Reducing content distribution time in P2P-based multicast using rateless codes

Marco Grangetto, Rossano Gaeta, Matteo Sereno
Dipartimento di Informatica
Università degli Studi di Torino
Corso Svizzera, 185, 10149 Torino
Email: {grangetto, rossano, matteo}@di.unito.it

Abstract—In this paper a novel protocol for application level multicast, that is designed so as to exploit the advantages offered by rateless coding, is proposed. With rateless coding a set of \( k \) packets can be used to generate an arbitrary number of coded packets. Any set of slightly more than \( k \) coded packets can be used by a node in the network to recover the information. The proposed protocol is based on a simple encoding and relaying policy that is able to reduce the average decoding delay in an application level multicast overlay. Moreover, no prior knowledge of the network topology is required thus making the solution very simple and general. The performance of the novel protocol is analyzed employing synthetic instances of random graphs. This study, besides showing the advantages of the proposed technique, permits to get a deeper insight into the behavior of rateless codes in complex networks. Moreover, the proposed protocol is evaluated on real topologies, obtained crawling a widespread p2p video streaming application. The proposed protocol turns out to be very efficient with a significant gain in terms of decoding delay on both synthetic and real networks.

I. INTRODUCTION

Several recent results point 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], [16] or efficient broadcasting in ad hoc wireless networks [3], [7], just to mention a few. Most of the cited results have been catalyzed by the seminal promises of network coding [8], [4], where nodes in the network are allowed to combine information packets instead of simply forward them. However till present, network coding solutions have fallen short of expectations in real applications. Nonetheless, a novel class of channel codes called rateless codes [11], [9], [15], specifically designed for application level coding, is turning out as a practical tool for efficient data dissemination.

Rateless codes [11], [9], [15] are a family of 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 depending on the state of the communication channel. 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 a plethora of users experiencing different channel conditions, i.e. different packet losses; the success of rateless codes in this field is witnessed also by the recent adoption of a code of this class in the DVB-H and 3GPP MBMS standards. Nevertheless, their application to complex networks, e.g. application level multicast, is not straightforward and deserves further investigations and research efforts. Some potentialities of rateless coding in the area of p2p video streaming have been recently shown in [1]. In particular, as discussed in [1], the exploitation of the resilience of rateless codes allows to cope with the network dynamics due to the unpredictable behavior of the peers. These dynamics are often in contrast with the demand for stable streaming bit rates. Another interesting feature of rateless codes is that they can address the problem of delivery redundancy and content reconciliation. In most of the p2p streaming proposals a peer concurrently downloads contents from multiple peers and uploads towards multiple peers. Although this improves the bandwidth availability and allows to counteract network dynamics, content reconciliation policies are required. In the p2p streaming applications, currently deployed, this issue is solved by buffer maps exchanges and rarest first search policies. However, buffer maps exchanges cause a significant overhead, that for some p2p streaming application can be up to 20% of the video bitrate, as reported in [2]. Thank to the rateless characteristic, in principle, there is no limit to the number of uniquely coded symbols generated from the original set of symbols and hence there is no need for content reconciliation, as no redundant contents exist in the network. Therefore, a simpler push approach can be adopted to let the information reach all the peers in the overlay.

Nonetheless, rateless coding poses novel issues, as well. In particular the information flows has to be divided into coding blocks; since the coding efficiency increases with the block size, that in turn determines the decoding delay, the code optimization and application real-time constraints may clash on each other. In [1] this issue has led to the selection of a small coding block, i.e., 100 packets, sacrificing the coding efficiency. Moreover, in [1] the network coding principle is exploited only partially, since the DF approach is applied on every peer-to-peer connection and peers are not allowed to propagate new coded information before the
complete decoding of a block; this again translates into a significant delay for every hop in the overlay. On the other hand, alternative solutions in the literature may not be viable in practical scenarios. As an example, in [14] a distributed rateless code design is presented; 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 at coding time. This latter assumption is clearly unfeasible in a real dynamic overlay. In [5] another approach to relay rateless codes across multiple nodes is proposed. Also this solution looks quite complex and not very practical and flexible when used in real applications.

In this paper, a simple and efficient protocol, based on the use of rateless codes, is proposed. The goal of the new protocol design is a more efficient exploitation of DF principle for fast propagation of the information in an application level multicast. The direct application of DF to every node to node connection, as in [1], is used as a starting point and a benchmark for performance evaluation. The main contribution of the work is the proposal of a novel relay strategy able to significantly reduce the delay, i.e. the time instant when a node is able to fully recover the transmitted sequence of packets. As opposed to [14], the proposed solution does not require coordination among the nodes and can be applied to any network topology. The presented experimental results, worked out on random graph topologies, show that our solution is able to reduce the delay from 2 to 4 times with respect to the approach used in [1]. Clearly, the achieved low delay points out that our protocol can be profitably exploited by real-time multicasting applications such as p2p video streaming. To evaluate this possibility the protocol has been also tested on real topologies derived from the PPlive [13] p2p television network, a widespread application for p2p streaming of TV channels. Another contribution of the paper is represented by the detailed analysis of the behavior of rateless codes and DF protocols on complex networks. In particular, Generalized Random Graph (GRG) [12] are used to test and understand the behavior of rateless code as a function of the network size and average number of connections per node. To the best of our knowledge, this is the first study of the kind, and besides revealing the advantage of the proposed design it contributes to get a deeper insight into the phenomena induced by the use of rateless codes for application level multicast. Therefore, the presented results are a useful guidance for future improvements.

The paper is organized as follows. In Sect. II some background on rateless codes is recalled. Sect. III presents our protocol design. In Sect. IV the simulator used to test the proposed protocol is described and validated versus an analytical model derived from GRG theory. Experimental results, performance evaluation and protocol analysis are reported in Sect. V. In Sect. VI our conclusions and future research directions are drawn.

II. RATELESS CODES BACKGROUND

Luby Transform (LT) codes [9] are the first class of efficient rateless erasure codes that achieve optimality as the data length growth. LT are random block codes; 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 symbols $y_j, j \geq 0$, 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 [9] the Robust Soliton Distribution (RSD) $RSD(i)$, is proposed for the selection of the degree $i = 1, \ldots, k$. In the RSD $\epsilon$ is a suitable positive constant and $\delta$ is the allowed failure probability at the decoder. It [9] is demonstrated that the decoder fails to recover the data with probability at most $\delta$ 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 $\lim_{k \rightarrow \infty} \epsilon = 0$, i.e. the code is asymptotically optimal. Given a set of $n$ encoded symbols the decoder can attempt reconstruction of the information message by applying Gaussian elimination on the equivalent code generator matrix $G_n = \{g_{i,j}^1, \ldots, g_{i,j}^n\}$, with row vectors $g_j = \{g_{0,j}, \ldots, g_{k-1,j}\}$. A simpler decoder is obtained using the sum-product algorithm [10]. In this case one looks 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 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. Another class of rateless codes is represented by Raptor codes [15] which exhibit lower encoding/decoding complexity and at the same time excellent performance. The complexity can be reduced by making use of a data precoding stage (usually a high rate block code) followed by an LT code with a modified degree distribution.

A key issue for the practical adoption of random rateless codes is the communication of the linear combinations $g_j$ associated to each coded packets. The simplest solution is based on the insertion of a $k$ bits header which explicitly signal the ex-ored packets. Such overhead can be limited if the sender and receiver use a common random generator with the same seed; in this latter case each packet only needs to be numbered incrementally to keep synchronization between the random generators. The header cost is $\log_2(k(1+\epsilon))$. To limit further such cost several contiguous rateless symbols can be packed in a single network transfer unit.

III. PROTOCOL DESCRIPTION

For the scope of this paper we assume that an overlay network topology $T$ has been established among a set of peers participating to a multicast event. At startup, there is a single source peer $s \in T$ that needs sending its information data to all other peers in the overlay. Let us assume that the overlay is constructed in such a way that $s$ can reach any other peer in $T$ with a certain number of hops. The protocols described
Algorithm 1: Store and Forward protocol($p$)

1. Initialize RSD random seed generator
2. Initialize the list of neighbors $\mathcal{N}(p)$.
3. Initialize the list of decode symbol identifiers $\mathcal{D} = \{0, \ldots, k-1\}$

\[ \text{if } p = s \text{ then} \]
   \[ \text{Set number of decode symbols } k' = k. \]
   \[ \text{Initialize the list of decoded symbol identifiers } \mathcal{D} = \{0, \ldots, k-1\} \]
\[ \text{else} \]
   \[ k' = 0. \]
   \[ \text{Initialize the list } \mathcal{D} = \emptyset \]
\[ \text{end if} \]

\[ \text{while } k' < k \text{ do} \]
   \[ \text{Receive a coded packet } y_j \text{ and retrieve the corresponding } g_j. \]
   \[ \text{for } i \in \mathcal{D} \text{ do} \]
       \[ \text{if } g_{j,i} = 1 \text{ then} \]
       \[ y_j = y_j + x_i, \quad g_{j,i} = 0. \]
   \[ \text{end if} \]
\[ \text{end for} \]
\[ \text{Add } g_j \text{ to } \mathcal{G}. \]
\[ \text{Run LT decoder; update } k' \text{ and } \mathcal{D}. \]
\[ \text{end while} \]

\[ \text{Remove } p \text{ from all neighbors lists.} \]
\[ \text{for } p' \in \mathcal{N}(p) \text{ do} \]
   \[ \text{Generate a novel LT symbol using seed } S(p) \text{ and forward it to } p'. \]
\[ \text{end for} \]

in the following aims at reducing the delay experienced by the peers in the network by exploiting rateless codes, while keeping the protocol operation as simple as possible.

The most straightforward solution is constituted by direct use of the DF approach over each peer-to-peer connection in the overlay. In order to avoid transmitting the generating equations $g_j = \{g_{0,j}, g_{1,j}, \ldots, g_{k-1,j}\}$ each peer in the network is characterized by a unique random generator seed $S(p)$ that can be shared during connection setup; as an example, the seed can be obtained by a proper hashing function of the peer identifier, e.g. IP address and port. The drawback of this approach, in the following referred to as DF Store and Forward (DF-SF), is due to the fact that a node can generate new coded symbols only after having decoded the $k$ original source packets. It is obvious that this requirement introduce a delay that depends on the value of $k$ and LT code overhead $\epsilon$. Unfortunately, in order to limit $\epsilon$ one would like to select a reasonably large value for $k$. The DF-SF has been already proposed in [1] for p2p video streaming application. In [1] very small values of $k$ in the order of 100 have been reported so as to limit the critical delay parameter in video streaming; nevertheless, this choice makes the rateless code very suboptimal in terms of the required overhead. It is worth pointing out that, notwithstanding the suboptimal LT code exploitation, the most significant advantage of the DF-SF approach is the possibility to design a simple push based packet routing. In fact, every coded packets is a useful drop of information for any receiver; this point permits to almost cancel the protocol overhead for chunk retrieval experienced by popular pull based p2p video streaming applications.

The DF-SF protocol used in this paper is described by Algorithm 1. In every time instant each node in the network can be either a source or a receiver. At startup there is a single source $s$ and all peers in $\mathcal{T}/s$ are the potential receivers. Every node $p$ has number of directed links towards its neighbors, represented by the list $\mathcal{N}(p)$. The receiving nodes keep waiting for additional LT coded packets $y_j$, construct the code generator matrix and progressively decode the information packets $x_i$. Progressive decoding is obtained by canceling from each received packet the symbols $x_i$ that the node has already resolved; then, a new generating equation can be added to the matrix $\mathcal{G}$ and the message passing algorithm is iterated until no more equations of degree 1 can be found. As soon as a node $p$ has decoded all the $k$ information packets, it signals its ancestor to stop forwarding packets and it turns into a new data source using its upload capacity to feed new packets across its output links. The uniqueness of the random generator seed $S(p)$ guarantees that every newly coded symbol is useful for the receiver, independently of the packets received on the other incoming connections. As time passes it is likely that more nodes turn into the source state, increasing the upload capacity available through the overlay.

The DF-SF protocol can be improved from the point of view of the delay experienced by the peers in the overlay; such delay is due to the fact that a node has to wait for complete decoding before uploading new packets. To improve on this side the DF Relay protocol (DF-RL) is proposed. The protocol operations are reported in Algorithm 2. In DF-RL each node maintains a buffer $\mathcal{B}$ of the received coded packets and it is allowed to upload them to its neighbors $\mathcal{N}(p)$ before complete decoding has occurred. It is worth noticing that in such a case the value of the seed used to generate the packet must be communicated to the receiver; the overhead is represented by the number of bits used in the random generator, e.g. 32, and turns out to be negligible for reasonable packet sizes. The only constraint is that the same packets cannot be sent to more than one recipient. In absence of loops in the overlay, this simple rule avoids the presence of duplicated packets in the overlay and still guarantees that each packet flowing in the network is useful for any recipient, given the DF principle. In other words, any set of received coded packets is guaranteed to exhibit RSD distributed degrees and therefore is characterized by asymptotic zero overhead. In this scenario the only limit to packet forwarding is represented by the ratio between the node upload and download capacities; as far as the download bandwidth is equal or larger to the upload the input buffer $\mathcal{B}$ is not empty and the node can contribute to the information flow. As in the DF-SF case, as far as the node has retrieved all the information packets it changes to the source state and start emitting newly generated LT symbols with its unique seed.

The constraint on the absence of loops in the overlay can be removed by using simple loop avoidance procedure as described in Algorithm 2. Each node $p$ maintains a supplementary list $\mathcal{S}(p)$ containing the identifiers of the peers that are pushing packets towards $p$. Each $p' \in \mathcal{S}(p)$ is characterized by a integer ranking value $\text{Rank}(p')$. Such rank is initialized to zero, incremented by 1 when a non duplicated packet is received from $p'$ and decremented when a duplicated packet...
is found. A duplicated packet is identified if the corresponding generating equation is already present in the generator matrix that is being progressively constructed at node $p$. As far as $\text{Rank}(p')$ falls below a threshold fixed to a negative integer value $R_{Lt}$, $p'$ is recognized as a redundant source and no more packets from $p'$ will be accepted. Moreover, the node $p$ informs $p''$ to remove the link between the two peers to avoid wasting transmission bandwidth in the overlay.

IV. SIMULATOR DESCRIPTION AND IMPLEMENTATION

The DF-SF and DF-RL protocols described by Algorithm 1 and Algorithm 2 have been implemented in C++. The kernel of the simulator is represented by the $LTnode$ class which implements a peer node and can be configured according to either the DF-SF or DF-RL approaches. The major methods implemented in the class are:

- maintain the list of neighbors $N(p)$;
- perform LT encoding;
- perform progressive LT decoding;
- relay packets in input buffer $B$ (DF-RL only);
- maintain the list of senders $S(p)$ and implement loop avoidance (DF-RL only).

In order to simulate the protocols performance an array of LTnodes is created and their neighbor lists are initialized according to the directed graph $T$ describing the topology.

The simulator operates on a time slot basis. During each time slot $t$ all the network nodes run the selected protocol; in the case of DF-RL only packets received in previous time slots are allowed to be relayed at time $t$ for causality. Finally each node is characterized by a maximum upload $B_u$ and download $B_d$ capacity, given by the number of coded packets that a node can upload and download respectively in a single time slot.

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

- $t_F(p,d)$: time slot of the first packet arrival;
- $t_T(p,d)$: time slot when LT decoding is fully accomplished (the $k$ information packets are recovered);
- $t_S(p,d)$: time slot when a node stops, since reception, relaying (DF-RL only) and LT coding operations are finished.

The most important performance index used in the following is represented by $t_D(p,d)$, which represents the delay experienced by peer $p$ at distance $d$. In order to rank the protocol behavior as a function of the distance from the source the average decoding delay at distance $d$ can be computed as:

$$t_D(d) = \frac{1}{|T_d|} \sum_{p \in T_d} t_D(p,d)$$

where $T_d$ is the subset of nodes in $T$ $d$ hops away from $s$. As a performance index for all the overlay we can compute

$$t_D = \frac{1}{N} \sum_d \sum_{p \in T_d} t_D(p,d)$$

where $N$ is the number of nodes in the graph $T$.

Finally, the estimates of the standard deviations $\sigma_D(d)$ and $\sigma_D$ of the decoding delays at distance $d$ and in all the overlay respectively are computed; this latters represent the delay jitters experienced in the all the overlay or at a given distance $d$.

Two simulators have been used with different levels of detail as far as the implementation of the LT encoder and decoder is regarded.
A. Full simulator

The full 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 $\tau_i(i)$ and can generate the required number of coded symbols in each time slot, based on linear combinations of the $k$ information packets. For simplicity each packet is represented by a 32 bit integer and the linear combinations are obtained by ex-or among such integers. This choice does not affect the performance evaluation since the LT coding principle holds for any packet size. In a real scenario the packet size would be selected depending on the application and networking protocols constraints, such as delay, protocol maximum transfer unit, etc. As a receiver, each node progressively constructs the generator matrix $G$, based on the generating equations of the received coded packets. The LT message passing decoder is run in every time slot to retrieve the maximum possible number of source packets $x_i$, given the currently received coded symbols. As a consequence, the generator matrix size changes dynamically depending on the number of received and decoded symbols.

The full simulator is definitively a precise implementation of both the LT encoder/decoder operations and the proposed DF protocols. Since LT codes have more-than-linear encoding/decoding complexity as a function of $k$, the precise simulator, which runs one LT encoder/decoder per simulated peer, requires a relevant computational burden. This issue is solved by resorting to a lighter simulator described in the following section.

B. Light simulator

The light simulator is able to perform protocols evaluation without resorting to precise LT encoding and decoding functions, which are the most demanding procedures in terms of both memory and processing power in the case of the full simulator. The light simulator emulates a DF without generating the actual coded symbols and does not require each node to operate the generation of random linear combinations. On the receiver side, complete decoding can be declared as far as $K = k(1 + \chi(k, \delta, c))$ symbols have been received, where the overhead $\chi(k, \delta, c)$ is a random variable whose distribution depends on the code block length $k$ and the RSD parameters. Sample realizations of $\chi(k, \delta, c)$ for any given set of the parameters can be obtained offline employing a single LT encoder and decoder. Each node in the light simulator is attributed a sample value of $\chi(k, \delta, c)$ and the LT decoder is substituted by a simple comparison between the number of downloaded symbols and $\chi(k, \delta, c)$; as far as one node has received more than $\chi(k, \delta, c)$ coded packets, successful decoding of the $k$ information messages is achieved. The above described approximation assumes that the distributed LT code, generated by different peers with unique seeds, is equivalent to an LT code generated by a single source/receiver couple. This hypothesis holds true as far as the random generators in the distributed system can be considered independent of each other. In conclusion, the light simulator only neglects the unlikely event of synchronization among different random generators. The implementation of this light LT decoder only amounts to maintain a counter of the received packets in each node. Nevertheless, this simple solution would prevent the use of the loop avoidance algorithm in DF-RL; in order to implement such part of the protocol, the node still need the ability to recognize duplicated packets sent by its ancestors. As a consequence, each packet in the light simulator is represented by a unique integer identifier\(^1\). The receiving node must maintain a list of such identifiers and can look for duplicated packets and implement the ranking policy avoiding relaying loops.

In both the light and full simulators the following system parameters are used. For the rateless code generation throughout this paper we considered an LT code with $k = 1000$, $c = 0.05$, $\delta = 10^{-4}$, yielding an overhead $\epsilon = 0.38$; such overhead can be reduced increasing the block length $k$ or, given $k$ finding the best RSD parameters $c$ and $\delta$ or alternative degree distributions. Such goals are outside the scope of this paper and remain open research issues in the channel coding area[6]. In the DF-RL case the loop control threshold has been fixed to $R_L = -50$. Moreover, the relay buffer ($B$) size has been limited to the value of 10.

DF-SF and DF-RL have been evaluated on static topologies, 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. It must be noticed that in a real scenario the logical connections among peers in a P2P network yield a constantly and randomly changing topology as the result of users joining and leaving the network. Nonetheless, if we assume that the time scale of search operations is much shorter than the time scale of the multicast of the $k$ packets, we

\(^1\)The uniqueness of the packet identifiers is guaranteed by simulator at a global level
can reasonably assume that at any instant in time the snapshot of the P2P network topology can be viewed as an instance of a finite graph of size $N$. For the generation of the network topology $T$ we considered 30 instances for each of two classes of directed GRG [12]: the first class is composed by Erdös-Rényi graphs, which are described by a Poisson probability distribution for both the outgoing and incoming degree whose average is equal to $z_1$. The outgoing and incoming degree distributions of the second class of graphs is a discrete uniform in the interval $[z_1 - 3, z_1 + 3]$ extremes included. The source is randomly chosen among the nodes from which it is possible to reach all the other $N - 1$ nodes.

Unless otherwise stated, in the rest of the paper the parameter $s$ is represented by:

- three values for $z_1$ (10, 15, 20);
- three values for the graph size $N$ (1000, 10000, 100000);

In order to validate the light versus the full simulator the performance indexes measured with both solutions have been compared. As already predicted, the light simulator turns out to provide an high level of accuracy. In particular, the two experimental settings considered in the paper. Given an instance we estimated the value of $t_d$ from which $d$ hops away from the source depends on the average number $n_s(d)$ of its uploaders. In general, if $B_d > B_u$ (as in ADSL access links) the actual download rate of a node that is $d$ hops away from the source is given by the complex expression $\min(B_d, n_s(d) \cdot B_u/z_1)$. The characterization of this random variable is a very difficult task therefore Equations 2 and 3 have been derived under the simplifying assumption that $B_d = B_u/z_1$, which amounts to limit the total download bandwidth from multiple sources to that of a single node.

For each node $p$ that is $d$ hops away from the source of an instance we estimated the value of $t_d(p, d)$ from which we derived the average value $t_d(d)$ as well as $t_d$. We also computed the standard deviations $\sigma_D(d)$ and $\sigma_D$. The results of the comparison between Equation (2) and the simulator outcomes are reported in Fig. 2 while the results concerning Equation (3) and the corresponding simulations are shown in Fig. 3. For the purpose of validation two values for $B_u$, (60, 600) and consequently $B_d = B_u/z_1$ have been considered. In both figures the left column refers to Poisson graphs while the right column refers to discrete uniform graphs. Also the top rows refer to the case $B_u = 60$ whereas the bottom rows refers to $B_u = 600$. It can be observed that the model predictions follow quite closely the simulation outcomes that are represented by the average values of $t_d(d)$ and $t_D$ with the corresponding $\sigma_D(d)$ and $\sigma_D$ represented as error bars. Actually, one of the reasons why the simulated results for $t_d$ are always greater than our approximation is due to the last term in Equation (3); in fact, (3) merges the contributions of all peers whose distance from the source is greater than or equal to $d(N)$, underestimating the delays of the farthest nodes. We omit all the results we obtained for the other combinations of the system parameters since in all cases results with similar quality have been observed.

V. Protocol analysis

In this section, the DF-SF and the novel DF-RL protocol performance is analyzed. The goal of this task is twofold. On the one hand, the advantage offered by DF-RL in terms of decoding delay is shown. On the other hand, the experiments
contribute to a deeper understanding of the behavior of protocols based on rateless codes in a complex network scenario.

A first set of experiments has been worked out using Poisson graphs and discrete uniform graphs with the same features already described in Sect. IV, i.e. variable average node degrees $z_1$ and sizes $N$. In the following we comment the results obtained in the Poisson case only; the same considerations hold for the discrete uniform graphs that are omitted here for brevity. The achieved performance has been evaluated in several scenarios with different bandwidth characteristics:

- all nodes with fixed symmetric upload and download bandwidths;
- all nodes with fixed asymmetric upload and download bandwidths;
- 5% of high performance symmetric nodes and 95% of nodes with lower asymmetric fixed bandwidths. The source $s$ is always assigned the high performance class.

The last scenario is representative of a real world situation where there is a majority of limited performance peers accessing through ADSL links and a limited percentage of high capacity institutional nodes.

In Fig. 4 the values of $t_D(d)$ for a Poisson graph are reported for DF-SF and DF-RL, along with the values predicted by Equation (2) in the case $N = 10^4$, $z_1 = 15$ and $B_u = B_d = 200$. Every value of $t_D(d)$ is accompanied by an error bar of $\pm \sigma_D(d)$, which represents the decoding delay jitter of the nodes at a given distance from the source. First, both DF-SF and DF-RL exhibit a decoding delay that becomes significantly lower than the prediction as $d$ increases. It must be recalled that Equation (2) has been derived under the assumption that a node can sustain only one uploader. Nevertheless, when $B_d > B_u/z_1$ as in all our cases, a node can download packets from multiple sources. In the DF-SF this effect shows up only for $d > 1$, since all nodes at distance $d = 1$ depend on the single source $s$. The peers at distance $d > 1$ can then exploit the presence of multiple sources closer to $s$ that have already decoded the message and act as a set of distributed uploaders. As a consequence, a node
that is farther away from $s$ is likely to obtain the packets from several of its neighbors, thus increasing the probability of fully utilize its download bandwidth. Moreover, DF-RL yields a lower delay for all values of $d$. This advantage is due to the fact that a node can always relay the packets is receiving without waiting for complete decoding. Therefore, even at distance $d = 1$, a node can have multiple uploaders, i.e. directly from the source and indirectly from longer paths connecting it to the source $s$. Since all the packets circulating in the network are independently generated LT symbols the node experiences an increased download capacity speeding up the message decoding. It is important to recall, that DF-RL gain is relevant even if the size of the relay buffer has been fixed to 10, that in the case $B_u = 200$ represents a strong limitation. Fig. 5 shows the same set of results in the case of a network with peers with asymmetric bandwidths. The precise settings are $B_d = 380$, $B_u = 20$, $N = 10^4$ and $z = 15$; in this case the advantage of DF-RL is more apparent with decoding delays that are 4 times lower with respect to DF-SF.

Finally, it is worth pointing out that in both figures the average decoding delay for high values of $d$ is almost constant. This is due to the the greater number of nodes in the network that behave as sources; for both DF-SF and DF-RL after a certain number of hops, the number of available uploaders of a node is likely to saturate its download bandwidth and the decoding time approaches the minimum theoretical value of $\lceil k(1 + \epsilon)/B_d \rceil$.

In Fig. 6 we plot the average relative decoding time $t_D(d) - t_F(d)$, i.e. the time passed on average between the arrival of the first packet and the complete decoding. The results refer to the case of $N = 10^4$ nodes with asymmetric bandwidths. The reported curves confirm the result that the farther the nodes are, the higher the number of sources they can exploit to fully utilize the download bandwidth. Moreover, the evaluation of $t_D(d) - t_F(d)$ permits to better appreciate the advantage offered by DF-RL at distance $d = 1$, that we briefly outline while commenting on Fig. 5. In the case $d = 1$ the lower relative decoding time is clearly due to the possibility
nodes, e.g. $C$ and $D$ (dash lines). In the figure, we assume that $B$ forwards its coded packets to $C$, which in turn acts as relay for $A$ and $D$. Since $A \in \mathcal{N}(D)$, $D$ can upload to $A$, as well. It is thus evident that $A$ has more chances to increase its download, therefore reducing the decoding delay and, more importantly for the overlay, shorten the time when it turns into a source of new LT coded packets. It is worth pointing out that a node in the relay phase has more difficulties in saturating its upload capacity, as opposed to a source node that can generate exactly the required amount of coded symbols. In fact, we imposed the constraint that the same packet cannot be relayed to several destinations; this simple rule, along with the loop control policy, avoids the presence of duplicated packets in the overlay and guarantees that the LT coded symbols collected by any node respect the RSD for the degree, yielding optimal overhead. If one has more control on the overlay topology, a possibility which has not been considered in this work, the same packet may be let traveling on separate subgraphs, e.g. separated multicast trees, improving the efficiency of the protocol. The simple but general relay solution adopted by DF-RL has another characteristic, namely the fact that the relay
capacity decreases as a function of $d$. Coming back to the example in Fig. 7, $B$ receives on average $B_u/z_1$ packets per time slot from $s$, but can upload only $B_u/z_1^2$ packet to each of its neighbors. As consequence, the propagation speed of the relay decreases exponentially with the distance from the source. Fig. 7 naturally suggests that a clustered topology, i.e. where neighbors of a node are neighbor themselves, would further improve the effectiveness of the relay phase by shortening the length of the indirect paths, thus increasing their capacity. Despite the clustering coefficient of the GRGs considered in this paper, that is asymptotically equal to zero for $N \to \infty$, the contribution of the indirect paths is relevant and is able to significantly reduce the decoding delay with respect to DF-SF.

In Fig. 8,9,10 the average delay $t_D$, experienced by all the peers in the overlay, is shown as a function of the network size. The error bar around each value is the average of the standard deviation $\sigma_D$. The results in Fig. 8,9,10 are worked out for $z_1 = 20$ in the case $B_u = B_d = 200$, $B_u = B_d = 1$, $B_u = 20$ and $B_d = 380$, respectively. It is worth pointing out that in all the bandwidth settings we considered the DF-RL consistently outperforms DF-SF in terms of decoding delay. In the case $B_u = B_d = 1$ the size of the relay buffer is not limiting the performance and the decoding delay reduction of DF-RL is more significant that in the case $B_u = B_d = 200$. The most evident improvements are shown in the asymmetric bandwidth case with decoding times about 3 times lower than DF-SF. Finally, DF-RL exhibits a lower standard deviation, i.e. the delay jitter is lower.

Let us define the DF-RL gain $G = t_{DF-SF}^D / t_{DF-RL}^D$ as the ratio between the decoding delays of the two protocols. In Fig. 11...
The values of $G$ are shown for graphs with $z_1 = 15$. The DF-RL gain for all the tested bandwidth allocations is analyzed as a function of the network size $N$. The 95% confidence interval of each estimated value is reported as well. It is worth noticing that the $G$ is higher when the upload capacity is limited, e.g. cases $B_u = 1$, $B_u = 20$; furthermore such gain is remarkable in the case of asymmetric bandwidths (from 4 to 2 in the case $B_u = 20$, $B_d = 380$). Fig. 11 also shows that $G$ decreases as the network size increases; this phenomenon can be explained taking into account that the average length of the indirect paths, that connect a given node to its multiple uploaders, scales approximately as described by Equation (1); therefore as already discussed, the relay contribution to the upload decreases in the sense that packets relayed by the neighbors arrive with higher delay and at a limited pace.

In Fig. 12 $G$ is reported as a function of $z_1$ in the case $N = 10^4$. It can be noticed that DF-RL exhibits a moderate advantage when $z_1$ increases because of the higher number of neighbors that a node can use to spread the packets. Clearly, the value of $B_u$ may limit this benefit.

The second set of experiments has been worked out on real topologies $T$ obtained from PPlive [13], a popular p2p live video streaming application. PPlive peers organize in an overlay to receive and relay multimedia content for a particular channel. We developed a crawler to gather topological information of PPlive channels. Because of the overlay dynamics, the accuracy of the captured snapshots depends on the crawling speed. In principle, a perfect snapshot is captured if the crawling process is complete (all peers are contacted) and instantaneous. To reduce crawling time we designed and developed a distributed crawler that allowed us 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, with an average degree $z_1 = 69.1$.

The PPlive snapshots allowed us to test the proposed protocols in a practical scenario. For such a purpose, we experimented DF-SF and DF-RL on 30 PPlive snapshots using the mixture of high and low performance peers we already considered in the case of GRG.

The results for $t_D$, $t_F$ and $t_S$ along with their standard deviations are reported in Tab. I. It is worth pointing out that DF-RL almost halves the decoding delay with respect to DF-SF also in this real scenario. Another important aspect is represented by the low value of $t_F$ allowed by DF-RL. Finally, in Fig. 13 the $t_D(d)$ is reported, showing the DF-RL speed-up as a function of the distance from the source of the information.

VI. CONCLUSIONS AND FUTURE WORK

In this paper the novel Digital Fountain Relay (DF-RL) protocol, based on the exploitation of rateless codes, has been proposed. DF-RL is able to significantly reduce the delay for information retrieval in a application level multicast with respect to other solutions in the literature. This gain has been reported on both synthetic instances of random graphs and real world snapshots taken from a PPlive video channel. The improvements are particularly apparent in the presence of network nodes with limited and asymmetric bandwidths. Moreover, DF-RL does not require any topological constraint, e.g. absence of loops or distribution trees. Finally, the protocol has been designed to assure fast and simple operations based on LT codes encoding and efficient relaying of the packets to the neighboring nodes.

Ongoing research in this area include the investigation of
smart scheduling policies for DF-RL. In fact, in current implementation, each node forward packets to the neighborhood in a round-robin fashion but more accurate exploitation of the network characteristics, e.g. prioritizing the transmission towards peers with more outgoing connections shall improve the performance. Moreover, our findings have shown that DF-RL can exploit networks with a high clustering coefficient, which have not been investigated in the present study. DF-RL can be improved also by using some topological knowledge to relax the strong constraint that the same packet cannot be forwarded to multiple recipient. Another important task is represented by the investigation of the impact of network dynamics on the protocol. Finally, the deployment of the proposed protocols in a real application will be addressed.

REFERENCES


