paper is that using RNC based on LT codes one can spread the information across a random network maintaining a low number of duplicated packets and keeping the encoding and decoding complexity limited.

The paper is organized as follows. In Sect. 2 some background on rateless codes is recalled. In Sect. 3 the related research results are briefly recalled and compared to our proposal. Sect. 4 presents models whose analysis supports the definition of several relaying strategies for coded packets while Sect. 5 describes the system we consider. In Sect. 6 the simulator used to test the proposed relaying strategies is described. Experimental results, performance evaluation and analysis are reported in Sect. 7. In Sect. 8 our conclusions and future research directions are drawn.

2. Rateless codes background

Luby Transform (LT) codes [10] are the first class of efficient rateless erasure codes that achieve optimality as the data length increases. LT are random block codes where the original data are divided into k information packets x_i, $i = 0, \ldots, k-1$ (the code performance does not depend on the size of the packet). Each coded symbol y_j, $j \geq 0$, is a packet constituted by a random combination, i.e., the exclusive-or, of the information packets: $y_j = \sum_{i=0}^{k-1} g_{i,j} x_i$. The key element in LT code design is the so called degree distribution, where the degree is defined as the number of source packets combined to obtain a given y_j. In [10] the Robust Soliton Distribution (RSD) $\tau_{\delta,c}(i)$, is proposed for the selection of the degree $i = 1, \ldots, k$. In the RSD c is a suitable positive constant and δ is the allowed failure probability at the decoder. In [10] it is demonstrated that the decoder fails to recover the data with probability at most δ from a set of $K = k + O(\sqrt{k} \cdot \ln^2(k/\delta))$ coded symbols, which means that successful decoding is attained with $K = k(1+\epsilon)$ with $\lim_{k\to\infty} \epsilon = 0$, i.e., the code is asymptotically optimal.

2.1. Decoding algorithms

The classical LT decoder is the BP algorithm [10, 13]. A node waits for a coded symbol of degree 1 that turns out in the decoding of a source

symbol; this latter is used to lower the degree of all previously received packets containing such symbol. The adoption of RSD guarantees that this process can be iterated and converges to the decoding of all symbols as far as the number of received packets is large enough.

In the most general case, given a set of n coded packets the decoder can attempt reconstruction of the information message by applying Gaussian elimination (GE) on the equivalent code generator matrix $\mathbf{G} = \{\mathbf{g}'_1, \ldots, \mathbf{g}'_{n-1}\}$, with row vectors $\mathbf{g}_j = \{g_{0,j}, \ldots, g_{k-1,j}\}$. In [12] the On the Fly Gaussian Elimination algorithm (OFG) has been proposed as an improvement over the classical GE algorithm, that makes the $\mathbf{G}$ triangularization process incremental. Indeed, OFG builds a triangular decoding matrix $\mathbf{G}$ by exploiting every received packet. Upon receiving a coded packet y_j OFG finds its position in $\mathbf{G}$ by possibly combining it, i.e., xoring, with already filled rows of $\mathbf{G}$. It follows that at any time, any full row of $\mathbf{G}$ contains the combination of possibly many received coded packets. This feature will be exploited in Sect. 5.2 to define an effective strategy for relaying coded packets. A packet that is inserted in $\mathbf{G}$ is called a *useful* packet; on the contrary, a packet that cannot be inserted in $\mathbf{G}$ represents a linear combination of the previously received packets and is thus discarded. These packets are called *duplicated* packets.

Shortly, BP requires less xor and swap operations than OFG for large values of k but it relies on the assumption that the degree of coded packets follows the RSD. The BP decoder spends most of its computational efforts when enough coded packets have been received as opposed to OFG that spreads the computations over all the packets reception thus turning out to be faster in practical contexts. Moreover, OFG does not force to use the RSD distribution in the coding phase to be able to decode the original information. It means that the distribution employed to encode the original message can be different from RSD. Furthermore, as shown in [12], OFG has lower overhead than BP, i.e., the number of duplicated packets is much smaller for fixed k.

3. Comparison to Related works

In [1] an in depth analysis of the reconciliation issues in conjunction with packet encoding is shown. A set of reconciliation algorithms trading off accuracy and complexity is proposed. [1] designs a family of reconciliation techniques, also tested in a real test-bed in [2], through which the peers

participating to the overlay attempt to coordinate the content downloading by means of both original packets coding and recoding of already coded data.

The approach presented in [14] follows a complementary approach, by avoiding the need for reconciliation, based on the optimal design of a distributed rateless code, i.e., coded packets are guaranteed by construction to be independent and equally useful. Nevertheless the solution proposed in [14] is limited to the case of a single network topology with a common relay node, that can be generalized only assuming perfect knowledge of the overlay connections. This latter assumption is clearly unfeasible in a real dynamic overlay. Moreover, the optimization algorithm complexity increases with the size and the number of the connections in the overlay. In [15] another optimal coding approach is proposed. This solution is based on re-encoding, where several coding stages are cascaded while moving from hop to hop. Besides being asymptotically optimal, recoding comes at a significant computational expense in intermediate nodes, and may lead to excessive coding overhead in a real scenario with several hops and limited code block length.

A simpler approach able to cancel the reconciliation phase is used in [4], where P2P streaming application adopting rateless coding and optimal peer selection is presented. In [4] the DF approach is applied on every peer-to-peer connection and peers are not allowed to propagate coded information before the complete decoding of a block. Therefore, at the expense of an additional delay, every peer waits for complete decoding of a block of original packets and then starts sending independent LT encoded packets. This strategy has low decoding and decoding complexity due the performance of LT codes, but the time needed to spread information in a network is large because the nodes have to wait to decode a whole block before start relaying. The proposed push approach is coupled with an optimal overlay formation strategy aiming at constructing high quality streaming topologies so that end-to-end latencies are minimized.

In [9] the authors propose to combine packets to achieve better performance: this strategy is called Random Network Coding (RNC). In fact, the seeder creates a new packet by a linear combination, in GF(q), of its input symbols, where q is a sufficiently large integer. The other nodes in the network create new encoded packets by a linear combination in GF(q) of the previously received packets. In the paper it is proven that q should be larger than the number of the peers in the network, that makes this strategy hardly applicable in a real scenario due its computational complexity, as stated in [16]. To face this problem, in [16] a peer waits to relay packets until it has

received a certain number of packets. Using this parameter it is possible to decrease the information spreading time, even if the performance of RNC is still considered poor; in fact, the computational complexity of encoding and decoding in GF(q), that increases with the increasing of q, is still too costly.

The lesson we learnt from [16] is that RNC is not feasible in a real scenario because all the encoding and decoding operations are made in a large-order Galois Field. Therefore, we propose strategies to increase the performance of RNC in low-order Galois Fields. We use LT codes as a pre-code for the source for two motivations: they have a low decoding complexity, and they are generated in GF(2), i.e. with the lowest possible value of q. Similarly, we use OFG decoding algorithm to decode received packets because it permits to decrease the time needed to create new packets thorough the combination of previously received packets (i.e. what in RNC is called encoding time). This is possible because, using OFG, the rows of the decoding matrix are random linear combinations of received packets; when a node needs a linear combination of packets to encode a new packet it can to use these rows rather than calculate a new linear combination. In addition, OFG yields a low overhead ϵ even if the degree distribution is not RSD, thus allowing one to exploit linear combinations of LT encoded packets within an RNC delivery approach.

Another lesson we learnt from [16] is that the performance of RNC improves with the number of received packets: when a node has received few packets, the encoding process (i.e. to create new packet by a linear combination of the previously received packets) is less useful. In opposition to [16], that proposes to block the recoding process, we propose to limit the number of linear combinations injected by a node depending on the number of the received coded packets. This idea is carried out by an analytical model, proposed in Sect. 4, to evaluate the effect of throttling the speed used by peers to saturate the available upload bandwidth.

Some preliminary ideas applied to the case of P2P video streaming have been already presented in [17]. In this work we considered a strategy where nodes, as opposed to [4], start relaying coded packet before complete decoding of a LT block, showing that lower spreading time is obtained. A coded packet can be forwarded only once to a single destination avoiding the need of reconciliation at the receiving peer. This simple approach proved useful in reducing delays and does not result in duplicated packets circulating in an overlay without loops. To cope with this issue an ad-hoc heuristics has been devised to prune loops. Both in [17, 4] the possibility of combining packets

is not considered: indeed, these studies rely on the standard BP decoder as opposed to the OFG technique used in this work, which enables the design of novel relaying solutions.

4. Modeling of Relaying Strategies

In this section we develop two analytical models to support the design of efficient relaying strategies of coded packets between any two peers in the overlay network. We first quantify the effect of saturating the upload bandwidth of nodes by randomly selecting a set of coded packets to be relayed. We successively prove that making linear combinations of the received packets, before forwarding them to the neighbors, yields a higher probability to provide useful packets to the receiving nodes, i.e. that RNC can be profitably exploited in GF(2) as well.

We consider a distributed application whose components have organized in a peer-to-peer overlay network $\mathcal{T}$. We make no hypothesis on how $\mathcal{T}$ is formed therefore multiple paths between pair of peers and cycles may be allowed. There is a single peer that holds valuable information for all the others (the original source). At startup all other peers are interested in retrieving the information and cooperate to obtain it. Every peer stores the coded packets that turn out to be useful in the buffer OB. By useful coded packet we mean a received packet that is not linear dependent on the previously received ones. Each peer is allowed to combine and forward packets from its buffer OB. Peers are characterized by their upload (B_u) and download (B_d) bandwidth expressed in *bps*. In the sequel we also need to express the peer bandwidths in *pps* (packets per second) for a given packet size; in this case we denote the peer bandwidths as N_u and N_d. Each peer is also characterized by the number of its neighbors from which information is downloaded (z_d) and the number of neighbors to which information is uploaded (z_u).

4.1. Saturating the upload bandwidth: delay vs overhead trade-off

To develop our model we consider a pair of neighbor peers A and B. Time is slotted and peer A randomly selects packets from its output buffer OB and sends them to peer B. Let $d(t)$ be the size of OB at peer A at time t (for $t = 0, \ldots$), and $s(t)$ the number of packets selected by A for B (at time t). In the following we denote by $P_{\text{use}}(t, q)$ the probability that in the time interval $[0, t]$ A has forwarded q *useful* coded packets to B, and by

$I(s(t), d(t), n, r)$ the probability to send n useful packets at time t given that B has already received r packets during the time interval $[0, t-1]$. By using these definitions we can write the following recurrence

$$P_{\text{use}}(t, q) = \begin{cases} I(s(t), d(t), q, 0) & \text{if } t = 1 \\ \sum_{i=0}^{q} P_{\text{use}}(t-1, q-i) \cdot I(s(t), d(t), i, q-i) & \text{if } t > 1. \end{cases}$$

We can write the probability $P_{\text{dup}}(t, u)$ that A has forwarded u duplicate packets, i.e. not useful for B, in the interval $[0, t]$ as

$$P_{\text{dup}}(t, u) = P_{\text{use}}(t, s_{tot}(t) - u),$$

where $s_{tot}(t) = \sum_{i=0}^{t} s(i)$ is the total number of packet extraction in the interval $[0, t]$.

At time t peer A may select $s(t)$ packets with or without replacement. In the former case we can derive that

$$I(s(t), d(t), n, r) = \binom{d(t) - r}{n} \cdot \sum_{h=0}^{r} \binom{r}{h} f(s(t), n + h, d(t)), \tag{1}$$

where $f(s(t), b, d(t)) = \frac{b! \left\{ {s(t) \atop b} \right\}}{d(t)^{s(t)}}$ if $s(t) \geq b$ and 0 otherwise. We use $\left\{ {s(t) \atop b} \right\}$ to denote the Stirling number of the second kind [18]. If we view packets in OB of A as urns and extractions as balls then the expression for $I(s(t), d(t), n, 0)$ is actually the probability distribution of the number of urns that contain at least one ball in a classical occupancy model [19].

In the case without replacement, the expression of $I(s(t), d(t), n, r)$ reduces to a hypergeometric probability distribution:

$$I(s(t), d(t), n, r) = \frac{\binom{d(t)-r}{n} \cdot \binom{r}{s(t)-n}}{\binom{d(t)}{s(t)}}. \tag{2}$$

In the following we test the proposed model with some numerical examples. To show the trade-off between delay and overhead we set $N_u = 1024$ and $z_u = 32$, which amounts to set a maximum available bandwidth from A to B of $\frac{N_u}{z_u} = 32$ pps. Furthermore, we make the simple assumption that useful packets of A increase linearly with time as $d(t) = x \cdot t$ for some x. The number of packets selected by A at each time slot is limited by $s(t) = \min(\frac{x \cdot t}{z_u}, s_{max})$, where $1 \leq s_{max} \leq \frac{N_u}{z_u}$. In other words the number

of packets forwarded by A is limited by the number of packets in OB and by a design parameter s_{max} which is upper bounded by the upload capacity. We consider original data coded using $k = 100$ so that A and B must, on average, collect $K = 107$ coded packets before decoding as experimented in Sect. 7; in Fig. 1 we show the time to complete the collection of K packets $t_{end} = \min\{t : \sum_{d=K}^{\infty} P_{\text{use}}(t, d) \geq 0.99\}$ for $x = 1$ (Fig. 1(a)) and $x = 64$ (Fig. 1(b)). Fig. 1 also reports the minimum feasible collection time defined as $t_{min} = \lceil \frac{K}{\min(x, s_{max})} \rceil$.

In Fig. 1 we also show the fraction of duplicated packets from A to B defined as

$$\overline{dup}(t_{end}) = \frac{\sum_{u=0}^{s_{tot}(t_{end})-1} u \cdot P_{dup}(t_{end}, u)}{s_{tot}(t_{end})}$$

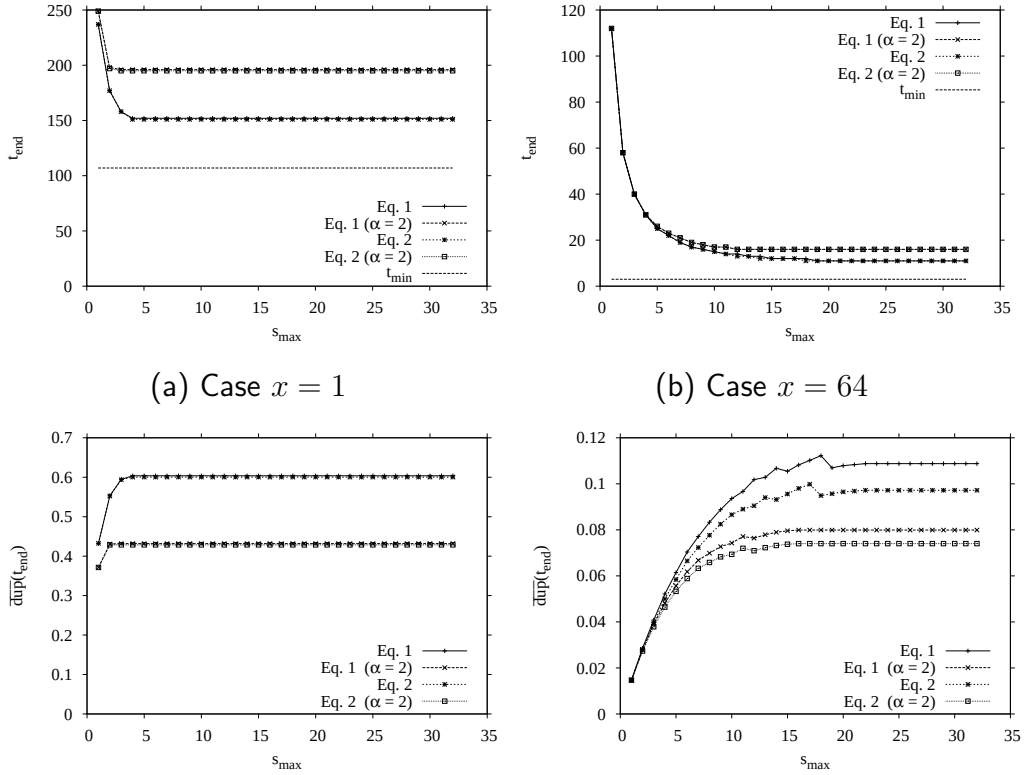
for $x = 1$ (Fig. 1(c)) and $x = 64$ (Fig. 1(d)). It can be noted that models defined in Eqs. 1 and 2 yield overlapping results for $x = 1$ while we note a slight increase in the values of $\overline{dup}(t_{end})$ for selection with replacement in the case of a quickly growing size of the buffer of peer A. As a consequence, in the following only selection without replacement will be considered. We easily observe that when we increase the maximum allowed upload bandwidth we manage to reduce the time for B to collect K packets but the price that is paid is an increasing fraction of duplicated packets received by B. A simple countermeasure to reduce the value of $\overline{dup}(t_{end})$ (while increasing the value of t_{end}) is to throttle the speed of A by setting $s(t) = \min(\frac{d(t)}{\alpha \cdot z_u}, s_{max})$ for a given $\alpha > 1$. The parameter α is used to lower the number of forwarded packets in the initial phase when A is filling its OB.

In Fig. 1 we also report the values obtained by setting $\alpha = 2$. Results clearly show that throttling the upload is crucial in reducing the amount of duplicated packets.

4.2. Combining packets

We now argue that if A combines packets selected from OB before sending them to B then the probability of sending a useful packet increases, i.e. that RNC is useful even if it is made in GF(2). In fact, we want to show that if node A has useful packets for B, then combining packets reduces the probability to send duplicates.

Assume peer A owns h linear independent packets, of which $h_d < h$ are linear dependent on the packets owned by B: this means that each of these



(a) Case $x = 1$ (b) Case $x = 64$

(c) Case $x = 1$ (d) Case $x = 64$

Figure 1: Time to collect K packets for $x = 1$ (a) and $x = 64$ (b) and fraction of duplicated packets from A to B for $x = 1$ (c) and $x = 64$ (d).

h_d packets would be a duplicate if A sends it to B. Call $\mathcal{H}_d$ the linear space generated by the packets of A linear dependent on the packets of B. Peers generally receive packets from different peers, thus A does not know which packets are in $\mathcal{H}_d$. If A sends a randomly chosen packet p to B, the probability p_d that B receives a duplicate is $p_d = P(p \in \mathcal{H}_d) = \frac{h_d}{h}$. If A draws r different packets $p_1, \ldots, p_r$ and xors them obtaining a new packet $p_s = \sum_{i=1}^{r} p_i$, such packet is useless if and only if all combined packets are in $\mathcal{H}_d$. In the case $r \leq h_d$ the probability that B receives a duplicate turns to be $p_d^c = P(p_s \in \mathcal{H}_d) = P(p_1, \ldots, p_r \in \mathcal{H}_d) = \frac{h_d}{h} \prod_{i=1}^{r-1} \frac{h_d - i}{h - i} < p_d$. Otherwise, if $r > h_d$ then $\exists i \leq r : p_i \notin \mathcal{H}_d$ and $p_d^c = 0$. Therefore we have that $p_d^c < p_d \, \forall r$.

This means that to combine packets decreases the probability to send duplicated packets in the network.

5. System description

In this section we describe the behavior of the various peers in the network and the relaying strategies they can follow.

5.1. State of the peers

Each peer in $\mathcal{T}$ can be in one of the following states:

- WAIT: the peer is waiting for the reception of the first coded packet. As soon as the peer receives the first packet it changes its state to DECODER.
- OFF: the peer is not cooperating to obtain or distribute the data. This state is assumed when the peer and all its neighbors have already received and decoded all the k packets.
- SEEDER: the peer has already received and decoded the k information packets but some of its neighbors are still in the DECODER state. In this case, the peer generates new LT coded packets, saturating the upload towards its neighbors. As soon as all its neighbors have decoded the original information, the peer changes its state to OFF.
- DECODER: the peer has not received enough information data to decode the original information. The relaying strategy followed by peers in state DECODER will be discussed in Section 5.2. As soon as the peer receives enough packets to decode the information data, the peer signals such event to all its seeding nodes so as to stop them from pushing more coded packets and changes its state to SEEDER.

At the beginning, the source state is set to SEEDER state while all the other peers are set to WAIT state. All peers in the SEEDER state encode the original data and send it to their neighbors in the DECODER state using the RSD. All peers in the DECODER state run the OFG decoding algorithm and progressively construct their generator matrix G, based on the generating equations of the received coded packets. At the same time, these peers insert only their *useful* packets in an output buffer OB from which packets are selected for relaying.

5.2. Protocol description

The lesson we learnt from Sects. 4.1 and 4.2 is that throttling peers in DECODER state and combining packets before relaying to neighbors in DECODER state are effective strategies to reduce the amount of duplicated packets while retaining the capability of spreading received packets as soon as possible. This allows us to define and compare several strategies for the relaying of coded packets while in the DECODER state:

- Store and Recover (SR): a peer does not forward any of the received coded packets that are used to recover the k original blocks. This means that a peer starts to forward packets only when it switches to the SEEDER state. This is the strategy used in rStream [4].

- Relay (RE): at every transmission opportunity, a peer selects a packet in OB and forwards it. Such packet is deleted from OB so to relay it only once. The procedure is repeated until OB turns empty or the upload capacity is saturated. This strategy is used in [17].

- Random Relay (RR): a peer at every transmission opportunity randomly draws from OB enough packets to fully use its upload capacity and sends them to its neighbors.

- Random Relay with Combinations (RRC): on every transmission opportunity the peer randomly draws from OB enough packets to fully use its upload capacity, it xors them with a randomly chosen row of the decoding matrix $\mathbf{G}$ and sends them to its neighbors. As recalled in Sect. 2.1 this amounts to combine the selected packet with a set of previously received packets at the cost of a single packet xor operation.

The aim of the RR and RRC strategies is to send as much information as possible: the high utilization of upload bandwidth allows us to reduce the information data spreading time. As already seen in Sect. 4, these strategies may be too aggressive, i.e. they could fill the network with too many duplicate packets. For this reason we consider two variants of previous strategies, namely TRR and TRRC, where the upload bandwidth of RR and RRC is *throttled* according to the criterion analyzed in Sect. 4.1. In particular at any transmission opportunity the number of relayed packets is limited to $\min(N(t)/\alpha, N_u)$.

6. Simulator description and implementation

All the distribution strategies described previously have been implemented in a C++ simulator. As already mentioned, the simulator builds an overlay network topology where a single peer (called *source*) begins sending its information data, divided into k packets, to all the other peers. At the beginning, the source state is set to SEEDER while all the other peers are set to WAIT. The simulator operates on a time slot basis. During each time slot all the network nodes run the selected distribution protocol. Each node is characterized by a maximum upload B_u and download B_d capacity, which determine the maximum number of coded packets that a node can upload/download in a single time slot. The simulator does not assume any form of rate control, thus each node behaves independently aiming at saturating all its upload capacity. The packets are uploaded with a round-robin scheduling without any feedback from the recipients. On the receiver side, each node can download packets up to the saturation of its download bandwidth; the packets received in excess of the B_d limit are simply dropped.

The simulator is based on a complete implementation of the LT encoding and decoding procedures. As a source, each node has its own random generator for the RSD distribution and can generate the required number of coded symbols in each time slot, based on linear combinations of the original k information packets. As a receiver, each node progressively decodes the received packets. Both the standard BP decoder based on backward substitution of degree 1 equations and the recent OFG algorithm are available. In the OFG case, each node progressively constructs the generator matrix G, based on the generating equations of the received coded packets. The OFG decoder is run in every time slot to retrieve the maximum number of source packets x_i, given the currently received coded symbols. As soon as a node successfully decodes the original k information packets it signals such event to all its seeding nodes so as to stop them from pushing more coded packets.

The simulator goal is the measurement of the following performance indexes for each node p that is reachable from the source in d hops:

- $t_F(p, d)$: time slot of the first packet arrival, i.e. transition from WAIT to DECODER;
- $t_D(p, d)$: time slot when LT decoding is fully accomplished (the k information packets are recovered), i.e. when a node state switches from DECODER to SEEDER.

- $t_S(p,d)$: time slot when a SEEDER turns OFF, i.e, all of its neighbors completed decoding;

- $\bar{\epsilon}(p,d)$: LT code overhead estimated at node p, i.e. percentage of packets in excess of k downloaded while in the DECODER state. In other words, the number of downloaded packets is expressed as $(1+\bar{\epsilon})k$. The term overhead has been introduced in the rateless coding literature to identify the penalty due to random linear coding. With abuse of notation we use it to sum up the sub-optimality of both the LT encoding and the distribution strategy.

Previous indexes, are then averaged on all peers at given distance d allowing us to analyze the behavior of the different distribution strategies. From $t_D(p,d)$ we obtain $t_D(d) = \frac{1}{|\mathcal{T}_d|}\sum_{p\in\mathcal{T}_d} t_D(p,d)$ where $\mathcal{T}_d$ is the subset of nodes in $\mathcal{T}$ d hops away from the source. As a performance index for all the overlay we can compute $t_D = \frac{1}{N}\sum_d \sum_{p\in\mathcal{T}_d} t_D(p,d)$ where N is the number of nodes in the graph $\mathcal{T}$. $t_D(d)$ and t_D are termed absolute decoding time and represent average delays between the time instant when the source has initiated the data distribution and the retrieval of the information. In the case of a P2P live video streaming application such index determines the play-out delay. Analogous averages can be computed on $t_F(p,d)$ and $t_S(p,d)$. In particular, as pointed out in the next section, it is interesting to analyze the behavior of $t_D(d) - t_F(d)$, termed the relative decoding delay, which represents the average time between the reception of the first coded packet and the complete LT decoding, i.e., the time spent in the DECODER state at d hops from the source.

We also defined performance indexes to characterize the computational complexity of the distribution strategies we consider. To this end we defined the following measures that are averaged over all nodes in $\mathcal{T}$:

- decoding complexity c_d to estimate the number of row XOR operations in the OFG decoding algorithm when nodes are DECODER;

- encoding complexity c_e for the number of XOR operations to create coded packets when nodes turn into SEEDER;

- combination complexity c_c to consider the number of XOR operations to obtain a combination of received packets when nodes are DECODER;

All protocols have been evaluated on static topologies $\mathcal{T}$, based on the representation of the set of N active nodes in a P2P network as a finite graph of size N, where a vertex represents a peer and application-level connections between peers are modeled as edges.

7. Simulation results

In this section the SR [17], RE [4], RR, TRR, RRC and TRRC strategies are analyzed and compared. The goal of this analysis is twofold. On one hand, the advantages offered by relaying and combining coded packets are shown. On the other hand, the experiments contribute to get a deeper understanding of the behavior of protocols based on rateless codes in a complex random network. We also consider an idealized form of Random Network Coding (that we denote as iRNC) to obtain a lower bound on the delay and an upper bound on the computational complexity for the performance indexes we defined in Sect. 6.

7.1. Simulation parameters

The simulator described in Sect. 6 has been used to simulate the temporal behavior of the system with a time slot equal to 30 ms. Information packets of 1000 bytes and LT coding with $k = 100$, $c = 0.05$ and $\delta = 0.01$ are used. When testing the TRR and TRRC strategies the parameter $\alpha = 2.0$ is used, unless otherwise stated.

For the generation of the network topology $\mathcal{T}$ we considered several instances of Erdős-Rényi (ER) random graphs [20], which are described by a Poisson probability distribution for both the outgoing and incoming degree whose average is equal to z. Another set of experiments has been worked out on real topologies obtained from PPLive video streaming application [21]. PPLive peers organize in an overlay to receive and relay multimedia content for a particular channel. We used the crawler [22] to gather topological information of PPLive channels. Because of the overlay dynamics, the accuracy of the captured snapshots depends on the crawling speed. The crawler in [22] reduces the crawling time by using a distributed approach that allows one to capture snapshots of the overlay supporting a PPLive Cartoon channel in times ranging from 5 to 8 minutes. The size of captured snapshots varies according to a daily behavior ranging from 4000 to 8000 peers. In this paper we selected 25 snapshots with an average size of 6300 nodes with an average number of neighbors per peer equal to 46. In order to compare the results

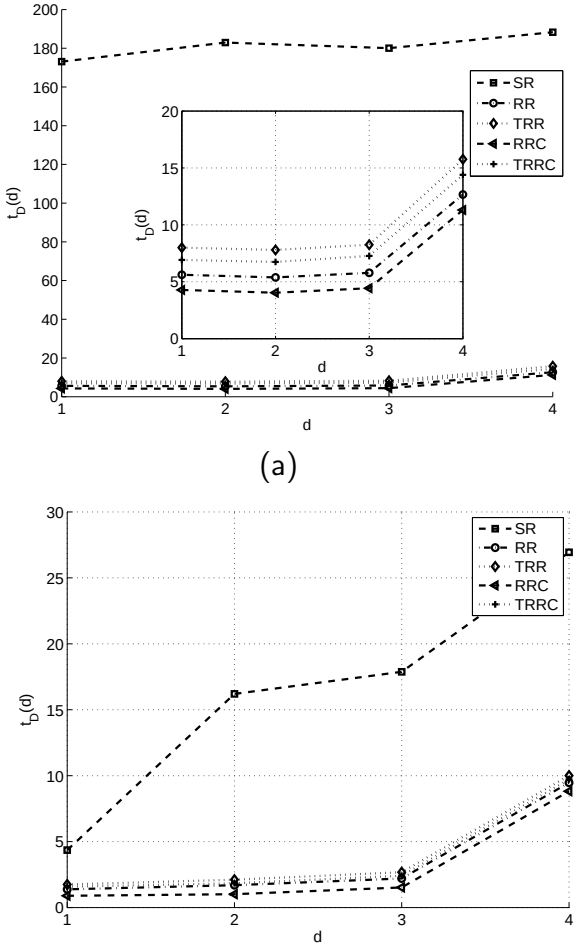


Figure 2: $t_D(d)$ for PPLive when source has $B_u = 256$ kbps (a) and $B_u = 10$ Mbps (b).

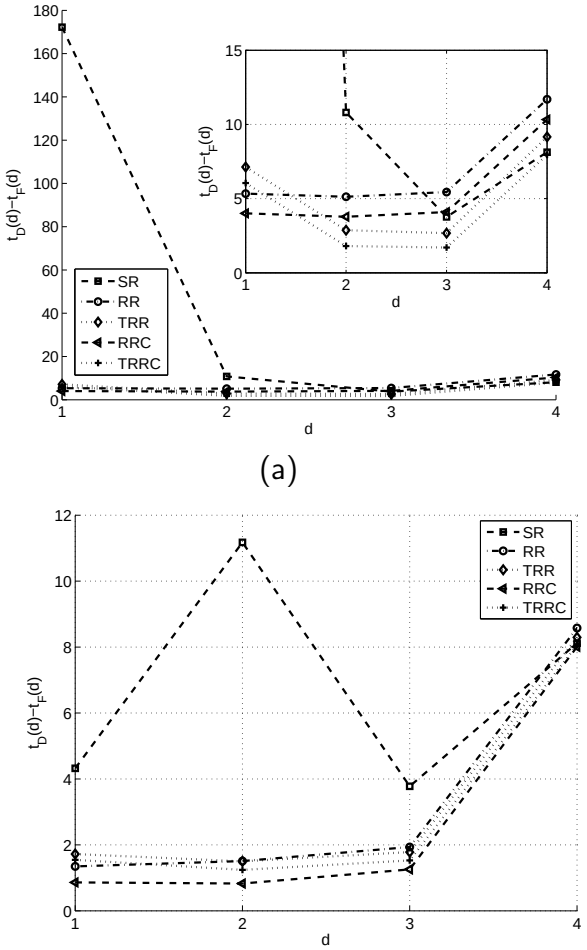


Figure 3: $t_D(d) - t_F(d)$ for PPLive, $B_u = 256$ kbps (a) and $B_u = 10$ Mbps (b) source.

Table 1: Bandwidth classes.

B_u	B_d	Percentage
10 Mbps	10 Mbps	5%
256 kbps	2 Mbps	15%
256 kbps	4 Mbps	60%
512 kbps	8 Mbps	15%
1 Mbps	20 Mbps	5%

with those obtained on random graph we used 25 ER graph instances with the same average size and $z = 46$. All the performance indexes reported in the following are obtained averaging the results of independent simulations on the available 25 graph instances. In each simulation the source is randomly chosen among the nodes from which it is possible to reach all the other $N - 1$ nodes. The bandwidth distribution shown in Tab. 1 has been used to randomly allocate the peer bandwidth capacities B_u, B_d. The selected bandwidth figures are representative of a realistic scenario with a majority of peers with limited capacities, e.g. accessing through ADSL links, and a small percentage of high capacity institutional nodes.

7.2. Comparison between OFG and BP

As already mentioned, in this work we propose to use OFG to decode LT codes. Such algorithm has a number of important features, the most important being the possibility of incrementally solving the system of linear equations determined by LT coding, its lower overhead for short block lengths k and its limited sensitiveness to the degree distribution of the coded packets. In Tab. 2 the average performance indexes t_F, t_D, t_S and $\bar{\epsilon}$, obtained when using the SR strategy and a source node with $B_u = 10$ Mbps on ER graphs, are shown for BP and OFG. As opposed to OFG, BP is very sensitive to the RSD parameters; in the BP case we noticed that using $c = 0.05, \delta = 1.0$ yields the best performance for $k = 100$, shown in Tab.2. It can be noticed that while the BP decoder requires a significant overhead $\bar{\epsilon} = 0.35$, which means that about 135 coded packets are needed to retrieve the original 100 packets, OFG needs only 107 coded packets. Clearly, the lower overhead positively impacts on all the other indexes, reducing by about the 18% the average decoding delay. Moreover, BP algorithm is not usable in a scenario where LT encoded packets are combined: indeed, the degree distribution of received

Table 2: Comparison between OFG and BP using SR on ER graphs: source with $B_u = 10$ Mbps.

	t_F	t_D	t_S	$\bar{\epsilon}$
BP	17.75	31.23	32.86	0.35
OFG	14.90	25.71	26.97	0.07

Table 3: Average performance indexes as a function of the upload capacity B_u of the source.

		$B_u = 256$ kbps				$B_u = 1$ Mbps				$B_u = 10$ Mbps			
		t_F	t_D	t_S	$\bar{\epsilon}$	t_F	t_D	t_S	$\bar{\epsilon}$	t_F	t_D	t_S	$\bar{\epsilon}$
ER graphs	iRNC	0.20	1.02	1.33	0.05	0.18	1.00	1.32	0.05	0.15	0.97	1.28	0.05
	SR [4]	170.95	183.20	184.48	0.07	50.99	62.21	63.48	0.07	14.90	25.71	26.97	0.07
	RE [17]	78.33	123.06	126.86	0.08	20.76	35.48	37.66	0.08	2.89	7.57	8.27	0.08
	RR	0.20	5.41	5.54	4.45	0.18	2.92	3.12	1.93	0.15	1.77	2.04	0.82
	TRR	2.86	5.94	6.12	1.55	1.23	3.04	3.29	0.50	0.67	2.13	2.41	0.27
	RRC	0.20	**3.77**	3.84	2.74	0.18	**1.83**	2.06	0.82	0.15	**0.99**	1.31	0.08
	TRRC	2.85	4.11	4.40	**0.09**	1.23	2.41	2.71	**0.07**	0.66	1.84	2.14	**0.07**
PPLive	iRNC	0.34	1.76	3.36	0.02	0.30	1.72	3.32	0.02	0.27	1.69	3.28	0.02
	SR [4]	173.92	181.39	182.72	0.07	49.08	55.58	56.90	0.07	11.47	17.65	18.97	0.07
	RE [17]	15.79	77.12	77.38	0.08	4.76	22.94	23.19	0.08	1.10	5.61	5.86	0.08
	RR	0.36	6.09	6.49	3.74	0.32	3.49	3.98	1.51	0.29	2.50	3.06	0.68
	TRR	5.43	8.53	9.08	0.61	1.47	3.92	4.48	0.38	0.86	2.95	3.56	0.20
	RRC	0.36	**4.75**	5.10	2.52	0.32	**2.86**	3.35	0.93	0.28	**1.81**	2.41	0.12
	TRRC	5.38	7.49	8.08	**0.08**	1.43	3.29	3.90	**0.09**	0.84	2.68	3.29	**0.09**

packets is different to RSD, making BP algorithm unusable. Therefore, all the experimental results reported in the following are worked out with OFG.

7.3. Analysis of delay and computational complexity

After this preliminary comparison we pass to the most important result of this work, i.e. the performance evaluation of the proposed random relay strategies. In Tab. 3 the average performance indexes obtained with SR, RE, RR, TRR, RRC, TRRC are shown as a function of the source upload capacity for both the ER graphs and the PPLive snapshots. It can be noticed that all the proposed RR, TRR, RRC and TRRC strategies, being able to increase the exploitation of the upload bandwidth, reduce tremendously all the delays. On the other hand, random coded packet forwarding can be heavily penalized in terms of overhead, due to the high probability to create duplicated packets in the overlay. In particular when the source upload is limited RR turns out to be practically unusable with $\bar{\epsilon} \gg 1$. As expected, using packet combinations in RRC significantly improves both in terms of de-

coding delay t_D and $\bar{\epsilon}$. This is clearly due to the lower probability to forward duplicated packets. It can be observed that in all the reported experiments RRC is the algorithm achieving the lowest t_D, shown in bold face in Tab. 3. RRC turns out to be efficient in terms of $\bar{\epsilon}$ only when the source has $B_u = 10$ Mbps. The performance impairment in the case of a source with limited upload capacity can be overcome by throttling the upload resorting to TRRC, which consistently yields the best $\bar{\epsilon}$ (boldface in Table). This advantage is paid in terms of slightly larger delays. TRR, i.e. the same upload reduction but without combinations, still yields large delays and overhead even if it performs better than RR. First row in Tab. 3 reports results for an idealized form of Random Network Coding in $GF(q)$ using some large prime power q. We simulated the data dissemination process by considering no overhead and by neglecting duplicate packets, i.e., all received packets are assumed to be innovative although they can be duplicates. It can be noted that this idealized form of RNC still has the overhead due to the arrival of late packets. Most important, as the source upload capacity increases the performance of RRC strategy gets closer to the lower bound represented by the idealized RNC strategy we chose. Furthermore, the overhead of the TRRC strategy is very close to the ideal one in all considered scenarios.

To get a better insight into the behavior of the various strategies in the following we will analyze the average $t_D(d)$ and $t_F(d)$, i.e. the delays as a function of the number of hops separating a node from the source. From this point on all the reported results refer to the case of PPLive snapshots. ER graphs show very similar behavior and they are omitted for conciseness. In Fig. 2 $t_D(d)$ is shown as a function of d when the source upload bandwidth is $B_u = 256$ kbps (a) and $B_u = 10$ Mbps (b). As already observed in terms of average t_D, RRC is the strategy yielding the lowest decoding delay at any distance from the source and independently of the source upload.

In Fig. 3 the relative decoding time $t_D(d) - t_F(d)$ is shown as a function of d when the source upload bandwidth is $B_u = 256$ kbps (a) and $B_u = 10$ Mbps (b). In Fig. 3(a) it can be noted that SR is heavily conditioned by the limited upload of the source when $d = 1$. Indeed, in the SR case nodes at distance $d = 1$ are directly served by the unique source; the other strategies, being based on relay of coded packets, let the information flow as soon as possible, thus reducing the LT decoding time. This advantage becomes less evident for higher distances since in such a case both SR and the other techniques rely on the presence of a number of nodes in the SEEDER state, that act as a set of independent sources. The number of such source nodes

clearly increases with the distance d. In the reported results such source propagation effect diminishes for $d = 4$ because of the particular topological structure of the PPLive overlay. Indeed PPLive snapshots are formed by very well connected nodes (average degree is 46) when $d \leq 3$. Nodes with $d > 3$ form low connected chains where packet upload can be limited by a single bottleneck. In Fig. 3(b) we show the same indexes when the source has a large upload; in such a case the SR relative delay for $d = 1$ is less penalized by the single source, whereas the values for $d > 1$ are the same as in 3(a). In the case of RR, TRR, RRC and TRRC the larger upload capacity of the source yields a relevant reduction of the relative decoding delays for all values of d (expect for $d = 4$ due to the mentioned topological issues).

Furthermore, TRRC achieves a high utilization of the upload capacities defined as $\eta_u(p,d) = \frac{R_u(p,d)}{B_u[t_S(p,d)-t_F(p,d)]}$, where $R_u(p,d)$ is the total amount of data uploaded by peer p placed at d hops from the source, averaged over all peers belonging to the same bandwidth class. In all the simulated scenarios TRRC exhibits an average η_u above 0.8 for ADSL links, and η_u of about 0.5 for the institutional nodes. As a reference SR achieves η_u around 0.2 for all bandwidth classes.

Finally, in Tab. 4 we present the results for the computational complexity of the considered strategies. We do not consider iRNC since in that case decoding is based on Gaussian Elimination requiring $\frac{k(k-1)}{2}$ row combinations; each row combination takes $\log_2 q$ XOR operations when q is a power of 2. Clearly, for large values of q this yields very large values for c_d [23]. In our settings, we obtain $c_d \approx 5000 \log_2 q$.

Except for SR, the decoding complexity c_d represents the most significant contribution to the overall computational complexity. It can be noted that RRC yields high decoding complexities, i.e., c_d, which can be explained by the fact that RRC is likely to create a large number of linearly dependent combinations of packets that translate into a high overhead as noted in Tab. 3. Linearly dependent combinations require several XOR operations in OFG to cancel out all terms. On the other hand, throttling reduces both overhead and computational complexity.

8. Conclusions and future works

In this paper we showed that the performance of data distribution in P2P networks can be improved tremendously at a very reasonable cost using rateless codes and RNC. In particular, we propose to perform the linear

Table 4: Average complexity indexes as a function of the upload capacity B_u of the source.

		$B_u = 256$ kbps			$B_u = 1$ Mbps			$B_u = 10$ Mbps		
		c_e	c_d	c_c	c_e	c_d	c_c	c_e	c_d	c_c
ER graphs	SR [4]	942.60	1138.40	-	943.63	1139.24	-	942.24	1138.01	-
	RE [17]	280.75	1218.03	-	279.15	1215.94	-	281.46	1206.50	-
	RR	138.61	2854.09	-	220.95	2675.89	-	235.80	1508.49	-
	TRR	77.36	1659.65	-	146.55	1354.23	-	220.60	1221.24	-
	RRC	204.63	6593.53	405.58	246.85	3772.38	191.38	248.65	2564.07	101.71
	TRRC	43.44	3182.34	93.05	178.65	2877.60	82.81	258.16	2470.56	82.45
PPLive	SR [4]	883.81	1094.41	-	882.89	1093.22	-	882.15	1094.56	-
	RE [17]	504.63	1119.89	-	507.88	1120.13	-	498.54	1121.60	-
	RR	235.22	2720.70	-	320.80	1599.64	-	336.75	1450.41	-
	TRR	173.42	1567.53	-	232.29	1287.41	-	300.14	1193.96	-
	RRC	278.36	6825.73	542.27	357.32	3305.85	376.49	361.22	2220.11	281.55
	TRRC	137.03	2981.95	88.94	270.47	2187.40	64.20	350.19	2176.33	63.76

combinations of information packets in GF(2), to reduce the computational complexity, and to use LT codes to pre-encode the information packets. We let nodes propagate encoded packets as soon as possible, so as to increase the utilization of the upload capacity and reduce the delay after which a node is able to accomplish the decoding, thus retrieving the original information. We showed that letting a node relay linear combinations, even if in GF(2), of the coded packets accumulated during the decoding process is very likely to reduce the amount of useless information, so improving the overall system performance in terms of both delay and bandwidth utilization. The most simple random relay approach is potentially dangerous because it is very likely to saturate the upload bandwidth with duplicated packets. To solve this problem we propose to initially throttle the upload bandwidth. The results we obtain are really promising: the improvement on the system performance are shown by means of very detailed simulation of an overlay network of nodes running encoding and decoding stages. The overlay networks we considered are both random graphs and snapshots of PPLive. The proposed distribution strategy turned out to yield very low decoding delay so as to sustain a larger throughput, while avoiding to flood the overlay network with excessive useless coded packets. In particular, we observed that as the source upload capacity increases the performance of RRC strategy gets closer to the lower bound represented by the idealized RNC strategy we chose. Furthermore, the overhead of the TRRC strategy is very close to the ideal one in all considered scenarios. As for the computational complexity, RRC requires a large number of XOR operations; this can be explained by the fact that RRC is likely to create a large number of linearly dependent

combinations. On the other hand, throttling reduces both overhead and computational complexity.

Ongoing research in this area includes the exploitation of network awareness for overlay formation and coded packet distribution, e.g. prioritizing the transmission towards peers with more upload capacity. Finally, the implementation of the proposed relaying strategies in a real P2P application is currently underway on Planetlab.

References

[1] J. Byers, J. Considine, M. Mitzenmacher, S. Rost, Informed content delivery across adaptive overlay networks, IEEE/ACM Trans. on Networking 12 (5) (2004) 767–780.

[2] D. Kostić, A. Rodriguez, J. Albrecht, A. Vahdat, Bullet: high bandwidth data dissemination using overlay mesh, in: 19th ACM SOSP, 2003, pp. 282 – 297.

[3] M. Wang, B. Li, R^2: Random push with random network coding in live peer-to-peer streaming, IEEE Journal on Selected Areas in Communications 25 (9) (2007) 1655–1666.

[4] C. Wu, B. Li, rStream: resilient and optimal peer-to-peer streaming with rateless codes, IEEE Trans. on Parallel and Distributed Systems 19 (1) (2008) 77–92.

[5] C. Fragouli, J. Widmer, J. Le Boudec, Efficient broadcasting using network coding, IEEE/ACM Trans. on Networking 16 (2) (2008) 450–462.

[6] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, J. Crowcroft, XORs in the air: practical wireless network coding, IEEE/ACM Trans. on Networking 16 (3) (2008) 497–510.

[7] C. Gkantsidis, P. Rodriguez, Network coding for large scale content distribution, in: IEEE INFOCOM 2005, 2005, pp. 2235 – 2245.

[8] R. Koetter, M. Medard, An algebraic approach to network coding, IEEE/ACM Trans. on Networking 11 (5) (2003) 728–795.

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

[10] M. Luby, LT codes, in: IEEE FOCS, 2002, pp. 271–280.

[11] M. Mitzenmacher, Digital Fountains: A Survey and Look Forward, in: IEEE ITW, 2004, pp. 271–276.

[12] V. Bioglio, R. Gaeta, M. Grangetto, M. Sereno, On the fly gaussian elimination for LT codes, IEEE Communication Letters 13 (2) (2009) 953–955.

[13] D. MacKay, Fountain codes, IEE Proc. Communications 152 (6) (2005) 1062–1068.

[14] S. Puducheri, J. Kliewer, T. Fuja, The design and performance of distributed LT codes, IEEE Trans. on Information Theory 53 (10) (2007) 3740–3754.

[15] R. Gummadi, R. Sreenivas, Relaying a fountain code across multiple nodes, in: IEEE ITW, 2008, pp. 149 – 153.

[16] M. Wang, B. Li, How practical is network coding?, in: 14th IEEE IWQoS, 2006, pp. 274 – 278.

[17] M. Grangetto, R. Gaeta, M. Sereno, Rateless codes network coding for simple and efficient P2P video streaming, in: IEEE ICME, 2009, pp. 1500 –1503.

[18] R. Graham, D. Knuth, O. Patashnik, Concrete Mathematics, Addison-Wesley, 1994.

[19] N. L. Johnson, Urn models and their application, Wiley, 1977.

[20] M. E. J. Newman, S. H. Strogatz, D. J. Watts, Random graphs with arbitrary degree distributions and their applications, Physical Review E 64.

[21] PPLive, http://www.pplive.com.

[22] S. Spoto, R. Gaeta, M. Grangetto, M. Sereno, Analysis of PPLive through active and passive measurements, in: HotP2P, 2009, pp. 1 – 7.

[23] C. Fragouli, E. Soljanin, Network coding applications, Found. Trends Netw. 2 (2) (2007) 135–269.