Distributed Scheduling for Low-Delay and Loss-Resilient Media Streaming with Network Coding

Distributed Scheduling for Low-Delay and Loss-Resilient Media Streaming with Network Coding

Anooq Muzaffar Sheikh, *Student Member, IEEE,* Attilio Fiandrotti, *Member, IEEE,* Enrico Magli, *Senior Member, IEEE*

***Abstract*—Network coding has been shown to be very effective for collaborative media streaming applications. A pivotal issue in media streaming with Network Coding (NC) lies in the packet scheduling policy at the network nodes, which affects the perceived media quality. In this paper we address the problem of finding the packet scheduling policy that maximizes the number of media segments recovered in the network. We cast this as a distributed minimization problem and propose heuristic solvers that make the proposed framework robust to infrequent or inaccurate feedback information. Moreover, the proposed framework accounts for the properties of layered and multiple description encoded media to provide graceful quality degradation in case of packet losses or lack of upload bandwidth. Experimental results on a local testbed as well as PlanetLab suggest that the our scheduling framework achieves better media quality, lower playback delay and lower bandwidth consumption than a random-push scheme.**

***Index Terms*—Distributed Scheduling, P2P, media streaming, Network Coding**

I. Introduction

Network Coding (NC) [1] has emerged as a recent breakthrough in multicast communications. Let us consider a scenario where a single source node must distribute a message to multiple receivers through some intermediate nodes. The intermediate nodes, instead of simply forwarding the received packets, transmit linear combinations of the received packets to the receivers. Once a receiver has collected enough linearly independent packets, it solves a system of linear equations and recovers the message. If the network nodes are allowed to recombine the received packets, then maximum network throughput can be achieved as demonstrated in [2].

NC finds application to a wide range of multimedia-related problems [3] including low-delay media communications such as live media streaming and videoconferencing. Such applications are particularly challenging because they entail distributing bandwidth demanding media contents to populations of cooperating users with tolerable lags that range from a few seconds to fractions of the second.

While previous research has demonstrated the benefits of NC in terms of reduced communication delays, comparatively little attention has been given to the design of bandwidth-efficient packet scheduling schemes. In random-push scheduling, nodes flood their neighbors in the network with coded packets until a stop message is received. Our previous research [4], [5], [6], [7] has demonstrated that random-push scheduling does enable low-delay communications, however that happens only at the price of increased bandwidth requirements. However, due to the lack of synchronization among nodes, part of the transmitted packets is received *after* the the generation has already been recovered, thus wasting part of the available bandwidth. The purpose of this paper is to address this problem, developing bandwidth-efficient scheduling schemes to enable practical low-delay NC. In particular, in this paper, we introduce a distributed push-based packet scheduling framework for low-delay media communications, which enables each network node to select the transmission policy that maximizes the number of generations recovered in the network. The main features of our framework are that 1) it accounts for the nodes decoding status, the packet playback deadlines, and the status of the network links; 2) it allows one to employ multiple description and layered coded media to achieve graceful quality degradation; 3) it is robust to outdated or infrequent feedback information.

We evaluate our framework on two platforms, namely a discrete events testbed and a real peer-to-peer media streaming application running on the PlanetLab distributed testbed. We show that the proposed distributed scheduler performs close to a centralized scheme and achieves better media quality than a random-push reference that is oblivious of the nodes and links status.

The remainder of this paper is organized as follows. In Section II we describe the basics of media streaming with NC and we overview the existing literature on packet scheduling. In Section III we describe the proposed distributed scheduling framework. In Section V, we experimentally evaluate our framework in terms of delivered media quality and bandwidth consumption for different network scenarios. Finally, conclusions are drawn in Section VI.

II. Background and Related Work

The theoretical foundations of NC were originally introduced by Ahlswede *et al.* in their pioneering work [2]. Chou *et al.* [8] first proposed a practical NC scheme suitable for real networks, where the node topology and the coding functions at the nodes are not known and communications between nodes take place asynchronously. They proposed to

Part of this work was supported by PRIN ARACNE, a research project funded by the Italian Ministry of Education and Research and COAST-ICT STREP, a research project funded by the European Union.

divide the information stream in chunks called *generations* and each generation is subdivided into k blocks of symbols (simply *blocks* in the following) of identical size. Each time the source transmits a packet for a generation, it randomly draws a subset of blocks and generates a linear combination thereof that is transmitted together with the list of combined blocks (*encoding vector*) in the form of a network packet. We call the number of linearly independent packets received by a node for a generation the *rank* of the generation. Once the rank of a generation is equal to k, we say that the generation has *full rank*. At this point, the node solves a system of linear equations (*e.g.,* via Gaussian elimination) and recovers the generation. In practical NC applications, a receiver must typically collect $k' > k$ packets because non all received packets are innovative.

The random-push scheduling mechanism was proposed by Wang and Li [6] with their R^2 architecture [9] for live video streaming. In their scheme, the network nodes proactively push random recombinations of the received packets to randomly selected recipient nodes in the network, hence the definition of *random-push* NC. As soon as a node decodes a generation it signals its neighboring nodes about this event by sending a *stop message*, after which the nodes stop sending packets for the specific generation. The benefits of the random-push approach are, among others, reduced buffering times and sustained throughput. A mathematical analysis on P2P streaming systems with NC is given in [10]. UUSee [11], represents a large scale deployed multimedia streaming system based on the design of R^2 [9]. SonicVoD [12] also represents a large scale implementation of a media streaming service that utilizes NC to distribute video contents in a mesh push-based design. In [13] a media streaming protocol is proposed using NC in a mesh-based P2P network. In particular, a specific network topology is used to group peers interested in the same video segment and NC is employed over the segments individually for improved resilience. Although NC possesses several benefits for video streaming, however, the coding operations at the intermediate nodes may introduce a delay that is detrimental for live video streaming. The authors in [14] propose to overcome this issue by appropriately selecting and placing a limited number of high performance nodes in the network for NC operations, while the other nodes simply relay the packets they receive.

Scalable media streaming refers to the coding technique which fragments a single high-quality media-stream to n substreams. Typically the substreams are formed by arranging the media content in classes of different importance having unequal error protection, or in classes of equal importance having equal error protection. With the reception of each additional substream the quality of the media stream increases gracefully. Multiple Description Coding (MDC) also allows to partition a single media stream into *descriptions* which can be decoded independently. Each description guarantees a basic level of reconstruction quality. Reception of additional descriptions allows to improve the reconstructed video quality.

A number of papers have proposed the use of NC with SVC and MDC [3], where NC is performed on each substream independently or among different substreams. For example the authors in [15], propose a prioritized video streaming system with NC that divides the video content into classes of priority to provide unequal error protection. NC packets for each class are formed separately and are served to network nodes with different capabilities. [16] employs layered coding in a tree-based P2P network to improve throughput using path diversity. In [17] NC is used with H.264/SVC, where encoded packets for a layer i are formed by linear combination of symbols from layer i and its preceding layers. A similar approach is used in [18], where a NC scheme is constructed in such a way to deliver the base layer with higher decoding probability. In [19] NC has been used for error protection purposes in a video streaming application. The source node distributes packets coded with rateless codes to other nodes in the network. Network nodes decode and forward the received packets and also forward linear combinations thereof for erasure correction purposes if the available bandwidth permits. A similar approach is used in [20], which also takes into account the linear independence of the symbols used for creating the NC encoded packets. NC for P2P media streaming has also been studied from the perspective of peer heterogeneity. For example [21] focuses on peer upload bandwidth by modeling the P2P system in a multi-rate multicast optimization problem. The solution is obtained using linear programming and leads to maximum aggregate rate assignment among the peers. The authors in [22] propose to use SVC with NC to adapt to peer heterogeneity. In their application "Chameleon", the symbols selected for encoding belong to the same layer to allow peers to subscribe to specific number of layers based on the bandwidth capacity. Nonetheless, as is the case with SVC, subscription of the base layer is mandatory to decode the enhancement layers. Moreover two-bit buffermaps are intelligently used to exchange and signal the decoding status of the peers, which aids effective content dissemination. The authors in [15] address P2P video streaming by dividing the video in different classes of priority. Peers request for packets belonging to specific priority class to the neighboring peers. The transmitter peers respond to the request with encoded packets by solving an optimization problem with the aim to assign optimal rate to each class and minimize the average distortion at the requesting peer. The authors in [23] study the use of rate allocation in MDC with prioritized NC, while in [24] they propose the use of feedback mechanism for rate allocation of the descriptions to maximize the video quality.

Regarding packet scheduling mechanisms for media streaming with NC, in our previous work [4], we evaluated the performance of the random-push mechanism for low-delay streaming by reducing the buffering time at the nodes. The results suggest that the random-push mechanism is suboptimal at allocating the upload bandwidth because recipient nodes may end up receiving more packets than are required to decode the generation, resulting in wasted bandwidth. Authors in [11] suggest an early signaling method to stop transmitter nodes from sending packets for generations close to decoding, resulting in only sufficient number of packets required for decoding. Some studies focus on the availability of media segments in the network, where a media segment owned by few nodes is highly prioritized to form the NC packets. For example [25] proposes a scheduling scheme for live streaming application

that prioritizes scarce segments that have an earlier playback deadline. However they do not consider the video quality perceived at the client nodes and the heterogeneous changes in the network conditions. In [26] a scheduling scheme is proposed to recover from packet losses using a push-pull approach in a multi-tree topology called *random multicast trees*. However in this case the packet scheduling mechanism is dependent on the proposed tree-based network topology. SPANC [27], presents a optimized scheduling model to minimize the packet delivery delay in a tree-based topology. It was shown that an appropriate packet scheduling at the network nodes can reduce the time required to recover the media contents, albeit in this scheme NC is used only for erasure-correction purposes. [22] uses H.264/SVC with NC to provide adaptive video quality to nodes in a mesh topology. Packet scheduling is based on subscription of recipient nodes to transmitter nodes for specific layers and exchange of buffermaps. The packet scheduling mechanism is however heavily dependent on the proposed application architecture.

In our previous work [5] we proposed to address the issues highlighted in [4] with a distributed packet scheduling approach, which showed promising results in terms of better media quality for reduced bandwidth requirements. In this paper we broaden the scope of the proposed scheduling scheme by accounting for the inter-layer dependencies of scalable and multiple description coded media, and by accounting for the rateless codes properties. Also, while in our previous work we evaluated a preliminary version of our scheduler on a synthetic testbed only, in this paper we perform an extensive evaluation using a real P2P application both on a controlled conditions testbed and on the PlanetLab.

III. Proposed Scheduling Model

In this section, we define a cost function that accounts for the bandwidth budget required at a network node to enable another node to recover a generation. Then, we formulate the problem of finding the packet scheduling policy that maximizes the number of generations recovered in the network under a maximum available bandwidth constraint as a distributed optimization problem that is independently solved by each network node. Finally, we show how to solve such problem in a way that is robust to out-of-date feedback and concurrent optimization at each node in the network and we describe a practical signaling protocol for delivering the required feedback information.

A. Media and Network Model

We model the media stream distributed to the network nodes as the bi-dimensional arrays of generations shown in Figure 1. Each generation is identified within the media stream by a temporal index $t \in [1, T]$ and a substream index $s \in [1, S]$: in the following we use the notation (t, s) to indicate the generation with temporal index t and substream index s. Each generation (t, s) is subdivided in $k_{t,s}$ blocks of symbols, where generations with identical temporal index t have identical playback duration. However, without loss of generality, in the following we assume that generations with identical quality

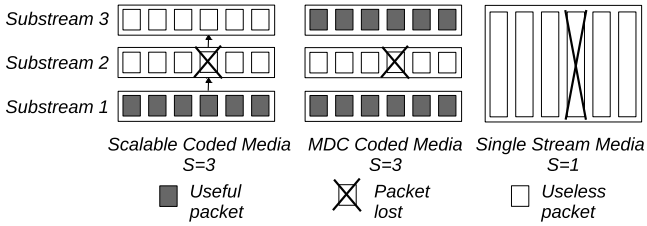

Fig. 1. Media stream model for single stream media, multiple-descriptions and scalable-coded media streams as a stack of substreams for one generation $T = 1$. Decoding dependencies between substreams are illustrated with arrows.

index s are subdivided in an identical number of blocks k_s and all generations have identical playback duration of G_t seconds.

Such media model is suited to describe traditional media coding formats as well as multiple descriptions and multiple layers (scalable) encoded media as follows. In the case of single layer media, we have only one substream (*i.e.,* S=1), and each generation can be decoded independently from the others by the media player as soon as enough innovative, *i.e.,* linearly independent, packets are recovered. In the case of multiple descriptions media, each description is modeled as a separate substream and each generation can be decoded independently from the others by the player, where recovering multiple generations with identical temporal index increases the playout quality. In the case of scalable media streams, each layer is modeled as a separate substream and enhancement substreams ($s > 1$) can be decoded by the player only if the dependencies between substreams illustrated in the figure with the use of arrows are met. In order to account for such dependencies, we say that a substream s can be decoded by the player only if its *predecessors* have been recovered too and we indicate as s' the set of substreams that are predecessors of substream s. For example for the scalable coded media, if $S = 3$ as in Figure 1 and $s = 3$, we indicate the ancestor of generation (t, s) as the set $\{t, s'\}$, where $s' = \{1, 2\}$.

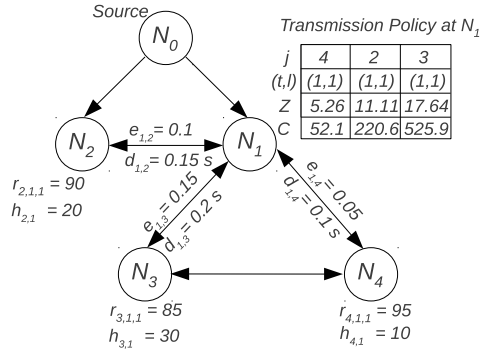

Fig. 2. Example of a network with $N = 5$ nodes and the parameters used in the model.

Each node is assigned an integer number i as unique identifier within the network, where $i = 0$ for the source node and $i > 0$ for the remaining nodes, and the i-th node in the network is indicated as $\mathcal{N}_i$ in the following. The overlay of network nodes is represented as the graph $(\mathcal{N}, \mathcal{E})$ composed

of the nodes $\mathcal{N} = \{\mathcal{N}_1, ..., \mathcal{N}_N\}$ and the edges $\mathcal{E}$, where the edge $\mathcal{E}_{i,j} \in \mathcal{E}$ is the directed link that connects peer $\mathcal{N}_i$ to $\mathcal{N}_j$, has packet error probability $e_{i,j}$ and delay $d_{i,j}$. For any node $\mathcal{N}_i$, we define $A_i \subset \mathcal{N}$ as the *neighborhood* of $\mathcal{N}_i$, that is the set of nodes that are connected to $\mathcal{N}_i$ and exchange packets with it, where $|A_i|$ is the size of the neighborhood of $\mathcal{N}_i$. We indicate as $r_{j,t,s}$ the rank of generation (t, s) at $\mathcal{N}_i$, so the number of packets that $\mathcal{N}_j$ must collect to recover (t, s) is equal to $k_s - r_{j,t,s}$. Generations with identical temporal index t must be played out at the same instant, however each node may be slightly misaligned in the playout position within the media stream. Therefore, we indicate the playout deadline of generations with temporal index t at node $\mathcal{N}_j$ as $h_{j,t}$. We say that generation (t, s) is successfully recovered at node $\mathcal{N}_j$ if the node receives enough innovative packets to recover (t, s) (plus all its predecessor $\{(t, s')\}$ if any) before the generation playout deadline. The key notations used in the rest of this section are summarized in Table I.

S	Number of substreams in the media
s	Substream index, $s \in [1, S]$
s'	Set of predecessors of s
N	Number of nodes in the network
$\mathcal{N}_i$	Network node i, $i \in [1, N]$
G_t	Temporal duration of generations
$r_{j,t,s}$	Rank of generation (t, s) at node $\mathcal{N}_j$
$h_{j,t}$	Decoding deadline of generations with temporal index t at $\mathcal{N}_j$
$e_{i,j}$	Packet error rate on link from $\mathcal{N}_i$ to $\mathcal{N}_j$
$d_{i,j}$	Delay on link from $\mathcal{N}_i$ to $\mathcal{N}_j$ [ms]

TABLE I
KEY NOTATIONS USED IN THE MEDIA AND NETWORK MODEL.

B. Decoding Cost Formulation

We define the *decoding cost* $Z_{i,j,t,s}$ as the number of packets that node $\mathcal{N}_i$ must transmit to $\mathcal{N}_j$ to enable $\mathcal{N}_j$ to recover generation (t, s) before the playout deadline $h_{j,t}$. Such cost function is computed as explained below.

At any moment, the number of innovative packets that node $\mathcal{N}_j$ must receive to recover generation (t, s) is equal to $k_s - r_{j,t,s}$ as discussed above. Due to the random packet combinations at the network nodes, not all packets received by $\mathcal{N}_j$ are however innovative. Any generation belonging to substream s is composed of k_s blocks of symbols, and all their possible combinations span over a space of approximate size 2^{k_s} in the considered binary network coding scenario. Any linear combination of the $r_{j,t,s}$ packets already received at $\mathcal{N}_j$ is not innovative, and the this set of "seen" combinations spans over a space of approximate size $2^{r_{j,t,s}}$. Hence, any packet received by $\mathcal{N}_j$ is innovative only if does not fall in such set of seen combinations, which happens with probability

$$1 - \frac{2^{r_{j,t,s}}}{2^{k_s}} = \frac{2^{k_s - r_{j,t,s}}}{2^{k_s}}. \tag{1}$$

So, accounting for the probability that some of the packets received by $\mathcal{N}_j$ are not innovative, the expected number of packets that $\mathcal{N}_j$ must collect to recover (t, s) is equal to

$$\frac{k_s - r_{j,t,s}}{\frac{2^{k_s - r_{j,t,s}}}{2^{k_s}}} = (k_s - r_{j,t,s}) \; \frac{2^{k_s}}{2^{k_s - r_{j,t,s}}}.$$

When dealing with scalable coded media, $\mathcal{N}_j$ recovers (t, s) only if all the predecessors s' have been recovered as well. So, the number of packets that $\mathcal{N}_j$ must receive to recover (t, s) is equal to

$$\sum_{x \in \{s, s'\}} (k_x - r_{y,t,x}) \; \frac{2^{k_x}}{2^{k_x - r_{y,t,x}}}.$$

Due to the errors on the links, each packet from $\mathcal{N}_i$ to $\mathcal{N}_j$ is lost with probability $e_{i,j}$. Eventually, accounting for the error rate of the links too, the number of packets that $\mathcal{N}_j$ must receive from $\mathcal{N}_i$ to recover the generation is equal to

$$Z_{i,j,t,s} = \frac{\sum_{x \in \{s, s'\}} (k_x - r_{j,t,x}) \; \frac{2^{k_x}}{2^{k_x - r_{j,t,x}}}}{1 - e_{i,j}}. \tag{2}$$

Finally, it should be noted that generations close to the decoding deadline should be allocated more transmission opportunities to account for an urgency principle. To this end, we weight the decoding cost of each generation at each node by their corresponding decoding deadlines. Accounting also for the transmission delay $d_{i,j}$, our *time-weighted decoding cost* function is

$$C_{i,j,g,s} = (h_{j,t} - d_{i,j}) \, Z_{i,j,t,s}. \tag{3}$$

Finally, the minimization of such cost function is the object of the optimization framework described in the next section.

C. Cost Minimization

Periodically, the opportunity to transmit one packet arises for each node of the network, and each node selects the packet scheduling policy that *maximizes the expected number of generations timely recovered in the network*. Let $b_{i,j,t,s}$ indicate the transmission of one packet for generation (t, s) from node $\mathcal{N}_i$ to $\mathcal{N}_j$, where $b_{i,j,t,s} = 1$ if the transmission takes place and 0 otherwise. Let us now hypothesize that a central coordinator exists in the network: the problem of finding the optimal policy at each node $\mathcal{N}_i$, for $i \neq j$, can be casted at a central coordinator in the form of the minimization problem

$$\text{minimize} \quad \sum_{i=1}^{N} \sum_{\mathcal{N}_j \in A_i} \sum_{\forall t} \sum_{\forall s} b_{i,j,t,s} C_{i,j,t,s} \tag{4}$$

subject to

$$\sum_{\mathcal{N}_j \in A_i} \sum_{\forall t} \sum_{\forall s} b_{i,j,t,s} \leq 1 \; \forall i \tag{5}$$

$$b_{i,j,t,s} \leq r_{i,t,s} \; \forall i, j, t, s \tag{6}$$

where $i \neq j$. The constraint (5) means that at each transmission opportunity node $\mathcal{N}_i$ is allowed to transmit only one

packet to one its neighbors for a single generation. Moreover, the transmission takes place only if $\mathcal{N}_i$ has already received packets to transmit for (t, s) (*i.e.,* $r_{i,t,s} > 0$).
This work deals however with networks of totally uncoordinated nodes, so we target a totally distributed packet scheduling scheme rather than a centralized strategy. Therefore, we recast the problem in (4) into N independent problems that each network node can solve independently. In detail, each node $\mathcal{N}_i$ solves the following problem

$$\text{minimize} \quad \sum_{\forall \mathcal{N}_j \in A_i} \sum_{\forall t} \sum_{\forall s} b_{i,j,t,s} C_{i,j,t,s} \qquad (7)$$

$$\text{subject to (5) and (6).}$$

Such problem can be solved selecting by node $\mathcal{N}_i$ the neighbor $\mathcal{N}_j$ and the generation (t, s) with lowest associated cost $C_{i,j,t,s}$ as we show in the following.

IV. Practical Scheduler Implementation

The minimization problem in (7) can be solved optimally when i) the network nodes have up-to-date feedback available at each transmission opportunity (perfect feedback), and ii) it does not happen that multiple nodes solve the optimization problem (7) concurrently. Perfect feedback is however expensive to provide due to the associated cost in terms of signaling bandwidth, if feasible at all. Also, feedback messages could be delayed or lost altogether due to the delays and the losses on the links. Moreover, in uncoordinated networks the nodes autonomously grant themselves transmission opportunities, so it may happen that multiple nodes attempt to transmit a packet at the same time. If multiple nodes concurrently solve (7), the outcome is a sub-optimal allocation of the available network bandwidth, even under perfect feedback hypothesis. In this section, we present two robust scheduling algorithms that solve the optimization problem (7) dropping the requirements, respectively, for perfect feedback and coordinated transmission opportunities. Both algorithms are based on an improved version of the feedback protocol described in detail in [4] that we briefly overview in the following.

A. Feedback Signaling

Every G_t seconds, the source parses one generation of the media stream (the *source position*) and distributes encoded packets to the network. The generation currently played out at a node is the *playback position* of the node and is updated every G_t seconds. Initially, each node buffers t_b seconds of the media stream, so a node playback position always lags $\frac{t_b}{G_t}$ generations behind the source position. Figure 3 illustrates the case of node $\mathcal{N}_2$ playing out generation $(5, 1)$ while the source is seeding generation $(10, 1)$, (buffering time is equal to 5 generations). Generations encompassed between the source position (included) and the playback position of a node (excluded) are the *decoding region* of the node, *i.e.,* the generations for which the node is interested in receiving packets. We define as *decoding status* of the node, the vector of integer numbers indicating the ranks of the generations in the node decoding region plus the playout deadline of the earliest generation in the decoding region. For the example scenario in Figure 3, the node decoding status is composed of the vector $[r_{2,6,1}, \ldots, r_{2,10,1}]$ plus the relative decoding deadline $h_{2,6}$. The nodes broadcast to their neighbors feedback information consisting of i) the node decoding status ii) the measured packet loss rate on the incoming links. The feedback information is encoded as a vector of few integer numbers that is exchanged by the nodes using two mechanisms.
i) Explicit feedback: periodically, a node transmits a keepalive message to its neighbors, so that its neighbors can tell if the node has gone offline due to a network failure. Keepalive messages carry updated feedback information in piggyback mode, so each time a node receives a keepalive message, it also receives updated feedback information from the neighbor. Moreover whenever a node decodes a generation (t, s) it broadcasts a *stop message* to all its neighbors to indicate that it has decoded the generation and no longer wishes to receive packets for that generation.
ii) Embedded feedback: in addition, each encoded media packet carries an updated feedback information from the transmitter in piggyback mode, so to increase the frequency at which feedback is spread in the network without further loading the network with extra feedback packets.
As the feedback information consists in a few integer numbers only, embedding feedback in every media packet results in a negligible bandwidth increase for an improved feedback precision, as we show later on in the experimental section.

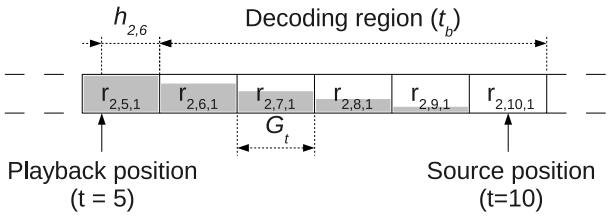

Fig. 3. Sample status of node $\mathcal{N}_j$, for $j = 2$ and $S = 1$. The source seeds generation $t = 10$, the node plays out generation $t = 5$ and buffers generations $t \in [6, 10]$. Next played out generation ($t = 6$) deadline is $h_{2,6}$.

B. Asynchronous Distributed Scheduler

The Asynchronous Distributed Scheduling (ADS) algorithm is meant to solve (7) in a way that is robust to out-of-date or missing feedback and is presented in pseudo-code as Algorithm 1. The algorithm is executed at each node $\mathcal{N}_i$ every time the node is granted a transmission opportunity and the algorithm output is the scheduling policy that represents the optimal solution to (7). Each policy is indicated in the following as the tuple $\pi = \{j, (t, s), Z, C\}$, where j is the identifier of the recipient node $\mathcal{N}_j$, (t, s) identifies the generation in the media stream and Z and C are the expected number of missing packets and the time-weighted costs function in (2) and (3) respectively. The node computes then a list of all the possible scheduling policies, and policies that do not meet the constraints in (5) and (6) are discarded from the list. At this point, the the list is sorted in increasing time-weighted cost C order, and in the following we indicate as

$\pi^x = \{j^x, (t,s)^x, Z^x, C^x\}$ the x-th policy in the list starting from the top (*e.g.,* π^1 is the policy at the top of the list and C^1 is the lowest cost among all policies). The policy at the top of the list is the solution with lowest expected cost to our optimization problem and hence the optimal solution. However, due to imperfect feedback, feedback information might be outdated and, for example, nodes may transmit packets to $\mathcal{N}_j$ for a generation that $\mathcal{N}_j$ has already recovered. Therefore, $\mathcal{N}_i$ draws the integer number $x \in [1, m^*]$ with uniform probability and the x-th policy from the top of the list is selected. The packets received so far by the node for generation $(t,s)^x$ are randomly recombined as described in Section II and a packet is eventually transmitted to node j^x. Finally, $\mathcal{N}_i$ updates the expected rank of (t,s) at $\mathcal{N}_j$ accounting for the probability that the transmitted packet is innovative (1) and is not lost on the channel $(1 - e_{i,j})$.

Algorithm 1 Asynchronous Distributed Scheduler - ADS

1: list feasible scheduling policies $\pi = \{j, t, s, Z, C\}$
2: sort list by increasing cost C
3: draw $x \in [1, m^*]$ from uniform p.d.f.
4: update expected rank of $\mathcal{N}_{j^x}$
5: **return** optimal policy π^x

Since in the asynchronous network the occurrence of a transmission opportunity is not synchronized among the transmitter nodes, selecting a tuple from $x \in [1, m^*]$ is beneficial for the following reasons: i) the algorithm prioritizes recipient nodes that are close to achieve a full rank for transmission, hence multiple nodes might choose the same recipient for packet transmission, which would result in surplus packets; ii) feedback cannot be sent very frequently, therefore the feedback information at the nodes is slightly outdated; iii) it may happen that a recipient node achieves full rank immediately after sending a feedback message. To explain this, consider the example of a recipient node $\mathcal{N}_5$ in a fully connected graph of 50 nodes, that requires only 10 packets to decode its generation (*i.e.,* $Z = 10$ for simplicity of illustration we assume no packet losses on links). At time $t = 1$ node $\mathcal{N}_5$ sends a feedback message to all its neighbors. The occurrence of a transmission opportunity for 10 of $\mathcal{N}_5's$ neighbors occurs in the time interval $t = [1.01 - 1.02]$, at which they transmit a packet to $\mathcal{N}_5$. At this point node $\mathcal{N}_5$ decodes the generation and broadcasts this information to all its neighbors. However a transmission opportunity arises for the remaining 40 nodes in the time interval $t = [1.02 - 1.03]$ before the message broadcasted by $\mathcal{N}_5$ reaches the remaining transmitter nodes. Thus the remaining transmitter nodes also transmit to $\mathcal{N}_5$ albeit $\mathcal{N}_5$ had already decoded the generation, so resulting in the delivery of surplus packets at $\mathcal{N}_5$. Conversely, if we consider the ADS scheduler with $m^* = 5$ then every transmitter node will select among the top 5 scheduling policies with uniform probability. Therefore, irrespective of the time the transmitter nodes receive the feedback message from $\mathcal{N}_5$, out of 50 neighbors of $\mathcal{N}_5$ on average only 10 would transmit a packet to $\mathcal{N}_5$. Hence, allowing $\mathcal{N}_5$ to recover the generation. By selecting a tuple $x \in [1, m^*]$ the amount of surplus packets sent to $\mathcal{N}_5$ are greatly reduced. The selection of m^* depends on the number of transmission policies in the costs list C *i.e.,* the size of the list of cost function $|C|$, which is a function of the number of neighbors $|A_i|$ of the node, the number of generations in a the decoding region $\frac{t_b}{G_t}$ and the number of substreams S in the media stream. The value of m^* is therefore bounded by $1 \leq m^* \leq |C|$. If m^* is selected closer to the lower bound *i.e.,* 1, the scheduler would select π^1 as the transmission policy that may result in delivery of surplus packets, as explained earlier. However, if m^* is selected closer to the upper bound *i.e.,* $|C|$, the selection of the scheduler would more likely be towards a random-push scheduler. In the experimental section we experiment with different values of m^* to find the optimal value and discuss in more detail.

C. Synchronous Distributed Scheduler - SDS

The Synchronous Distributed Scheduling algorithm (SDS) solves (7) by exploiting unique identifiers of the nodes in the network to cope with the issue of concurrent optimizations. We recall that i is the integer number that uniquely identifies node $\mathcal{N}_i$ in the network. The algorithm is executed each time node $\mathcal{N}_i$ is granted a transmission opportunity and is described in pseudo-code as Algorithm 2. The algorithm first computes a list of all the possible scheduling policies compatible with constraints (5) and (6) and sorts the list in increasing C cost. We indicate the x-th policy from the top of the list as $\pi^x = \{j^x, (t,s)^x, Z^x, C^x\}$, so that π^1 indicates the policy at the top of the list and so on. Variable σ is initialized to zero at the first iteration of the algorithm and is updated at the end of each iteration of the algorithm while cycle.

At the first iteration, we have $x = 1$, so node $\mathcal{N}_i$ tests the condition $i \leq \sigma + \lceil Z^1 \rceil$ and checks if it is enabled to transmit a packet to $\mathcal{N}_j^1$ for generation $(t,s)^1$. If the condition is met, the algorithm returns π^1 as the optimal policy and $\mathcal{N}_i$ transmits a packet for generation $(t,s)^x$ to $\mathcal{N}_{j^x}$. Otherwise, the algorithm updates the variable $\sigma = \sigma + \lceil Z^1 \rceil$, x is incremented by one and the algorithm iteration. At the second iteration, we have $x = 2$ so the algorithm tests the condition $i \leq \lceil Z^1 \rceil + \lceil Z^2 \rceil$ and checks if it is allowed to transmit a packet to $\mathcal{N}_j^2$, hence selecting policy π^2. Again, if the check fails, σ is updated as $\sigma \leftarrow \sigma + \lceil Z^2 \rceil$ and the algorithm iterates again until the condition $i \leq \sum_{q=1}^{x} \lceil Z_q \rceil$ is met at some iteration. By imposing this condition, the maximum number of network nodes that will concurrently transmit a packet to $\mathcal{N}_{j^x}$ for generation $(t,s)^x$ is upper bounded by Z^x, which is the number of packets required by N_{j^x} to recover (t,s).

To clarify the issue addressed by the SDS algorithm and how the algorithm solves it, let us consider as an example a fully connected graph composed of $N = 17$ nodes where the links are affected by an average packet losses rate of 10% as illustrated (in part) in Figure 4. For the sake of simplicity, we assume that the stream is composed by just one generation (*i.e.,* , $T = 1$ and $S = 1$) of $k = 100$ symbols and the playback deadline of the nodes minus the delay is 1 (*i.e.,* , $h_{j,t} - d_{i,j} = 1 \forall i, j$), which allows us to drop the substream and generation appendices from the rank notation. Let us assume that nodes $\{\mathcal{N}_1, \ldots, \mathcal{N}_{15}\}$ (*transmitters*, in the following) have already recovered the generation, whereas nodes $\mathcal{N}_{16}$ and $\mathcal{N}_{17}$

Algorithm 2 Synchronous Distributed Scheduler - SDS

1: list all feasible scheduling policies $\pi = \{j, t, s, Z, C\}$
2: sort list by increasing cost C
3: $\sigma \leftarrow 0$; $x \leftarrow 1$
4: **while** $true$ **do**
5: **if** $i \leq \sigma + \lceil Z^x \rceil$
6: update expected rank of $\mathcal{N}_{j^x}$
7: **return** optimal policy π^x
8: **end if**
9: $\sigma \leftarrow \sigma + \lceil Z_i^x \rceil$; $x \leftarrow x + 1$
10: **end while**

(*receivers*) have not recovered the generation yet. Node $\mathcal{N}_{16}$ has rank $r_{16} = 96$, so it requires $k - r_{16} = 4$ packets to recover the generation; node $\mathcal{N}_{17}$ has rank $r_{17} = 92$, so it requires 8 packets to recover the generation. Let us assume that the exact decoding status of all the network nodes is known at the transmitters and that a transmission opportunity arises at the same time for all transmitters. For each transmitter, the figure shows the set of transmission policies (just two policies in the list, in this example) listed in increasing time-weighted cost (3) order. At all transmitters, $\mathcal{N}_{16}$ is the receiver with lowest associated decoding cost, so all 15 transmitters would send a packet to it. Therefore, $\mathcal{N}_{16}$ would recover the generation but a total of 15-4 = 9 transmission opportunities would be wasted transmitting surplus packets, while $\mathcal{N}_{17}$ would not be able to recover the generation.

Conversely, the SDS algorithm avoids a suboptimal allocation of the available transmission bandwidth operating as follows. We recall that each node is identified by the integer number i, that the iterations of the algorithm are numbered using the variable x and in the following the cost of the x-th policy from the head is indicated as Z^x. At the first iteration ($x = 1$), the condition $i \leq \sigma + \lceil Z_i^{(x)} \rceil$ is evaluated at each transmitter to decide whether the node shall send the packet to $\mathcal{N}_{16}$, which is at the head of the list. The check at line 5 of Algorithm 2 results true only at nodes $\{\mathcal{N}_1, \ldots, \mathcal{N}_5\}$, so these five transmitters send one packet each to $\mathcal{N}_{16}$ and the algorithm terminates. Accounting for the losses on the links, the total number of packets transmitted to N_{16} by $\{\mathcal{N}_1, \ldots, \mathcal{N}_5\}$ is 5;one greater than $k - r_{16} = 4$, so N_{16} is expected to recover the generation. The check at line 5 of Algorithm 2 is instead false at transmitters $\mathcal{N}_6 \ldots \mathcal{N}_{15}$, so at these nodes no transmission occurs and the algorithms iterates. At the second iteration (only transmitters $\mathcal{N}_6 \ldots \mathcal{N}_{15}$), we have $x = 2$ and $\sigma = Z^1$, so the check is true at all transmitters, which send a packet each to $\mathcal{N}_{17}$ and the algorithm terminates. Accounting for the losses on the links, the expected number of packets received by N_{17} is $9 > 8$, so N_{17} is expected to recover the generation too.

D. Considerations on Computational Complexity

Finally, we briefly discuss the computational complexity of the algorithms described above. At each transmission opportunity, a node invokes the algorithm, whose complexity largely lies in the sorting of the list of transmission policies,

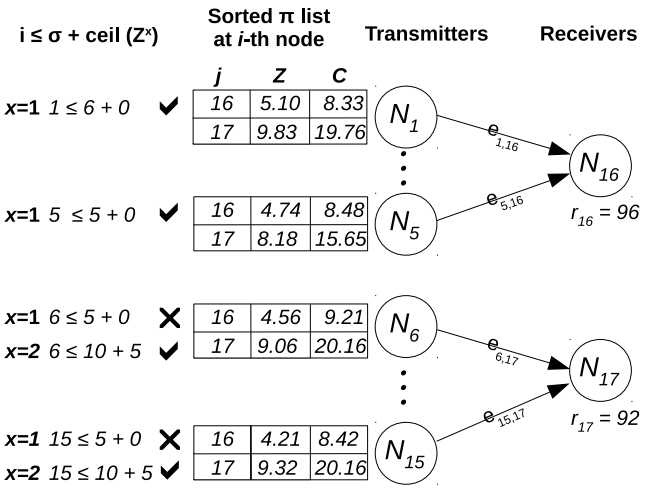

Fig. 4. Illustration of the SDS scheduler operations for a toy network: both receivers are able to recover the generation thanks to a correct allocation of the transmission opportunities.

which in turn depends on the list length. The number of transmission policies depends on the number $|A_i|$ of neighbors of the node, the number of generations in a the decoding region $\frac{t_b}{G_t}$ and on the number of substreams S in the media stream and is upper bounded by $|A_i| \frac{t_b}{G_t} S$. Such parameters drive however not only the complexity of the scheduler, but also the quality of the media recovered at the nodes. For example, lower t_b reduces the scheduler complexity, but it also reduces the time available to recover a generation and hence impairs the expected media quality. Our experiments revealed however that the probability that a random-push scheduler transmits surplus packets for a generation increases with the rank of the generation. Therefore, we argue that the complexity associated with the proposed scheduling algorithms can be reduced with little penalty if the nodes allocate some transmission opportunities to distribute packets for the generation currently seeded by the source, which is the most far from full rank, to random recipients in the network while optimized transmissions are reserved for generations close to full rank. The trade-off between performance and complexity of our schedulers is experimentally evaluated in the next section.

V. Experimental Results

In this section, we experiment with our robust scheduling algorithms streaming a network-coded media sequence to a set of cooperating nodes arranged in an unstructured graph with cycles where the links are affected by packet losses and delays. We measure both the quality of the media received by the network nodes and the efficiency of the packet scheduling scheme at allocating the limited upload bandwidth available in the network. The quality of the media recovered at the network nodes is measured in terms of *Continuity Index* (CI), *i.e.,* the fraction of generations timely recovered in the network. The bandwidth allocation efficiency is measured in terms of overhead with respect to the media encoding bandwidth, *i.e.,* as the extra bandwidth required to achieve a given CI with respect to the media bandwidth. In the following we

compare with a random-push reference scheduler (*RND*, in the following) similar to [9], where each network node transmits a packet to a random neighbor for a random generation each time a transmission opportunity arises. We consider two different network scenarios with the purpose to evaluate the resilience of our robust scheduling algorithms, respectively, to concurrent optimizations and imprecise feedback. Table II lists the notations used in the rest of this section.

N	Number of nodes in the network
$\mathcal{N}_i$	i-th network node, $i \in [1, N]$
$\|A_i\|$	$\mathcal{N}_i$ neighborhood size
RTT	Average round trip time between nodes
C_v	Encoding bandwidth of test media
C_s	Upload bandwidth of the source node
C_n	Upload bandwidth of the other nodes

TABLE II
KEY NOTATION USED FOR THE EXPERIMENTS.

A. Synchronous Network

We experiment streaming a network-coded media sequence to a set of cooperating nodes arranged into an unstructured graph with cycles where the nodes are deterministically granted transmission opportunities.

In the first experiment, we compare our distributed scheduler with a centralized oracle (*ORC*, in the following) that always knows the up-to-date decoding status of all nodes and selects the globally optimal scheduling policy solving problem 4. We stream a single-layer ($S = 1$) media sequence encoded at constant bitrate C_v = 500 kbit/s, where each generation is subdivided in $k = 100$ blocks of symbols and accounts for 1 Mbit of the media sequence, *i.e.,* each generations accounts for G_t = 2 seconds of media sequence. The test network is composed by N=100 nodes with constrained upload bandwidth C_n kbit/s and one source node that distributes encoded packets at a rate of $C_s = 2C_n$ kbit/s. The network links are affected by an average 10% packet loss rate and each node broadcasts an explicit feedback message every 10 media packets transmitted. Figure. 5 reports the CI measured at the network nodes as a function of the upload bandwidth C_n. The ORC scheme enables all nodes to seamlessly recover the media stream when the upload bandwidth is just enough to compensate the losses on the links, *i.e.,* $C_n = 1.1C_v$, and represents the upper bound to the media quality. The figure shows that SDS and ORC curves almost overlap, proving that our distributed framework enables nearly the same performance as the centralized counterpart. The figure also shows that the ADS scheduler achieves lower media quality than the SDS scheduler, albeit the two algorithms share the same theoretical framework. Our analysis showed that the ADS may waste the transmission opportunity when multiple nodes select the same policy at the same time due to the deterministic transmissions scenario, that the heuristic implemented in the SDS scheduler avoids instead. As a result, the ADS scheduler requires an upload bandwidth equal about 1.28 times the media bandwidth (*i.e.,* about 640 kbit/s in this setup) to achieve a CI close to 1. Finally, the RND scheduler achieves a much lower CI because it does not take into account the status of the nodes nor the links, and the nodes waste many transmission opportunities transmitting surplus packets (it requires C_n to be about 1.35 times C_v, *i.e.,* about 675 kbit/s, to achieve a CI close to 1).

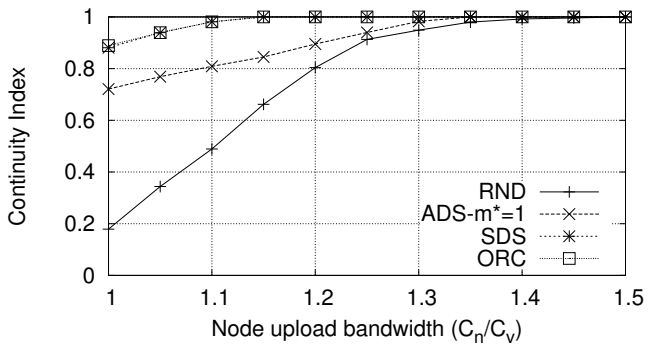

Fig. 5. Media quality as a function of upload bandwidth ($N = 100$, $|A_i| = N$). Our distributed scheduler performs nearly as well as a centralized scheduler reference.

In the second experiment, we evaluate the media quality at the network nodes as the number of nodes in the network N increases. As N increases, organizing the nodes in a fully connected graph becomes undesirable due to the associated signaling cost, which grows with the square of the neighborhood size $|A_i|$. Moreover the complexity of the scheduler also increases linearly with $|A_i|$. On the other hand a greater neighborhood size $|A_i|$ allows the scheduler to have complete knowledge of the decoding status of the network nodes to form an optimal scheduling policy. In this experiment, we limit each node to include in its neighborhood at most $|A_i| = \frac{N}{4}$ nodes drawn at random in the network and evaluate the media quality achieved. Each node independently selects the optimal policy at each transmission opportunity using the SDS scheduler, which the previous experiment showed to perform close to a centralized reference. The output bandwidth of the nodes is set to $C_n = 1.15\ C_v$, as the previous experiment showed being enough for the SDS scheme to achieve a CI close to 1. Figure. 6 shows the results of the experiments. The CI achieved by the RND scheduler never exceeds 0.7, while the SDS scheduler achieves a CI close to 1 and its performance is equivalent to that of the ORC scheduler for all N. The figure shows that even for large network sizes the SDS scheduler effectively avoids the transmission of surplus packets, resulting in a better bandwidth utilization and improved CI.

In the third experiment, we evaluate the trade-off between complexity and performance of our scheduling framework. In Section IV, we suggested that part of the transmission opportunities of a node can be reserved to distribute packets for the generation seeded by the source to reduce the number of times the scheduler is invoked, with little penalty in terms of media quality. Therefore, here we experimentally assess the minimum number of optimized transmissions that a node must afford before the media quality starts to degrade. Moreover, we also assess how much the neighborhood of a node $|A_i|$ can be shrunk before the media quality starts to degrade. We experiment with the SDS scheduler in the

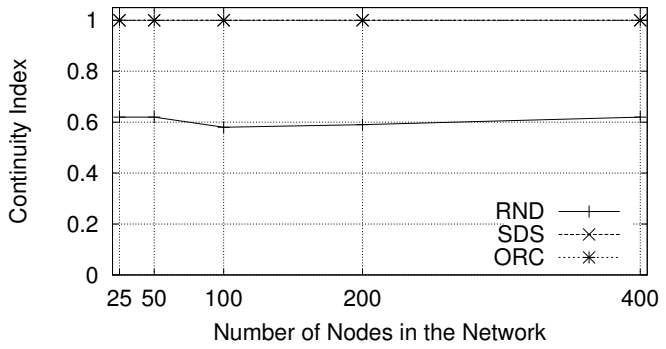

Fig. 6. Media quality as a function of the number of network nodes N (upload bandwidth $C_n = 1.15\ C_v$).

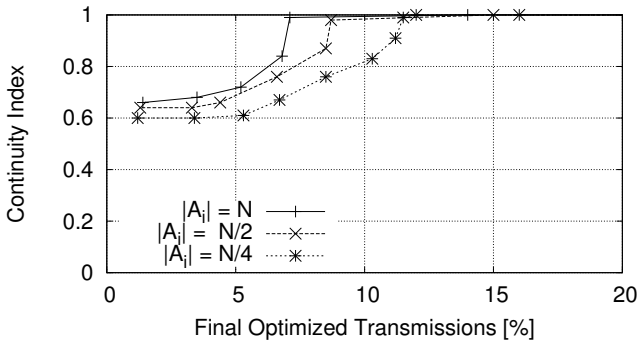

Fig. 7. Media quality as a function of the number of optimized transmissions for different neighborhood sizes $|A_i|$.

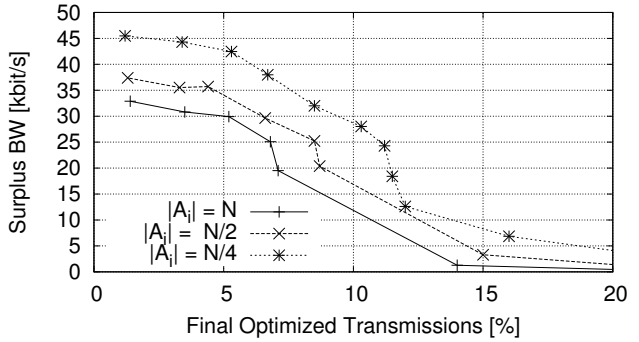

Fig. 8. Bandwidth wasted as a function of the final optimized transmissions for different neighborhood sizes $|A_i|$.

same conditions as for the experiments reported in Figure. 5 and set $C_n = 1.15\ C_v$. Figure. 7 shows the results of the experiment. When the number of optimized transmissions is 0, all nodes transmit packets for the generation currently seeded by the source, many transmission opportunities are wasted transmitting surplus packets and hence the SDS scheduler performs similarly to the RND reference in Figure. 5. As the number of optimized transmissions increases, the nodes waste fewer and fewer transmission opportunities transmitting surplus packets and so the CI increases. When the number of optimized transmission further increases between 7% and 13% of the total, the network nodes are able to seamlessly recover the media. Figure. 8 represents the amount of bandwidth wasted by transmitting surplus packets as a function of the optimized transmissions. The figure confirms that with the increase in the number of optimized transmissions, the amount of surplus bandwidth drops rapidly. In Figure. 8 at 7% and 13% of optimized transmissions, there is still some surplus bandwidth while in Figure. 7 the CI achieved by optimizing 7% and 13% of the transmissions is 1 for $|A_i| = N$ and $|A_i| = \frac{N}{4}$, respectively. This is because the node bandwidth $C_n = 1.15\ C_v$ is set slightly higher than the theoretical limit to cope with 10% of packet losses. If the number of optimized transmissions is increased further the surplus bandwidth finally becomes negligible. Moreover the amount of surplus bandwidth when $|A_i| = \frac{N}{4}$ is slightly higher than the case when $|A_i| = N$, this is because of two reasons i)when $|A_i| = N$, the nodes have knowledge of the decoding status of all the nodes in the network that helps to form the scheduling policies optimally and ii) when the neighborhood size $|A_i|$ is decreased from N to $\frac{N}{4}$, the probability of transmitting surplus packets increases with the decrease in the amount of optimized transmissions; this trend is also shown in the figure that as the number of optimized transmissions is increased beyond 12% the difference in the amount of surplus bandwidth for $|A_i| = N$ and $|A_i| = \frac{N}{4}$ is decreased greatly. The experiments show that the our scheduling framework enables seamless media recovery just by optimizing a small fraction of the total transmissions, hence reducing the scheduler impact on the node complexity.

In this experiment, we stream a two layer (S=2) scalable coded media sequence where the base layer is encoded at 375 kbit/s and the enhancement layer is encoded at 125 kbit/s (that is, C_v = 500 kbit/s), yielding a base layer of k_1=75 blocks of symbols per generations and an enhancement layer of k_2=25 blocks per generation. We experiment with the SDS scheduler and compare the performance with the reference RND scheduler: as the base layer accounts for 75% of the media encoding bandwidth, the RND scheduler allocates 75% of the node upload bandwidth C_n to the base layer and the rest to the enhancement layer. Figure. 9 reports the CI as a function of $\frac{C_n}{C_v}$ for both layers (enhancement layer generations are accounted as recovered only if the corresponding base layer generation is recovered as well). The figure includes the RND and SDS curves for a single stream media as reported in Figure. 5 to assess the benefits of scalable coding over single-stream coding. The RND reference fails to exploit the advantages offered by scalable coding: not only RND is unable to deliver full quality media to all network nodes (EL-RND), but the CI achieved for the base layer (BL-RND) is just slightly more than the CI achieved by the single stream media (RND). Conversely, our scheduler exploits the rate-adaptation properties of layered media to deliver reduced quality media when the available bandwidth is scarce. When the available bandwidth is enough to compensate the packet losses ($\frac{C_n}{C_v} > 1.1$), our scheduler (EL-SDS) delivers better CI than a random push scheduler distributing a single layer media (RND). When the upload bandwidth is not enough to compensate the packet losses ($\frac{C_n}{C_v} < 1.1$), the EL-SDS curve remains below the SDS curve because, due to smaller generations ($k_2 < k_1 < k$), the network code is less efficient and it takes more bandwidth to achieve identical CI. Despite the

lower network code efficiency, almost all nodes are however able to recover a low-quality version of the media (BL-SDS) even when the upload bandwidth is not enough to account for the packet losses ($\frac{C_n}{C_v} < 1.1$).

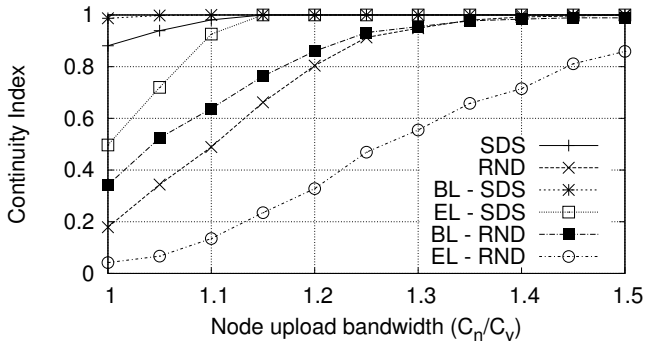

Fig. 9. Media quality as a function of the upload bandwidth for scalable media streams.

Finally, we consider the case of a media sequence encoded as two independently decodable descriptions where each description is encoded at 250 kbit/s (*i.e.,* C_v = 500 kbit/s), yielding generations of $k_1 = k_2$ = 50 blocks. The reception of any description allows a node to recover a base quality version of the generation, whereas reception of both generation allows the node to recover a full quality media. As in the previous experiment with scalable coded media, we experiment with the SDS algorithm and the RND reference and we compare with the single description sequence reported in Figure. 5. Since each description is of equal importance to the end of recovering the full quality sequence, in this experiment the RND scheduler evenly allocates the upload bandwidth to the two descriptions. Figure. 10 shows the media quality at the nodes as a function of $\frac{C_n}{C_v}$ and its analysis demonstrates similar findings to the results of the previous experiments with scalable coded media. The RND reference hardly allows the nodes to achieve a CI close to 1 for both descriptions (2D-RND), and the CI achieved by the nodes that recover at least one description (1D-RND) is systematically lower than the CI achieved for the single stream media (RND). On the contrary, our scheduler allows all network nodes to achieve a CI of 1 for at least one description (1D-SDS), even when the upload bandwidth is not sufficient to compensate the losses on the network. Moreover, it delivers full quality media (2D-SDS) to all nodes, despite smaller generations, only with a slight increase in the bandwidth required to deliver full quality single description reference (SDS).

B. P2P Media Streaming

The second scenario we consider is a P2P media streaming using the ToroStream protocol [4]. We briefly overview the the protocol. The network nodes are organized into an unstructured graph by a central *tracker*. The tracker maintains a list of all the nodes in the network and listens to join requests from the nodes. The tracker replies to the join requests by sending a list of nodes selected randomly. Then a handshake procedure starts between the nodes to become neighbors. After every 10 seconds each node removes some of its neighbors and a new

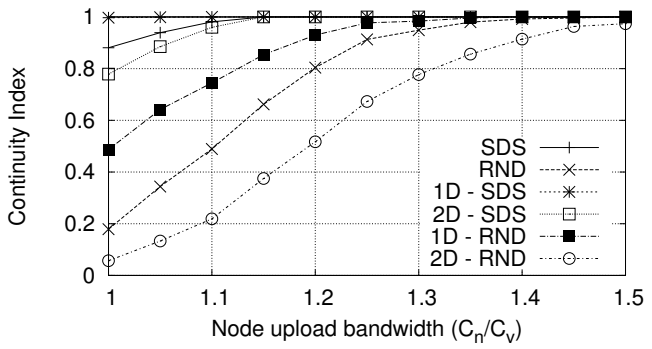

Fig. 10. Media quality as a function of the upload bandwidth for a multiple descriptions encoded sequence.

list of neighbors is assigned to it by the tracker. Moreover if a node does not receive any packet (data or keepalive message) from its neighbor for a given amount of time it is removed from the neighborhood and is considered offline. The links connecting the nodes are affected by an average 10% packet loss rate and each node updates its neighbors about its decoding status and the measured packet loss rate on the incoming links every 1 second broadcasting an explicit feedback message. Moreover, each node keeps in its buffer up to 10 generations that have already been played, in order to help its neighbors that are behind playback to recover the generations.

Differently from the previous scenario, the uncoordinated nature of P2P networks mitigates the problem of concurrent optimizations. However, the limited feedback precision due to packet losses and delays on the links and the churning of the network nodes is a challenge for our scheduling framework. Therefore, we modified the ToroStream so that, at each transmission opportunity, each node selects the transmission policy using the ADS scheduling algorithm.

As in the previous experiments, the source node streams a 5 minutes long media sequence encoded at C_v = 500 kbit/s subdivided in generations of k=100 blocks for a generation time of G_t = 2 seconds. The initial buffering time of the network nodes t_b is equal to 6 seconds, *i.e.,* the decoding region of each node encompasses 3 generations of media. Our reference is the same random-push scheduler (RND) considered in the previous experiments that transmits a packet to a random node for a random generation at each transmission opportunity.

In the first experiment, we consider a network composed of N=100 nodes where each node has a neighborhood size $|A_i|$=50 nodes. We experiment with the ADS scheduler for different values of the m^* parameter $m^* \in [1, 10, 30]$ and compare with the RND reference. The logs of the experiment show that the average value of the size of costs list is $|C| = 47.9$. Thus the value of m^* is bounded by $1 \leq m^* \leq 47$. Figure. 11 shows the media quality at the nodes as a function of $\frac{C_n}{C_v}$. The ADS scheduler performs better with $m^* = 10$ as compared with other value of m^*. The performance of ADS-$m^* = 1$ is comparable to the performance of ADS-$m^* = 10$. However when $\frac{C_n}{C_v} = 1.10$, ADS-$m^* = 1$ selects π^1 as the transmission policy at all the nodes, that results in some

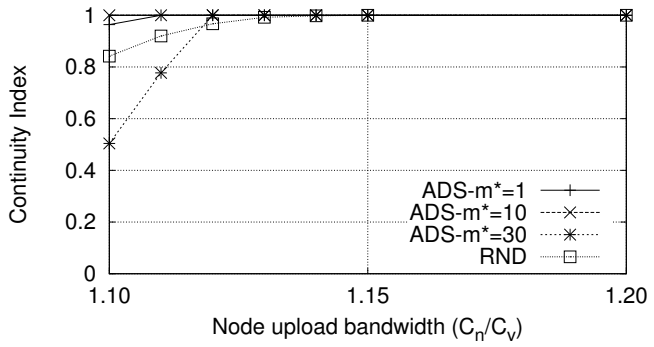

Fig. 11. Media quality as a function of the upload bandwidth ($N = 100$, $|A_i| = 50$).

surplus packets and a drop in the CI. Whereas if the value of m^* is increased to $m^* = 30$, initially the performance of the ADS scheduler drops very low for $C_n \in [1.10, 1.11]$. This is because when m^* is increased to a value close the $|C|$, the scheduler acts and performs towards a random-push scheduler, resulting in a inefficient bandwidth utilization. A comparison of the ADS-$m^* = 10$ and RND graphs shows that the ADS-$m^* = 10$ delivers better media quality than the RND reference and enables all nodes to recover the media sequence when the upload bandwidth is just sufficient to compensate the 10% packet loss rate on the links. The reason for better media quality lies in the improved efficiency in utilizing the available bandwidth, as illustrated in Figure. 12, which shows the bandwidth wasted for the transmission of surplus packets. For the RND reference, the surplus bandwidth increases with C_n and it amounts up to 50 kbit/s, *i.e.,* almost 10%, of C_n. Conversely, the ADS surplus bandwidth remains below 4 kbit/s, *i.e.,* it is about 1% of C_n and one order of magnitude smaller than for the RND reference.

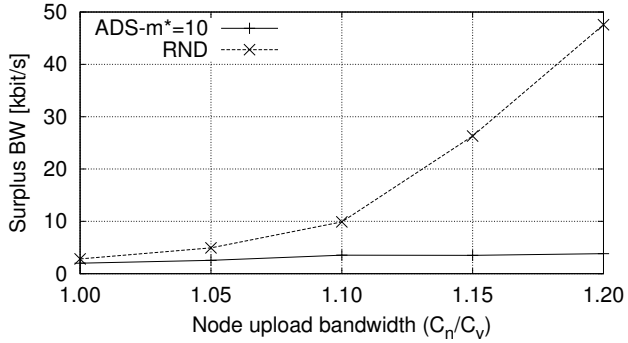

Fig. 12. Bandwidth wasted transmitting surplus packets (the lower, the better).

In the next experiment we evaluate the performance of the ADS and RND schedulers reducing the neighborhood size $|A_i|$ to 25 nodes, while using the same network settings as in the previous experiment. Since the previous experiment showed that setting $m^* = 10$ showed better results for the ADS scheme, again we select $m^* = 10$ for this and the following experiments. The benefits of small neighborhoods are smaller signaling bandwidth and lower scheduling complexity. On the other hand, a greater neighborhood size $|A_i|$ enables the ADS scheduler to form an optimal scheduling policy, because every transmitter node in this case has information about the decoding status of more nodes in the graph. Figure. 13 shows the CI as a function of the node bandwidth C_n. The figure shows that when $C_n = 1.10\ C_v$ the CI achieved by the ADS scheduler is about 0.98 while the RND scheduler achieves a CI of about 0.85. For the RND scheduler the minimum required node bandwidth C_n to achieve a CI of 1 is $C_n = 1.14\ C_v$. The experiment suggests that even when the neighborhood size is reduced the ADS scheduler gives a performance close to the ideal case, while also reducing the computations necessary for the sorting process.

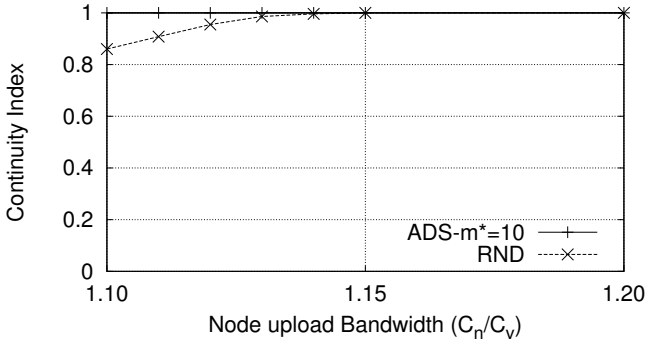

Fig. 13. Media quality as a function of upload bandwidth ($N = 100$, $|A_i| = 25$).

Next, we evaluate the performance of both schedulers with more constrained network conditions by setting $|A_i| = 25$ and increasing the round trip time to $RTT = 25ms$. With the introduction of some delay on the links all the packets exchanged among nodes get delayed. For the ADS scheduler this entails that the feedback parameters used for the packet scheduling optimization are not the true ones; therefore, this setting also tests the sensitivity of the scheduler to inaccuracy of the parameters. For the RND scheduler, increasing the delay should also affect the performance since the message sent by a recipient node when it decodes a generation is also delayed. Figure. 14 represents the performance results in terms of CI as a function of the node upload bandwidth in the network. When $C_n = 1.10\ C_v$, the ADS scheduler achieves a CI equal to 0.91 while the RND scheduler achieves a CI of about 0.84. A CI of 1 is achieved by the ADS scheduler at $C_n = 1.12\ C_v$, while for the RND scheduler it takes $C_n = 1.15\ C_v$ to achieve a CI of 1. The figure shows that even with the increase in RTT and smaller neighborhood size, the proposed ADS scheduler decodes more than 90% of the content when the node bandwidth C_n is just sufficient enough to cope with the packet losses (*i.e.,* $C_n = 1.10\ C_v$) and outperforms the reference RND scheduler by effective bandwidth utilization.

Moreover for the same network settings we experiment by varying amount of explicit keepalive messages for the ADS scheduler and evaluate the effect on the CI. We recall that apart from explicit keepalive messages every packet transmitted by the nodes also contain the node decoding information. Figure. 15 shows the media quality achieved as a function of $\frac{C_n}{C_v}$ for explicit feedback message frequencies of [500, 1000, 2000 and 4000]ms. The figure shows that, since the information

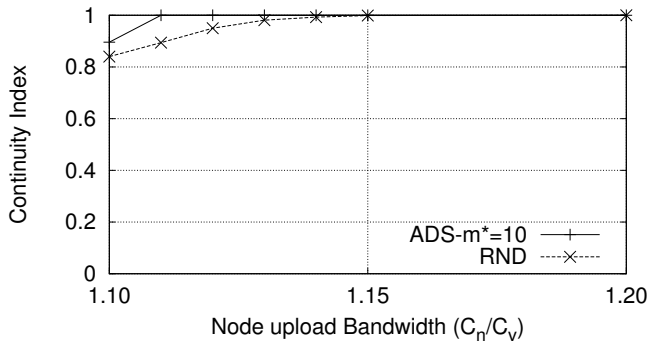

Fig. 14. Media quality as a function of upload bandwidth (RTT = 25 ms, $N = 100$, $|A_i| = 25$).

carried by the explicit feedback messages is also carried by all packets exchanged by the nodes, sending explicit feedback messages less frequently has little impact on the CI. The figure shows that the proposed ADS scheduler requires very little amount of explicit feedback information for optimization of the scheduling policies, resulting is less overhead.

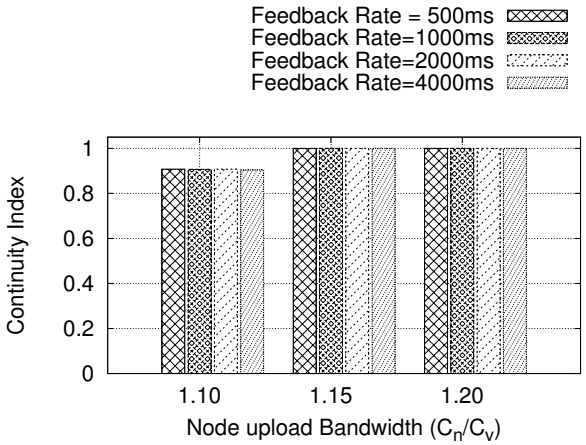

Fig. 15. Media quality by ADS-$m^* = 10$ as a function of upload bandwidth for different rates of explicit feedback messages (RTT = 25 ms, $N = 100$, $|A_i| = 25$). Reducing the amount of explicit feedback messages has little impact on CI.

Next, we experiment with a low-delay media streaming scenario by reducing the maximum allowed buffering time t_b of the nodes. A decrease in the buffering time is very desirable from the users perspective. However, achieving a high CI with reduced buffering time requires increased upload bandwidth due to the lower efficiency of the push mechanism. Moreover, reducing the buffering time requires to reduce k, which increases the encoding overhead k'/k. The experimental setup consists of the source node that streams to $N = 100$ nodes a media sequence with $B_v = 500kbps$ and corresponding $k = 25$ and generation time $G_t = 500ms$. We set the buffering time of the nodes equal to the generation time, *i.e* ($t_b = G_t = 500ms$); this means that the nodes only buffer one generation before playback of the stream; rather than 3 as in the previous experiments. Figure. 16 represents the results of the experiment in terms of CI as a function of the upload bandwidth of the node. With 10% average packet losses at $\frac{C_n}{C_v} = 1.10$, the CI achieved by the ADS scheduler is about 0.75, while the RND scheduler achieves a CI of about 0.70. To achieve a CI equal 1 the ADS scheduler requires an upload bandwidth ratio of $\frac{C_n}{C_v} = 1.32$, while the requirement for the RND scheduler is $\frac{C_n}{C_v} = 1.45$ to achieve a CI equal 1. The ADS scheduler requires about 10% less bandwidth than the RND scheduler to provide a continuous media playback. The figure shows that the ADS scheduler is suitable for low-delay communications and provides better media quality to the users with reduced buffering times in comparison with the RND scheduler.

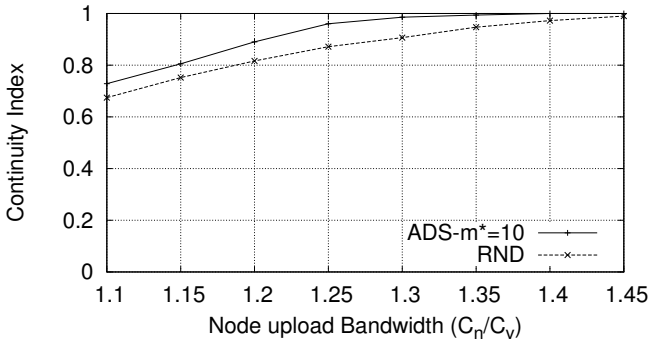

Fig. 16. Low-delay media streaming scenario with $C_t = t_b = 500ms$ ($N = 100$, $|A_i| = 50$).

Finally, we experiment a low-delay video-conferencing scenario using the Planetlab network [28]. Planetlab is a network composed of hundreds of Internet nodes in different locations and enables us to experiment using a real P2P network with delays, packet erasures and out-of-order delivery. We experiment using 25 Planetlab nodes in a fully connected graph and we further reduce the maximum allowed buffering time t_b of the nodes to $250ms$. The source node streams a media sequence encoded at $C_v = 500kbps$, which yields a generation size of $k = 13$. Each node sends an explicit feedback message (*i.e.,* a keepalive message) containing its updated decoding status to its neighboring nodes each second. We set the bandwidth of the source node $C_s = 2.5Mbps$ thus on average each node receives 20% of the media content from the source node. Figure. 17 shows the media quality achieved at the nodes as a function of $\frac{C_n}{c_v}$ for the ADS and the RND schemes. Due to the packet losses on the Internet and out of order delivery of the messages, the bandwidth required to achieve a CI close to 1 has increased with respect to the previous experiment in Figure. 16. At $\frac{C_n}{C_v} = 1.88$ the ADS scheduler allows the nodes to recover about 96% (*i.e.,* CI = 0.96), while the nodes using the RND scheduler require $\frac{C_n}{C_v} = 3.3$ to achieve a similar media quality, thus the ADS scheduler requires less bandwidth than the RND scheduler to provide the same media quality. The experiment suggests that on a real network with more tight network parameters of reduced buffering time and reduced network size the ADS scheduler still outperforms the RND scheduler, resulting in a improved user experience and better bandwidth utilization than the RND scheduler.

VI. Conclusions and Future Work

In this paper, we target the packet scheduling problem for media streaming with NC in unstructured random graphs. We

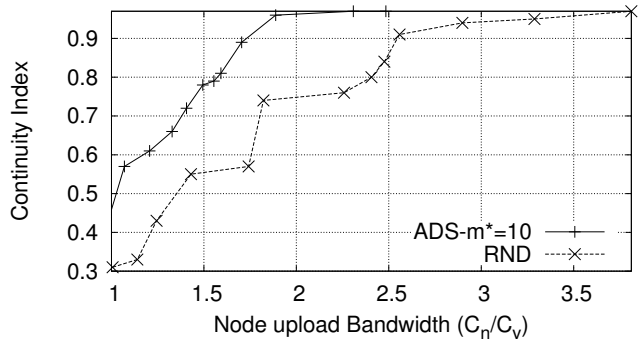

Fig. 17. Media quality achieved on the Planetlab network towards a video-conferencing scenario with $G_t = t_b = 250ms$ and $N = 25$.

formulated the packet scheduling problem as an optimization model to maximize the media quality delivered to the nodes using a push-based approach. The model utilizes the decoding status of the nodes to estimate the amount of packets required by the node to be able to decode a generation, taking into account the decoding overhead, packet loss probability and delay on the links. To solve the problem we propose two algorithms that are robust to imprecise and infrequent feedback information from the recipient nodes, and solve the optimization problem in linear time. We perform numerous experiments with the proposed algorithms and a reference random-push scheduler in different network scenarios and conditions including packet losses and delays on the links. Our experiments, including real-world tests using PlanetLab, with the proposed algorithms and a reference random-push scheduler show that the proposed algorithms provide improved user experience in terms of media quality, delay and bandwidth requirements, while only using little amount of feedback information from the recipient nodes.

REFERENCES

[1] C. Fragouli, J. Le Boudec, and J. Widmer, "Network coding: an instant primer," *Computer Communication Review*, vol. 36, no. 1, p. 63, 2006.

[2] R. Ahlswede, N. Cai, S.-Y. Li, and R. Yeung, "Network information flow," *Information Theory, IEEE Transactions on*, vol. 46, no. 4, pp. 1204–1216, 2000.

[3] E. Magli, M. Wang, P. Frossard, and A. Markopoulou, "Network coding meets multimedia: A review," *Multimedia, IEEE Transactions on*, vol. 15, no. 5, pp. 1195–1212, 2013.

[4] A. Fiandrotti, A. M. Sheikh, and E. Magli, "Towards a p2p videoconferencing system based on low-delay network coding," in *Proceedings of the 20th European Signal Processing Conference (EUSIPCO)*, 2012, pp. 1529 –1533.

[5] A. M. Sheikh, A. Fiandrotti, and E. Magli, "Distributed media-aware scheduling for p2p streaming with network coding," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, 2013, pp. 3597–3601.

[6] M. Wang and B. Li, "Lava: A reality check of network coding in peer-to-peer live streaming," in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, 2007, pp. 1082–1090.

[7] M. Wang and B. Li, "Network coding in live peer-to-peer streaming," *Multimedia, IEEE Transactions on*, vol. 9, no. 8, pp. 1554–1567, Dec 2007.

[8] P. Chou, Y. Wu, and K. Jain, "Practical network coding," in *Proceedings of the Annual Allerton Conference on Communication Control and Computing*, vol. 41, no. 1, 2003, pp. 40–49.

[9] M. Wang and B. Li, "R2: Random push with random network coding in live peer-to-peer streaming," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 9, pp. 1655 –1666, december 2007.

[10] C. Feng and B. Li, "On large-scale peer-to-peer streaming systems with network coding," in *Proceedings of the 16th ACM International Conference on Multimedia*, ser. MM '08. New York, NY, USA: ACM, 2008, pp. 269–278. [Online]. Available: http://doi.acm.org/10.1145/1459359.1459396

[11] Z. Liu, C. Wu, B. Li, and S. Zhao, "Uusee: Large-scale operational on-demand streaming with random network coding," in *INFOCOM, 2010 Proceedings IEEE*, 2010, pp. 1–9.

[12] L. Yu, L. Gao, J. Zhao, and X. Wang, "Sonicvod: A vcr-supported p2p-vod system with network coding," *Consumer Electronics, IEEE Transactions on*, vol. 55, no. 2, pp. 576–582, May 2009.

[13] S. Annapureddy, S. Guha, C. Gkantsidis, D. Gunawardena, and P. R. Rodriguez, "Is high-quality vod feasible using p2p swarming?" in *Proceedings of the 16th International Conference on World Wide Web*, ser. WWW '07. New York, NY, USA: ACM, 2007, pp. 903–912. [Online]. Available: http://doi.acm.org/10.1145/1242572.1242694

[14] N. Cleju, N. Thomos, and P. Frossard, "Selection of network coding nodes for minimal playback delay in streaming overlays," *Multimedia, IEEE Transactions on*, vol. 13, no. 5, pp. 1103–1115, 2011.

[15] N. Thomos, J. Chakareski, and P. Frossard, "Prioritized distributed video delivery with randomized network coding," *Multimedia, IEEE Transactions on*, vol. 13, no. 4, pp. 776–787, Aug 2011.

[16] J. Zhao, F. Yang, Q. Zhang, Z. Zhang, and F. Zhang, "Lion: Layered overlay multicast with network coding," *Multimedia, IEEE Transactions on*, vol. 8, no. 5, pp. 1021–1032, 2006.

[17] K. Nguyen, T. Nguyen, and S.-C. Cheung, "Peer-to-peer streaming with hierarchical network coding," in *Multimedia and Expo, 2007 IEEE International Conference on*, 2007, pp. 396–399.

[18] D. Vukobratovic and V. Stankovic, "Unequal error protection random linear coding for multimedia communications," in *Multimedia Signal Processing (MMSP), 2010 IEEE International Workshop on*, 2010, pp. 280–285.

[19] N. Thomos and P. Frossard, "Raptor network video coding," in *Proceedings of the International Workshop on Workshop on Mobile Video*, ser. MV '07. New York, NY, USA: ACM, 2007, pp. 19–24. [Online]. Available: http://doi.acm.org/10.1145/1290050.1290056

[20] N. Thomos and P. Frossard, "Network coding of rateless video in streaming overlays," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 20, no. 12, pp. 1834–1847, 2010.

[21] N. Sundaram, P. Ramanathan, and S. Banerjee, "Multirate media stream using network coding," in *in Proc. 43rd Annual Allerton Conference on Communication, Control, and Computing*, 2005.

[22] A. T. Nguyen, B. Li, and F. Eliassen, "Chameleon: Adaptive peer-to-peer streaming with network coding," in *INFOCOM, 2010 Proceedings IEEE*, 2010, pp. 1–9.

[23] A. K. Ramasubramonian and J. W. Woods, "Video multicast using network coding," *Proc. SPIE 7257, Visual Communication and Image Processing 2009*, vol. 7257, pp. 72 570E–72 570E–11, 2009. [Online]. Available: http://dx.doi.org/10.1117/12.805588

[24] A. K. Ramasubramonian and J. W. Woods, "Md/pnc with feedback for heterogeneous video multicast in lossy networks," *Proc. SPIE 7882, Visual Information Processing and Communication II*, vol. 7882, pp. 78 820M–78 820M–10, 2011. [Online]. Available: http://dx.doi.org/10.1117/12.872897

[25] L. Cui, Y. Jiang, and J. Wu, "Optimizing push scheduling algorithm based on network coding for mesh peer-to-peer live streaming," in *Communications (ICC), 2012 IEEE International Conference on*, 2012, pp. 2075–2080.

[26] M. Toldo and E. Magli, "A resilient and low-delay p2p streaming system based on network coding with random multicast trees," in *Proceedings of the IEEE International Workshop on Multimedia Signal Processing (MMSP)*, 2010, pp. 400–405.

[27] K.-H. Chan, S.-H. Chan, and A. Begen, "Spanc: Optimizing scheduling delay for peer-to-peer live streaming," *Multimedia, IEEE Transactions on*, vol. 12, no. 7, pp. 743 –753, nov. 2010.

[28] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "Planetlab: an overlay testbed for broad-coverage services," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 3, pp. 3–12, 2003.