Joint Scheduling and Network Coding for Multicast in Delay-Constrained Wireless Networks

Ketan Rajawat, *Student Member, IEEE*, and Georgios B. Giannakis, *Fellow, IEEE*

***Abstract*—This paper deals with network-coded multicast for real-time and streaming-media applications where packets have explicit expiration deadlines. Most of the popular network coding approaches require asymptotically large block-lengths, thereby incurring long decoding delays. The present paper introduces a joint scheduling and network coding design that aims to maximize the average throughput while respecting the packet deadlines. The novel approach relies on a time-unwrapped graph expansion in order to construct the network codes. The resultant algorithm draws from the well-known augmenting-path algorithm, and is both distributed as well as scalable. For networks with primary interference, a lower-bound on the worst-case performance of the algorithm is provided. The associated optimization problem is also analyzed from an integer programming perspective, and a set of valid inequalities is derived to obtain an upper bound.**

***Index Terms*—Cross-layer designs, delay-sensitive networks, network coding.**

I. Introduction

NETWORK CODING (NC) is a packet-level coding technique that generalizes the classical routing paradigm [1]. The simplest of these coding schemes, called linear NC, is in fact throughput optimal for single-source multicast in wired networks [2]. This optimality result has encouraged the extension of linear NC to diverse areas including distributed storage, streaming media, network monitoring, *ad hoc* topologies and sensor networks [3], [4]. Wireless applications in particular, offer new possibilities for improvement while posing many unique challenges. For instance, joint design of NC and scheduling promises throughput gains and energy savings by utilizing the redundancy and spatial diversity inherent to wireless networks [5]. However, optimum scheduling is itself a difficult task that has prompted researchers to design several approximate and near-optimal designs; see, e.g., [6]–[8], and references therein. Joint designs have also been considered within general cross-layer optimization [9] and back-pressure [10] frameworks.

Manuscript received October 08, 2010; revised May 30, 2011; accepted August 01, 2011. Date of publication August 15, 2011; date of current version November 16, 2011. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Gerald Matz. This work was supported by the NSF by Grants CCF-0830480 and ECCS-0824007. The material in this paper was presented in part at the International Workshop on Wireless Network Coding, Rome, Italy, June 2009.

The authors are with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455, USA (e-mail: ketan@umn.edu; georgios@umn.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSP.2011.2165061

An important, but often overlooked, aspect of several wireless applications is the sensitivity of packets to delays. Streaming media and real-time sensor data, for example, are associated with strict deadlines, failing which, packets become useless. However, many wireless NC implementations, such as [11], operate under the assumption of large block-lengths. This requires the sinks to accumulate a large number of packets before commencing the decoding process, thereby incurring prohibitively large delays.

This paper develops a joint scheduling and NC (JS-NC) algorithm for wireless networks with packet delay constraints. A single source multicast scenario is considered where packets must be decoded at each sink within a specified number of timeslots since their first transmission by the source node. Delay constraints significantly complicate the JS-NC design since the optimal codes may have infinite block-lengths; see [12] and other references in Section II.

Since infinite block-length codes are difficult to design as well as implement, Section III proposes a simpler periodic version of this joint design problem that operates on a time-unwrapped graph, thus allowing for finite block-length NC. The *periodic formulation* is employed to derive a constant-factor approximate, augmenting-path design algorithm that is both scalable and distributed, as shown in Section V. The resultant NC protocol does not require any end-to-end feedback or asymptotically large field size, and needs only a brief setup time.

For networks with primary interference constraints, Section VI analyzes the JS-NC design problem from an integer programming (IP) perspective. A set of valid inequalities is developed which is subsequently used to derive a linear programming upper bound on the achievable throughput. Finally, Section VII presents simulated tests corroborating the performance of the approximate JS-NC algorithm and the quality of the associated bounds.

II. Related Work

The design of JS-NC schemes with delay constraints has not been addressed in literature. Several works though, have analyzed the delay performance gains of NC in single-hop scenarios [13]–[15]. Extension to multihop networks is nontrivial since the presence of scheduling constraints significantly complicates the solution.

A related problem in the context of wired networks has been analyzed in [12]. One major difference however is that [12] considers bit-by-bit transmission and NC. This results in a problem similar to that of determining the minimum finite field size for the given network. In packet networks though, field size is usually not the bottleneck.

Several heuristic NC schemes in media streaming applications are also available; see [16]–[18], and references therein. These do not design NC jointly with scheduling constraints, and focus primarily on implementation issues. Instead, the focus here is on joint designs and performance guarantees.

Recently, there has been an attempt to reduce queuing delays in back-pressure methods by modifying the Lyapunov function [19]. This may also result in reducing queuing delay in NC schemes that employ back-pressure. However, most of these methods also require large block-lengths, thereby rendering decoding delay a challenging bottleneck.

III. System Model

Consider a wireless network represented by a directed acyclic graph $G = (V, E)$, with V denoting the set of nodes and E the set of edges. The set E consists of tuples (u, v) denoting the two nodes that the edge connects. The network supports a multicast session consisting of a source node $s \in V$ that intends to transmit a packet-stream to each of the sink nodes $T \subset V$.

Linear NC is performed at intermediate nodes, which allows them to linearly combine and forward received packets. A block-NC model is assumed, wherein the packet stream is parsed into blocks before transmission. Subsequently, only packets belonging to the same block are allowed to be mixed. The sinks also decode the packets in a block-wise fashion; that is, upon receiving linear combinations of the packets belonging to each block.

The network operates in a time-slotted fashion, where one time slot carries one packet. The *deadline constraint* dictates that the sinks must be able to decode a block within D time slots of the transmission of the *first* packet from that block by the source. Further, the wireless interface imposes the following *scheduling constraints*:

SC1) The nodes adhere to a half-duplex operational mode; and
SC2) The nodes experience interference of either a) primary; or b) secondary nature.

The half-duplex constraint SC1) prevents a node from transmitting and receiving in the same time-slot. The primary interference PI) constraint SC2a), holds for orthogonal (i.e., channelized) access, e.g., via spreading codes or frequency division multiplexing. SC2a) allows each node to receive from at most one neighboring node per time slot; see, e.g., [20]. The secondary interference SC1) constraint SC2b) imposes additional restrictions: two links $(u_1, v_1) \in E$ and $(u_2, v_2) \in E$ cannot be used for transmission in the same time slot if either $(u_1, v_2) \in E$ or $(u_2, v_1) \in E$; see, e.g., [21]. Clearly, broadcast is allowed for both PI and SI types of constraints. Fig. 1 shows the key difference between the PI and SI constraints.

The aim is to find the maximum multicast throughput, given here by the rate (packets per time slot) at which the source transmits packets that reach all the sinks within the stipulated deadline. In this JS-NC framework, both the time slots at which each node transmits as well as the linear combinations it uses to code must be designed.

Throughput optimization with JS-NC design is well known to be difficult even without deadline constraint [6], [7]. Some approximate JS-NC designs are described in [20], [21] but cannot be extended to the deadlined case as they rely on using NC with large block lengths, and consequently incur long decoding delays. This consideration motivates the following operational assumption.

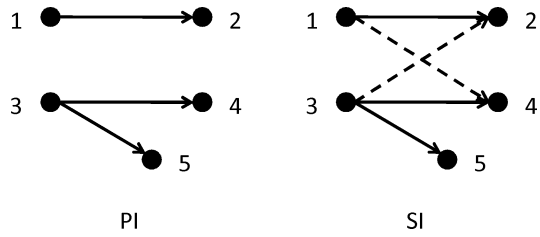

Fig. 1. The key difference between PI and SI constraints. Under the PI constraint, nodes 2 and 4 can simultaneously receive from transmitters 1 and 3. Under SI constraints however, the two transmitters interfere with reception at nodes 2 and 4, and should not be scheduled at the same time. Node 5 can receive from node 3 in both cases.

AS1) The source begins transmitting the next block of packets only after the previous block has been decoded at all sinks.

Together with the deadline constraints, AS1) implies that each block of packets stays in the network for at most D time slots. As a result, the goal reduces to that of finding the JS-NC design maximizing the number of packets that can be multicast to the sinks within the first D slots. The schedule and NC can both be reused for subsequent blocks, which also makes the transmission and reception patterns of the nodes periodic with finite period D. The next section describes a time-unwrapping technique used for solving this simplified problem.

Before concluding this section, a few remarks on the assumptions are due.

Remark 1: The block-decoding assumption at the sink nodes is not necessarily throughput optimal. This is because the sinks begin decoding only upon receiving the linear combinations corresponding to the entire block, resulting in long waiting-times for the first few packets of the block (i.e., decoding delay). An alternative is to use infinite block-length convolutional NC designs that allow for sequential decoding at the sinks [22]. However, designing infinite block-length codes that satisfy the deadline constraints is known to be difficult even in wired networks [12].

The use of only a single block per period, as implied by AS1), incurs an overhead. This is because the source is not allowed to transmit the next block until the last packet of the current block has been received at each sink. However, as shown in Section V, a solution obtained with this assumption, can be converted into a pipelined solution, that significantly reduces this overhead.

Remark 2: Compared to a back-pressure approach, the JS-NC design is not dynamic, i.e., the scheduling and NC decisions are not made on a per-packet basis and do not depend on the instantaneous channel conditions. Back-pressure schemes however, are well known to exhibit poor delay performance [19]. Further, most dynamic JS-NC algorithms require large block-lengths, resulting in prohibitive decoding delays [10]. On the other hand, the static JS-NC design proposed here offers flexibility to operate with a specified deadline D. The channel-oblivious nature of the design also makes it simpler, and easier to distribute relative to the dynamic designs.

Varying channel conditions always result in packets getting dropped or erased even in the absence of collisions. In delay-critical applications, it may not be possible to recover the lost packets at all, due to the extra time required for the sink to send feedback, and the source to retransmit. To a certain extent, erasures can be handled through classical forward error correction codes that are applied at the source node. Alternatively, specialized random network codes are available to correct packet-erasures; see e.g., [23], [24], and references therein. In cases when the number of erasures becomes too large, partial recovery may be acceptable, and can be provided through the use of priority-encoding and transmission (PET) [11]. Several practical PET designs have been proposed in the context of NC for video applications [25], [26].

IV. Time-Unwrapping and NC Design

Under AS1), the goal is to find the JS-NC design allowing the source to multicast the maximum number of packets to each sink within D slots. This section introduces the idea of "time-unwrapping" of a graph as a tool for JS-NC design. Time-unwrapping has been employed by time-slotted networks, e.g., to solve the quickest-flow problem [27], and in the context of NC designs over wired networks [12], [28]. As the name suggests, a time-unwrapped graph can be used to represent the entire transmission and coding schedule for a given number of time slots on a single graph. The proposed construction is similar but here it must adhere also to the scheduling constraints SC1)–SC2). Specifically, each node is first split into several functional subnodes, namely receiver-, combiner-, and transmitter-subnodes, before being replicated. The entire procedure proceeds in these steps.

- U1) Each node v is split into receiver-, combiner-, and transmitter-subnodes, and replicated D times. The subnodes corresponding to the kth time slot are denoted by $v^r(k)$, $v^c(k)$ and $v^t(k)$, respectively.
- U2) A directed edge (u,v) in the original graph is replaced by D directed edges $(u^t(1),v^r(1))$, $(u^t(2),v^r(2))$, and so on.
- U3) Since packets received in the current time slot are only available for transmission in the subsequent time slots, a subnode $v^r(k)$ is only connected to subnodes $v^c(k+1),\ldots,v^c(D)$.
- U4) Each combiner-subnode $v^c(k)$ is connected to its corresponding transmitter-subnode $v^t(k)$.
- U5) Finally, the source node s is modeled as a "wired" source-subnode s^ν connected to D transmitter-subnodes $s^t(1),s^t(2),\ldots,s^t(D)$, i.e., s has no receiver- and combiner-subnodes.
- U6) Similarly, the set of sink nodes T is modeled by a corresponding set of "wired" sink-subnodes $\mathcal{T}$. Each wired sink-subnode $t_i^\nu \in \mathcal{T}$, for $i=1,2,\ldots,|T|$, receives from D receiver-subnodes $t_i^r(1),t_i^r(2),\ldots,t_i^r(D)$.

Fig. 2 shows a time-unwrapped node. The overall time-unwrapped graph is denoted by $\mathcal{G}=(\mathcal{V},\mathcal{E})$, with $\mathcal{V}$ denoting the set of nodes and $\mathcal{E}$ the set of edges. Further, transmission on an edge of the form $(u^t(\ell),v^r(\ell))\in\mathcal{E}$ corresponds to transmission on the edge $(u,v)\in E$ at time slot ℓ. Similarly, two transmissions on G, that violate SC1)-SC2), give rise to a set of so-termed *conflicting* edges in $\mathcal{G}$. Thus, the entire operation (i.e., reception, combination, and transmission of packets) of the wireless network G over D time slots can be described using the time-invariant graph $\mathcal{G}$.

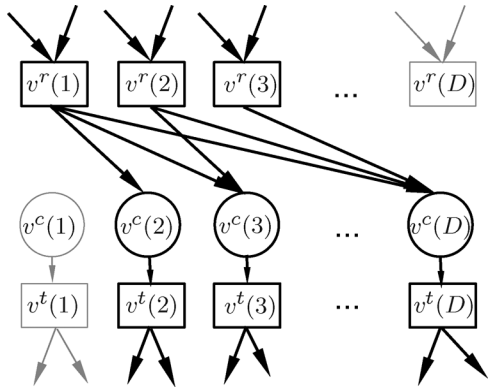

Fig. 2. A time-unwrapped node. Note that the first combiner- and transmitter-subnodes, and the Dth receiver-subnode are redundant.

Given a generic time-invariant graph, any NC design algorithm takes as input the sets of edge-disjoint paths from the source to each of the sinks. The additional constraint in the present case is that the edges on these paths must not conflict with each other. Given a set of μ edge-disjoint, nonconflicting, $s^\nu - t^\nu$ paths in $\mathcal{G}$ for each sink $t^\nu\in\mathcal{T}$, an NC can be designed using one of the methods in [29]. These algorithms return the coefficients used at each edge $e\in\mathcal{E}$ for linearly combining the μ packets. Formally, the vector of coefficients for edge e, referred to as the global encoding kernel, is given by $f(e)\in\mathbb{F}_q^\mu$, where $\mathbb{F}_q$ is the finite field of alphabet size q [1]. These global encoding kernels can be obtained using deterministic or randomized algorithms as those described in [29] and [30]. Indeed, if the field size $q\geq|T|$, randomly drawn kernels $\{f_e\}$ suffice with high probability, and will be used henceforth.

The linear combination provided for an edge $(u^t(\ell),v^r(\ell))$ may then be used on the edge $(u,v)\in E$ at time slot ℓ. All other edges are internal to the nodes and are only used to determine which packets need to be combined per time slot. The overall NC is therefore a list of global encoding kernels of the form $f'(e,\ell)$ for each $e\in E$ and $\ell=1,2,\ldots,D$. For convenience, the schedule at the ℓth time slot will be denoted using a graph $G_\ell=(V,E_\ell)$, where $(u,v)\in E_\ell$ if and only if the edge $(u^t(\ell),v^r(\ell))\in\mathcal{E}$ carries a nonzero encoding vector. The overall NC operation can therefore be viewed as the sequence of graphs namely, $G_1,G_2,\ldots G_D;G_1,G_2\ldots,G_D;G_1,\ldots$.

It should now be clear that the multicast throughput can be maximized by finding the largest possible value of μ such that there are as many nonconflicting, edge-disjoint, augmenting paths from the source s^ν to each sink $t_i^\nu\in\mathcal{T}$, and this is the focus of the next section. But before pursuing this direction, a few remarks pertaining to the time-unwrapping procedure are in order.

Remark 3: It can be seen that the proposed time-unwrapped graph itself takes care of the scheduling constraints partially. For instance, the graph does not allow any path from traversing through both $v^r(\ell)$ and $v^t(\ell)$. A packet received at time slot ℓ can only be sent at a later time-slot. Similarly, the combiner-

subnodes allow only one packet to be transmitted/broadcast per-time slot.

Remark 4: Since each packet traverses at most a single hop per-time slot, some transmitter and receiver-subnodes may be redundant. Examples include the first transmitter and combiner, and the last receiver-subnodes since only the source transmits in the first time slot, and only the sink receives in the last slot. These nodes can be removed to reduce algorithm complexity.

V. An Augmenting Path Approach

This section develops a greedy augmenting path (GAP) algorithm for maximizing the value of μ, the number of edge-disjoint, nonconflicting, augmenting paths from source s^ν to each sink $t^\nu \in \mathcal{T}$. A worst-case performance bound on the performance of GAP algorithm for PI networks is also established. The proposed algorithm can be viewed as an extension of the well-known Edmond-Karp algorithm [31, Ch. 26] for the wireless setting considered here.

In order to describe the GAP algorithm in detail, some graph-theoretic notions are introduced. A *flow* is an assignment of $\mathbb{F}_2$ (i.e., 0–1) values to the edges of the graph. A *valid flow* is one satisfying the flow conservation constraints; i.e., the total flow on the incoming and outgoing edges of a node should be the same. A *unit valid flow* is the assignment of 1 s along an *augmenting path*, defined as any directed source-sink path. Given a flow, the *residual graph* is obtained by reversing the direction of all edges with unit values.

Finding the maximum number of edge-disjoint augmenting paths, is equivalent to finding the maximum number of unit valid flows (the max-flow problem). The Edmond-Karp (EK) max-flow algorithm proceeds as follows:

EK0) Initialize flow values on all edges of graph $\mathcal{G}$ to zero;

EK1) Find the shortest augmenting (source-sink) path using, e.g., Dijkstra's algorithm [31, Ch. 24];

EK2) Increment the flow values along the path found in EK1);

EK3) Set $\mathcal{G}$ equal to the residual graph; and go back to EK1).

The idea of finding edge-disjoint augmenting paths via EK0)–EK3) can also be extended to the wireless setting, albeit with a modification. Specifically, after obtaining an augmenting path $\mathcal{P}$ in EK1), all other edges that conflict with any of the edges $e \in \mathcal{P}$ must be deleted from the residual graph obtained in EK3). This ensures that the augmenting paths found across iterations do not conflict with each other. Since edges are only being removed, any set of nonconflicting augmenting paths is also a feasible solution to the wired case.

In a nutshell, while repeating EK1)-EK3), constraints SC1)-SC2) can be respected by deleting conflicting edges till no more augmenting paths can be found. The modified EK algorithm however may not always find all the edge-disjoint augmenting paths because [31, Lemma 26.2] no longer applies. Intuitively, once an edge is deleted to obey SC1)-SC2), those augmenting paths that could contain it are not present in the output of the modified EK algorithm. On the other hand, the fact that EK1) explores *shortest* augmenting paths helps to reduce the number of conflicting edges deleted.

Further modifications of the EK algorithm are needed for extension to the case with multiple sinks; see Algorithm 1. In this case, the algorithm maintains $|T|$ copies $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_{|T|}$ of the graph $\mathcal{G}$, one per sink. The modified EK algorithm is run per $\mathcal{G}_t$, except that conflicting edges are deleted from *all* copies $\{\mathcal{G}_t\}_{t=1}^{|T|}$. The set of edge-disjoint augmenting paths for each sink t consist of edges with unit flow values in $\mathcal{G}_t$. The *overall* flow on $\mathcal{G}$ can be obtained by assigning unit flows to those edges in $\mathcal{G}$ which have unit flows on one of the graph copies $\mathcal{G}_t$.

Algorithm 1: Greedy Augmenting Path (GAP) Algorithm

1 **Initialization**: Create copies $\{\mathcal{G}_t\}_{t=1}^{|T|}$ of the time-unwrapped graph $\mathcal{G}$. Initialize flow values on all edges for each graph $\mathcal{G}_t$ to zero. Set the number of edge-disjoint augmenting paths $\mu = 0$.
2 **repeat**
3 **for** $t = 1, 2, \ldots, |T|$ **do**
4 Find the shortest augmenting $s^\nu - t^\nu$ path $\mathcal{P}^{(t)}$ on the graph $\mathcal{G}_t$.
5 Remove edges conflicting with edges in $\mathcal{P}^{(t)}$ from all graphs $\mathcal{G}_1, \ldots, \mathcal{G}_{|T|}$.
6 **end**
7 Increment a unit valid flow and reverse the edges along the augmenting paths $\mathcal{P}^{(t)}$ for each graph $\mathcal{G}_t$. Increment μ by one.
8 **until** *an* $s^\nu - t^\nu$ *path can be found*

The list of conflicting edges to be deleted depends on the interference model used. Let, $I_v(O_v)$ denote the set of incoming (outgoing) edges to the node $v \in \mathcal{V}$. For the PI model, the set of edges deleted at each inner iteration of Algorithm 1 is as follows.

P1) For every receiver-subnode $v^r(j) \in \mathcal{P}^{(t)}$,
  a) delete edge $(v^c(j), v^t(j))$; and
  b) delete edge $e \in I_{v^r(j)}$ if $e \notin \mathcal{P}^{(t)}$.

P2) For every transmitter-subnode $v^t(k) \in \mathcal{P}^{(t)}$, delete all edges $e \in I_{v^r(k)}$.

Edges are deleted in P1a) and P2) to prevent violation of the half-duplex constraint SC1); while those deleted in P1b) prevent violation of SC2a).

The list of edges to be deleted in the SI constraint SC2b) is slightly more extensive.

S1) For every receiver-subnode $v^r(j) \in \mathcal{P}^{(t)}$,
  a) delete edge $(v^c(j), v^t(j))$;
  b) delete edge $e \in O_u$, where $(u, v^r(j)) \in \mathcal{E}$ and $u \notin \mathcal{P}^{(t)}$; and
  c) delete edge $e \in O_{u^t(j)}$ where $(v, u) \in E$.

S2) For every transmitter-subnode $v^t(k) \in \mathcal{P}^{(t)}$,
  a) delete all edges $e \in I_{v^r(k)}$;
  b) delete all edges $e \in O_w$, where $w \notin \mathcal{P}^{(t)}$ such that $(w, u) \in \mathcal{E}$ and $(v^t(k), u) \in \mathcal{E}$ for some node u and $e \neq (v^t(k), u)$; and
  c) delete edge $e \in I_{u^r(k)}$, where $(u, v) \in E$.

As with the PI model, S1a) and S2a) take care of the half-duplex constraints. However, unlike the PI model, correct reception at a node v under SI is only ensured if all its neighboring nodes are

silent. The edges corresponding to these cases are listed in S1b), S1c), S2b), and S2c). Note that Algorithm 1 can be extended to include more general interference models by appropriately modifying these steps.

It is worth stressing that only the original graph $\mathcal{G}$, and not the residual graph copies, are used while determining the edges to be deleted. Thus, Algorithm 1 outputs edge-disjoint augmenting paths for each sink as argued. Likewise, Algorithm 1 does not eliminate the possibility of choosing augmenting paths that delete a large number of edges. The use of shortest augmenting paths however reduces this possibility. The next subsection provides further improvements by appropriately modifying the shortest-path finder employed by Algorithm 1.

A. GAP Enhancements

The first part of this subsection describes a *pipelined* approach to multicast that reduces the overhead caused due to AS1) [cf. Remark 1]. Pipelining alters AS1) by allowing the source to multicast more than one block of packets per D time slots. The second part describes an earliest-shortest path (ESP) algorithm, for use in Algorithm 1 which improves throughput by reducing the number of deleted edges. In addition, the ESP algorithm enables development of worst-case bounds for the performance of Algorithm 1.

1) Pipelined Multicast: As observed in Remark 1, AS1 results in an overhead by allowing only one block of packets per D time slots. This yields an overall throughput of $\frac{\mu}{D}$. The throughput can be improved if consecutive blocks are allowed to overlap while ensuring that they do not interfere with each other. This is effected through pipelining, which allows the source to begin transmitting the next block of packets as soon as all its neighbors have finished transmitting the current one.

The idea can be formalized using the notation from Section IV. Let d_t denote the length of the shortest path between the source s and a sink $t \in T$ in the graph G. Without loss of generality, let t_1 be the sink that is nearest to s, and let $d_1 := d_{t_1} = \min_t d_t$. Since the source begins transmitting at the first time slot, the sinks start receiving on the d_1st time slot at the earliest. Similarly, the source stops sending at the $(D-d_1+1)$st time slot since all packets must arrive by the Dth time slot. Depending on the interference model used, it is now possible to calculate the exact time slot on which the source can start sending the next block of packets without mixing it with the current block.

For PI networks, one-hop neighbors of the source are no longer transmitting at the $(D-d_1+3)$rd time slot. Thus, if the source transmits the next block of packets at the $(D-d_1+3)$rd time slot, its one-hop neighbors can receive them without interference. This is equivalent to saying that the schedules G_1 and G_{D-d_1+3} are conflict free. Similar deductions can be made about G_2 and G_{D-d_1+4}, and so on. Define the union operation for two conflict-free schedules G_{ℓ_1} and G_{ℓ_2} as

$$G_{\ell_1} \cup G_{\ell_2} = (V, E_{\ell_1} \cup E_{\ell_2}). \tag{1}$$

The overall pipelined network schedule can thus be expressed as the sequence of graphs $G_1, G_2, \ldots, G_{D-d_1+3} \cup G_1, G_{D-d_1+4} \cup G_2, \ldots$. Further, the asymptotic throughput, given that Algorithm 1 returns μ edge-disjoint paths, becomes $\frac{\mu}{(D-d_1+2)}$.

The argument for SI networks is similar, except that the one-hop neighbors of the source can only receive when *their* next-hop neighbors stop transmitting. This happens at the $(D-d_1+4)$th time slot, which yields an effective throughput of $\frac{\mu}{(D-d_1+3)}$.

In the analysis so far, it is assumed that in the worst case, the source may transmit the last packet to the sink t_1 at the $(D-d_1+1)$st time slot. However, all other sinks are farther than d_1 hops and the number of packets reaching every sink is the same. Therefore, a more efficient choice of augmenting paths may make the source send its last packets at the $(D-d_2+1)$st time slot where $d_2 = \max_t d_t$. This observation can be used to derive an upper bound on the achievable throughput. Specifically, if the source were to transmit one packet per-time slot, it is possible to transmit at most $D-d_2+1$ packets. If the augmenting paths are chosen carefully, transmission of the next block of packets may start immediately at time slot $D-d_2+2$, resulting in the maximum achievable throughput of 1. This is also the maximum achievable throughput for any JS-NC scheme, including those which do not consider delay constraints, since the source can only transmit at most one packet per-time slot.

2) The Earliest-Shortest Path (ESP) Algorithm: Recall that unlike the EK algorithm, it is possible to choose augmenting paths that may cause deletion of a large number of edges, thus yielding a small final value of μ. The choice of shortest augmenting paths is therefore a justifiable heuristic since shorter paths are expected to conflict with fewer edges. Another factor influencing the throughput returned by Algorithm 1 is the number of time slots for which each packet stays in the network. Intuitively, any packet that is transmitted within the first few time slots should be received by the sinks as soon as possible; or else, it may (unnecessarily) cause congestion to packets transmitted later. One way to ensure this is to always choose the shortest path $\mathcal{P}^{(t)}$ whose ending time slot (i.e., the time slot ℓ for which $t^r(\ell) \in \mathcal{P}^{(t)}$) is the least among all shortest paths.

This strategy can be implemented by using Dijkstra's algorithm to find the shortest path, but with a simple modification. Recall that Dijkstra's algorithm visits nodes starting at s^ν, and maintains an upper bound on the minimum distance from s^ν to each node. This upper bound is updated if a shorter distance is found, and the algorithm terminates when the entire graph has been visited. However, when all edges are of unit-length (as in the present case), if the nodes are visited in a breadth-first manner (i.e., the algorithm first visits all one-hop neighbors, then two-hop neighbors, and so on), the algorithm can terminate as soon as the destination is encountered.

The modified algorithm does not terminate on reaching the destination t^ν for the first time. Instead, a new variable $S_{\max}$ is initialized to store the time slot of the last visited receiver-subnode of t^ν. The next time t^ν is visited, the value of $S_{\max}$ is updated to the minimum of $S_{\max}$, and the time slot of the last-visited receiver-subnode. The algorithm terminates when all nodes as far from the source as t^ν have been visited. The earliest-shortest path can be recovered by backtracking along the receiver-subnode with time slot equal to the final value of

$S_{\max}$. In other words, the ESP algorithm is similar to Dijkstra's, except that the time slot of the last visited receiver-subnode is used to break ties while choosing the shortest path at the sink node. The full ESP scheme is listed as Algorithm 2.

Algorithm 2: Earliest-Shortest Path (ESP) Algorithm

Initialize $Q \leftarrow \{s^\nu\}$
Initialize variables $d_{\max} \leftarrow 0$, $d_{s^\nu} \leftarrow 0$, and $d_v \leftarrow \infty$ for all $v \in \mathcal{V} \setminus \{s^\nu\}$
Initialize $S_{\max} \leftarrow \infty$, and $S_v \leftarrow$ time slot associated with node v, for all $v \in \mathcal{V} \setminus \{s^\nu, \mathcal{T}\}$
repeat
  $u \leftarrow \arg\min_{w \in Q} d_w$
  $Q \leftarrow Q \setminus \{u\}$
  $d_{\max} \leftarrow d_u$
  for each *node v in set* $\{v|(u,v) \in \mathcal{E}\}$ **do**
    if $v = t^\nu$ **then**
      $S_{\max} \leftarrow \min(S_u, S_{\max})$
    end
    $d_v \leftarrow \min(d_v, d_u + 1)$
    $Q \leftarrow \{Q \cup \{v\}\}$
  end
until $d_{\max} = d_{t^\nu}$
Backtrack path starting from $t^r(S_{\max})$ to s^ν.

Note that Algorithm 2 visits nodes in a breadth-first manner starting at s^ν. This is accomplished by maintaining a set Q of all nodes which have themselves been visited but whose neighbors have not been visited. At each iteration, the neighbors of a node closest to the source is visited and the distance metrics are updated. Interestingly, Algorithm 2 can be used to claim certain approximation guarantees for the PI model. This is established in the ensuing subsection.

B. Performance Bounds

Performance bounds are developed in this section for Algorithm 1 applied to PI networks. The following theorem gives a bound on the achievable throughput.

Theorem 1: *The throughput ρ obtained through Algorithm 1 using the ESP and pipelining enhancements can be bounded as follows:*

$$\frac{\left\lfloor \frac{D-d_2+2}{2} \right\rfloor}{D-d_1+2} \leq \rho \leq 1. \tag{2}$$

As a corollary, it can be seen that as $D \rightarrow \infty$, the bound reduces to $\frac{1}{2} \leq \rho \leq 1$. Next, the proof of Theorem 1 is provided.

Proof: The upper bound has already been derived in Section V-A-1). The proof of the lower bound relies on the special structure of the time-unwrapped graph $\mathcal{G}$. In particular, notice that the shortest $s^\nu - t^\nu$ path in $\mathcal{G}$ corresponds to the set of wireless nodes that lie on the shortest $s - t$ path in G. Thus, for each sink t, there exist several shortest paths in $\mathcal{G}$ each of which has the following form:

$$\mathcal{P}^{\boldsymbol{\ell}}(\ell) = \big(s^\nu, s^t(\ell_1), v_1^r(\ell_2), v_1^t(\ell_2), \ldots, v_{d_t-1}^r(\ell_{d_t}-1), v_{d_t-1}^c(\ell_{d_t}), v_{d_t-1}^t(\ell_{d_t}), t^r(\ell_{d_t}), t^\nu\big) \tag{3}$$

where $\boldsymbol{\ell} := (\ell_1, \ell_2, \ldots, \ell_{d_t})$ and $1 \leq \ell_1 \leq \ell_2 \leq \ldots \leq \ell_{d_t} \leq D$. The length of the shortest $s^\nu - t^\nu$ path is therefore $3d_t + 1$. This is also the minimum length of the shortest augmenting path on any residual graphs that arise in Algorithm 1. In other words, the length of the shortest augmenting path always increases as iterations of Algorithm 1 go on. Such a behavior of increasing path-lengths is well known for the EK algorithm. It also holds here since any augmenting path found in Algorithm 1 is a feasible EK augmenting path.

For each sink $t \in T$, define the quickest-shortest (QS) path starting at time slot ℓ as $Q^{(t)}(\ell) := \mathcal{P}^{(t)}(\boldsymbol{\ell})$, where $\boldsymbol{\ell} = (\ell, \ell+1, \ell+2, \ldots, \ell+d_t-1)$. Note that given any graph $\mathcal{G}$, such a path exists for every sink $t \in T$ and every time slot $1 \leq \ell \leq D - d_t + 1$. Further, starting at time slot ℓ, $Q^{(t)}(\ell)$ is a shortest path that reaches the sink $t^\nu \in \mathcal{T}$ at the earliest possible time slot $\ell + d_t - 1$. Thus, for a given sink $t \in T$ Algorithm 2 will return the QS path $Q^{(t)}(\ell)$ for some ℓ, as long as such a path exists in the residual graph. Note that the QS paths $\{Q^{(t)}(\ell)\}_{t \in T}$ do not conflict with each other, and thus form a shortest path tree (SPT).

Next, define a partial order on all shortest augmenting paths returned by Algorithm 2 with starting time slot ℓ_1 and ending time slot ℓ_{d_t}. Specifically, given two augmenting paths $\mathcal{P}^{(t_1)}(\boldsymbol{i})$ and $\mathcal{P}^{(t_2)}(\boldsymbol{j})$, ending at two, possibly different sinks t_1 and t_2, define $\mathcal{P}^{(t_1)}(\boldsymbol{i}) \leq \mathcal{P}^{(t_2)}(\boldsymbol{j})$ if and only if $i_1 \leq j_1$ *and* $i_{d_{t_1}} \leq j_{d_{t_2}}$. The partial ordering can now be used to understand Algorithm 2 better. For instance, if at some iteration in Algorithm 1, it is known that a path $\mathcal{P}_1^{(t)}$ exists in the residual graph, then Algorithm 2 will always return a path $\mathcal{P}_2^{(t)} \leq \mathcal{P}_1^{(t)}$ of length less than or equal to the length of $\mathcal{P}_1^{(t)}$.

The following lemma states another useful aspect of the QS paths.

Lemma 1: *If t_a and t_b be two (possibly different) sinks, and ℓ denotes any time slot such that the paths $Q^{(t_a)}(\ell)$ and $Q^{(t_b)}(\ell+2)$ exist on the graph $\mathcal{G}$, then the path $Q^{(t_b)}(\ell+2)$ does not conflict with any path $\mathcal{P} \leq Q^{(t_a)}(\ell)$.*

The proof of Lemma 1 is provided in Appendix A. The result is interesting in the sense that the entire SPT formed by QS paths starting at a given time slot ℓ does not conflict with any path in the SPT starting at time slot $\ell + 2$. This observation can now be used to prove the main result of Theorem 1 as follows.

- L1) In the first iteration, the paths $Q^{(t)}(1)$ for each sink $t \in T$ exist and are therefore returned by Algorithm 2. At the end of the first iteration, all edges that lie on any of the paths $Q^{(t)}(1)$ are reversed and assigned unit flow values.
- L2) At the second iteration, any ESP starts at or after the second time slot. While the paths $Q^{(t)}(2)$ may not necessarily exist, Lemma 1 ensures that the QS paths $Q^{(t)}(3)$ still exist; that is, they are not deleted from the residual graph in the first iteration. As observed earlier, an ESP returned at the second iteration is such that $\mathcal{P}^{(t)} \leq Q^{(t)}(3)$.
- L3) Generically, the ith iteration returns a path $\mathcal{P}^{(t)} \leq Q^{(t)}(2i+1)$. Since the farthest sink allows the source to transmit up to the $(D - d_2 + 1)$th time slot, there are at least $\left\lfloor \frac{D-d_2+2}{2} \right\rfloor$ iterations, and as many augmenting paths.

L4) As observed earlier, transmission of the next block can begin at time slot $(D-d_1+3)$. This yields the asymptotic throughput of $\frac{\left\lfloor \frac{(D-d_2+2)}{2} \right\rfloor}{D-d_1+2}$.

□

Note that for the SI model, it is not possible to provide similar guarantees as the QS paths for different sinks $t \in T$ starting at the same time slot $Q^{(t)}(\ell)$ may conflict with each other. Thus, the existence of the SPT itself is not guaranteed. However, the proof of Theorem 1 provides some justification for the ESP heuristic even when the algorithm is applied to generic interference models.

C. Distributed Implementation

Algorithm 1 readily lends itself to a distributed implementation. Assume that each wireless node is aware of its two-hop neighbors, the source and sink nodes, and the graph parameters D and d_t for each $t \in T$. The following observations may then be used to distribute the algorithm.

D1) Construction of the time-unwrapped graph $\mathcal{G}$ only involves creation of several subnodes per node, which can be done locally, without requiring any communication among the nodes.

D2) The source must calculate the ESP for every iteration and every sink. Distributed and asynchronous versions of Dijkstra's algorithm are available [32, Ch. 5], and can be readily adapted for Algorithm 2 here. A speed improvement can be obtained by always visiting the nodes with an earlier time slot first.

D3) Finally, the source sends a packet along the shortest augmenting path, found in D2), informing the nodes of its choice. The participating nodes may then obtain the residual graph, update flow values along their edges, and delete conflicting edges by informing their neighbors.

Before ending this section, a few remarks are in order.

Remark 5: During the operation, each node in Algorithm 1 transmits and receives on predetermined time slots with a fixed schedule. This allows most nodes to sleep for most of the time slots, except when operating or performing maintenance tasks. This aspect of Algorithm 1 makes it attractive for sensor and *ad hoc* networks.

Remark 6: Most deterministic NC designs, such as those in [29], result in relatively small finite field sizes, typically $O(|T|)$. Randomized schemes such as the one in [30] only require a field size q that is a prime power greater than $|T|$. This is in contrast with most random NC schemes that assume asymptotically large field sizes (usually 2^8 or 2^{16}). Smaller field sizes translate to lower overhead since the coding coefficients are usually carried in the packet headers [11].

Remark 7: The distributed version of the algorithm works in a feed-forward way. Thus, neither link-by-link nor end-to-end acknowledgments are required. Such an ACK-free operation makes sense in networks with hard deadlines since nodes do not have time for retransmissions anyway. This is appealing for video streaming applications, where feed-forward operation is commonly used; see, e.g. [25].

VI. Linear Programming Bounds

This section examines the maximization of μ from an IP perspective. Section VI-A describes an IP formulation for PI networks. While it may be impossible to efficiently solve the resultant IP for large networks, the formulation provides ways of obtaining upper bounds. For example, a linear programming (LP) bound is obtained in Section VI-A by relaxing the integrality constraints in the IP. Section VI-B further improves this bound by adding a class of *valid inequalities*.

A. Integer Programming Formulation

Following the notation of Section IV, the problem of finding the maximum number of edge-disjoint paths from the source s^ν to each of the sinks $t^\nu \in \mathcal{T}$, can be expressed as the following integer program:

$$\mu^* = \max \mu \tag{4a}$$

$$s.t. \quad \sum_{e \in I_v} x_e^{(t)} = \sum_{e \in o_v} x_e^{(t)} \quad \forall\, t \in T, v \in \mathcal{G} \setminus (s^\nu, t^\nu) \tag{4b}$$

$$\sum_{e \in O_s} x_e^{(t)} = \mu \qquad \forall\, t \in T \tag{4c}$$

$$\sum_{e \in I_t} x_e^{(t)} = \mu \qquad \forall\, t \in T \tag{4d}$$

$$z_e \geq x_e^{(t)} \qquad \forall\, t \in T, e \in \mathcal{E} \tag{4e}$$

$$x_e^{(t)}, z_e \in \{0,1\} \tag{4f}$$

where variables $x_e^{(t)}$ and z_e represent the virtual and real flows, respectively, on the edge $e \in \mathcal{E}$ [9]. The flow variables are related to the flows defined in Section V. The virtual flow $x_e^{(t)}$ corresponds to the flow values assigned to edges on $\mathcal{G}_t$, while the real flow z_e corresponds to the overall flow on $\mathcal{G}$.

In the wireless setting, the scheduling constraints SC1)–SC2a) must also be added. For a node v and time slot k, these constraints can be represented by

$$\sum_{e \in I_{v^r(k)}} z_e + z_{(v^c(k), v^t(k))} \leq 1 \quad \forall v \in \mathcal{V}, 1 \leq k \leq D \tag{4g}$$

where the first summand in (4g) represents the total flow on edges incoming to the receiver-subnode $v^r(k)$, and the second term is the flow leaving the combiner/transmitter-subnode at the same time slot. The inequality ensures that in a single time slot k, at most a single packet is either received (from a single node) or transmitted (broadcast to possibly multiple nodes).

The LP bound for the problem (4a)–(4g) can be obtained by relaxing (4f) with

$$x_e^{(t)}, z_e \in [0,1]. \tag{4h}$$

This bound can be further improved by adding "tightening" inequalities that are valid only for the original IP constraints. In other words, these valid inequalities may "cut-off" regions of the polyhedron defined by the linear inequality constraints (4b)–(4e) and (4g), (4h). Valid inequalities can also be used to exactly solve the IP using methods such as branch-and-cut [33, Ch. 8], although the worst-case complexity of these IP-solvers is still not polynomial. The next subsection focuses on developing a set of such valid inequalities.

B. A Class of Valid Inequalities

Before describing the valid inequalities, some simplifications and related notation is introduced. First note that it is straightforward to eliminate the variable μ from the set of (4b)–(4d).

Next, let $\mathbf{w}$ be the $n \times 1$ super-vector that contains all remaining optimization variables $\{x_e^{(t)}, z_e\}$. After eliminating μ, the constraints (4b)–(4e) and (4g) can be generically denoted by the set of inequalities $\mathbf{Aw} \leq \mathbf{b}$, where each equality constraint is simply expressed as two opposing inequalities. The set of all feasible IP solutions is then given by $\{\mathbf{w} \in \{0,1\}^n | \mathbf{Aw} \leq \mathbf{b}\}$, while the corresponding LP relaxation lies in the polyhedron represented as the set $\{\mathbf{w} \in [0,1]^n | \mathbf{Aw} \leq \mathbf{b}\}$. A set of inequalities $\mathbf{Cw} \leq \mathbf{d}$ is said to be valid if

$$\{\mathbf{w} \in \{0,1\}^n | \mathbf{Aw} \leq \mathbf{b}\} = \{\mathbf{w} \in \{0,1\}^n | \mathbf{Aw} \leq \mathbf{b}, \mathbf{Cw} \leq \mathbf{d}\} \tag{5}$$

while

$$\{\mathbf{w} \in [0,1]^n | \mathbf{Aw} \leq \mathbf{b}\} \supseteq \{\mathbf{w} \in [0,1]^n | \mathbf{Aw} \leq \mathbf{b}, \mathbf{Cw} \leq \mathbf{d}\}. \tag{6}$$

It is well known that the optimum solution of an LP always lies on an extreme point of the polyhedron defined by its linear inequalities [33, Ch. 2]. For an IP however, its LP-relaxation polyhedron may not necessarily have integral extreme points. The optimum solution of the LP may, therefore, be fractional, and its optimum value may lie far from the IP optimum. Valid inequalities can be used in such cases to cut-off some or all of the fractional extreme points of the LP relaxation polyhedron.

In principle, a finite number of "necessary" valid inequalities is sufficient to ensure that all extreme points of $\{\mathbf{w} \in [0,1]^n | \mathbf{Aw} \leq \mathbf{b}, \mathbf{Cw} \leq \mathbf{d}\}$ are integral. A well-known method of generating valid inequalities is the Chvatal-Gomory (CG) procedure, which can generate all necessary valid inequalities in a finite number of steps [33, Ch. 8]. Given a system of m linear inequalities, $\mathbf{Aw} \leq \mathbf{b}$, and a vector $\mathbf{g} \in [0,1)^m$, the CG procedure generates the valid inequality (also called a CG-cut) denoted as

$$\lfloor \mathbf{g}^T \mathbf{A} \rfloor \mathbf{w} \leq \lfloor \mathbf{g}^T \mathbf{b} \rfloor \tag{7}$$

where $\lfloor \boldsymbol{\alpha} \rfloor$ stands for the elementwise floor operation of the vector $\boldsymbol{\alpha}$.

The caveat however is that the CG-procedure generates an exponentially large set of inequalities, which cannot be handled efficiently by any LP solver. The remainder of this section describes a method to generate a smaller class of valid inequalities that can be efficiently *separated*, and thus accommodated by LP solvers.

Fig. 3 depicts a simple example network and its time-expanded graph for $D = 3$, with redundant nodes and edges removed. The solution obtained for this network using the LP relaxation is, $x_e = z_e = 1$ for $e = (s^\nu, s^t(1)), (s^t(1), v^r(1)), (v^t(3), t^r(3)), (t^r(3), t^\nu)$, and $x_e = z_e = 0.5$ for all other $e \in \mathcal{E}$. This gives a total flow of 1.5 that also satisfies the constraints (4g). The integral solution, on the other hand, achieves only one unit of end-to-end flow, i.e., a single packet is transmitted from s to v in the first time slot, and from v to t in the second time slot.

An observation that follows from this example is that in three time slots, only one packet goes "through" v. Interestingly, this also holds for larger values of D and for any three, possibly non-contiguous, time slots. For instance there can be at most three packets, each either transmitted or received by a node in three time slots k_1, k_2, and k_3. However, there can be at most one packet which is both transmitted *and* received in these three time slots. In contrast, (4g) allows 1.5 packets to be transmitted and received.

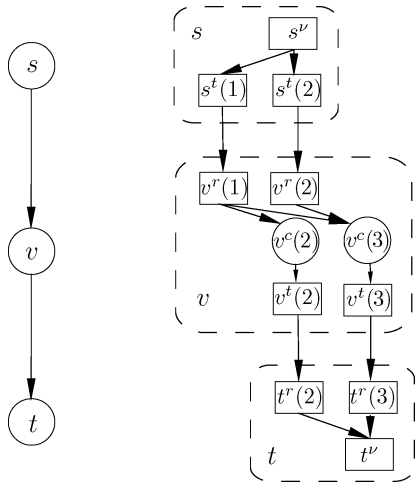

Fig. 3. An example of wireless network and its time-expanded version.

In order to enforce this condition, note that the flow passing through a node v in time slots k_1, k_2, and k_3, is given by the total flow through the edges in $\mathcal{C} := \{(n_1, n_2) | n_1 \in \{v^r(k_1), v^r(k_2)\}, n_2 \in \{v^c(k_2), v^c(k_3)\}\}$. Thus, an extra inequality can be introduced, limiting the total flow on these edges to one. Note that for the case of multiple sinks, the virtual flow corresponding to each sink requires a separate inequality. The idea can be generalized to any odd number of time slots as asserted by the following theorem.

Theorem 2: *For the time slots $1 \leq k_1, k_2, \ldots, k_\ell \leq D$, and a node v, define the set of edges*

$$\mathcal{C} := \{(n_1, n_2) | n_1 \in \{v^r(k_1), v^r(k_2), \ldots, v^r(k_{\ell-1})\} \quad n_2 \in \{v^c(k_2), v^c(k_3), \ldots, v^c(k_\ell)\}\}. \tag{8}$$

Then, the following is a valid inequality for all $t \in T$

$$\sum_{e \in \mathcal{C}} x_e^{(t)} \leq \left\lfloor \frac{\ell}{2} \right\rfloor. \tag{9}$$

Proof: The cuts can be generated by applying $\{0, \frac{1}{2}\}$-CG cuts to some of the constraints in (4b)–(4e) and (4g). Note that for any slot k, the following holds:

$$\begin{aligned} &\sum_{e \in I_{v^r(k)}} z_e + z_{(v^c(k), v^t(k))} \leq 1 \\ \overset{\text{[cf. (4e)]}}{\Rightarrow} &\sum_{e \in I_{v^r(k)}} x_e^{(t)} + x_{(v^c(k), v^t(k))}^{(t)} \leq 1 \\ \overset{\text{[cf. (4b)]}}{\Rightarrow} &\sum_{e \in O_{v^r(k)}} x_e^{(t)} + \sum_{e \in I_{v^c(k)}} x_e^{(t)} \leq 1. \end{aligned} \tag{10}$$

Adding up the last set of equations for slots $k = k_1, k_2, \ldots, k_\ell$, we obtain

$$\sum_{k \in \{k_1, \ldots, k_\ell\}} \sum_{e \in O_{v^r(k)}} x_e^{(t)} + \sum_{k \in \{k_1, \ldots, k_\ell\}} \sum_{e \in I_{v^c(k)}} x_e^{(t)} \leq \ell. \tag{11}$$

Note that in (11), the terms $x_e^{(t)}$ for all $e \in \mathcal{C}$ occur twice, while all other terms occur only once. Thus, dividing (11) by 2 and rounding towards zero, we arrive at (9). □

The generated set of inequalities (9) is much smaller than the full set of all possible valid inequalities, and is therefore not necessarily optimal. Further, even this set contains exponentially many constraints. Interestingly however, the set admits an efficient separation oracle which identifies a possibly violated inequality given any feasible solution of the relaxed LP problem. The solution to the entire LP can also be found efficiently using the ellipsoid method [34, Chap. 5], whose calls to the separation oracle can be bounded polynomially. For the set of inequalities generated by (9), the following result holds.

Lemma 2: *The worst-case complexity of the separation oracle is* $O(nD|T|)$.

Proof: Given a candidate solution $(x_e^{(t)}, z_e)$, the problem of verifying its feasibility can be stated as follows:

For every node v and sink t,

F1) Find the set of time slots $k_1, \ldots, k_\ell$ such that a constraint in (9) is violated; or

F2) output that there is no such set.

It will be argued next that this problem is equivalent to finding a separation oracle for the *matching* problem on a derived graph. Given a graph $G_v = (U_v, E_v)$, associate variables y_e^v with each edge $e \in E_v$. Let C_u denote the set of edges connected to a node $u \in U_v$. A matching is a set of edges, such that no two edges of the set connect to the same node. Equivalently, a matching is an assignment of binary values to variables y_e^v such that

$$\sum_{e \in C_u} y_e^v \leq 1, \qquad y_e^v \in \{0,1\}. \tag{12a}$$

Interestingly, it is possible to replace the integrality constraints in (12a) with simple nonnegativity constraints, by adding the following set of valid inequalities

$$\sum_{\{e=(u_1,u_2)|u_1,u_2 \in \mathcal{S}\}} y_e^v \leq \left\lfloor \frac{|\mathcal{S}|}{2} \right\rfloor \ \forall \text{ sets of nodes } \mathcal{S}. \tag{12b}$$

Although the number of valid inequalities in (12b) is also exponential, it is possible to design a separation oracle that returns the violated inequality in $O(|U_v|)$ [34, Chap. 25].

In the present case, for a node v, construct graph G_v with nodes $1, 2, \ldots, D$, and connect pairs of nodes (i, j) for all $i > j$. The edge (i, j) in G_v represents the edge $(v^r(i), v^c(j))$ in the original graph $\mathcal{G}$. Similarly, set the variables y_{ij}^v equal to the corresponding edge variables $x_{(v^r(i),v^c(j))}^{(t)}$. A related set of constraints for y_{ij}^v can be derived based on (4g) and (9) as follows.

M1) All flow variables are positive, thus implying $y_e^v \geq 0$ for all $e = (i, j)$.

M2) The set of edges connecting to a node $k \in \{1, \ldots, D\}$ in G_v correspond to all the edges $e \in O_{v^r(k)} \cup I_{v^c(k)}$. Thus, (10) implies that

$$\sum_{e \in C_k} y_e^v \leq 1. \tag{13a}$$

M3) Since any set of time slots $\{k_1, \ldots, k_\ell\}$ corresponds to an equivalent set of nodes in G_v, (9) translates to

$$\sum_{\{e=(k_i,k_j)|1 \leq i,j \leq \ell\}} y_e^v \leq \left\lfloor \frac{\ell}{2} \right\rfloor \quad \forall\, 1 \leq \ell \leq D. \tag{13b}$$

It can be seen that the constraints (13a), (13b) resemble the matching constraints (12a), (12b). Thus, given a candidate solution $x_e^{(t)}$, an assignment to variables y_e^v can be calculated for each node v. Invoking the matching separation oracle then results in a possibly violated inequality in terms of y_e^v, which can finally be translated to a corresponding inequality in terms of $x_e^{(t)}$. Since the separation oracle runs in time $O(D)$ and must be invoked for every node and every virtual flow, the total time complexity is $O(nD|T|)$. □

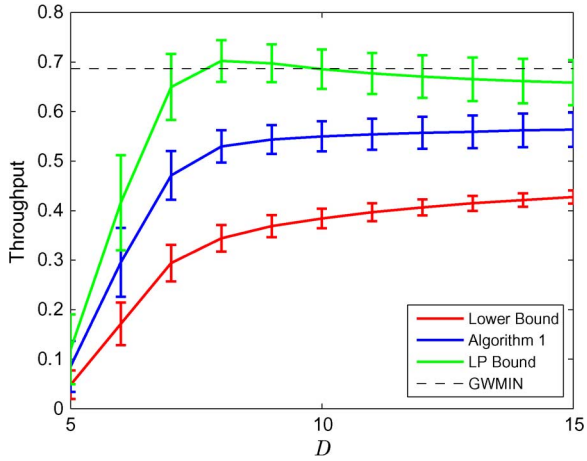

Fig. 4. Performance and bounds on a PI network.

VII. Numerical Comparisons

This section presents simulations on the performance of Algorithm 1. For comparison, the throughput obtained using a delay-agnostic, conflict-graph method from [21] along with the bounds derived in Sections V-B and VI, are also plotted.

Random networks are generated using the MAX-DPA algorithm outlined in [35]. The algorithm generates graphs by placing nodes one-by-one, while respecting certain maximum-degree and proximity constraints so as to simulate a realistic *ad hoc* network. The algorithm parameters are chosen to be $d = 5$, $d_{\max} = 8$, and $d_0 = 0.2$, which denote respectively the average and maximum node degrees, and the minimum distance between neighbors. The nodes are placed in a square area chosen such that the average node density is one. Next, the leftmost node is chosen to be the source and all edges are chosen to be directed away from the source. Finally, all nodes without any outgoing edges are chosen to be sinks.

Fig. 4 shows the performance of Algorithm 1 for small PI networks. The throughput is averaged over 2000 different networks with 20 nodes each, and is plotted for a range of values of the deadline D. The length of the bars equals half the standard deviation over the network instances. Pipelining in Algorithm 1 is implemented such that the source does not necessarily wait till the $(D - d_1 + 3)$rd time slot, but may begin transmission earlier if possible. For comparison, the lower bound stated in Theorem 1 and the LP upper bounds are also plotted. In the absence of any deadlines, it is possible to evaluate the maximum achievable throughput using one of the approximation algorithms outlined in [21]. The dashed line in the figure shows this value, calculated using the random approach of [21] with 500 random maximal-independent sets, assuming no erasures on links.

As expected, Algorithm 1 exhibits graceful degradation in performance as the deadline is reduced. Interestingly, the trend is also visible in the curves showing upper and lower bounds on

TABLE I
PERFORMANCE OF ALGORITHM 1 ON LARGE NETWORKS

D	15	20	25	30	35	Random approach [21]
Network 1	0.33	0.36	0.37	0.38	0.38	0.50
Network 2	0.20	0.20	0.24	0.25	0.27	0.39
Network 3	0.25	0.21	0.29	0.30	0.28	0.39

the throughput. Further, it can be seen that the bounds become tighter as the value of D increases, reaffirming their usefulness. Finally, note that the variation apparent from the standard deviation bars is largely because of the variation among random networks. Thus, the overlap between the bars for lower and upper bound does *not* mean that the bounds are incorrect for some network instances.

Next, the performance is analyzed on a large network with 100 nodes and SI constraints. Three random networks are generated, and Algorithm 1 is run for different values of D. While the lower and upper bounds do not apply to this case, the random approach from [21] is again used here as a benchmark. Note that for large networks, the number of hyperarcs, and consequently the size of the resulting LP used in [21] becomes prohibitive. Towards this end, the interference model used in [21] is simplified by considering only broadcast transmissions, i.e., each node either broadcasts its packets to all its receivers or stays silent. This translates to the simple rule, used in several MAC protocols: a node transmits only when its two-hop neighborhood is silent. Further, only 200 maximal-independent sets are generated. Table I lists the throughput achieved for all three realizations. As with the PI model, the throughput-delay trade off is again apparent here. In this case however, the difference between the deadline-free case and the GAP throughput with large D is not as pronounced. Further, the quality of approximation in Algorithm 1 also depends on the topology of the network. Thus for some networks, such as Network 3 in Table I, the throughput does not always decrease monotonically with D.

Finally, the performance of the resulting network protocol is studied for different erasure probabilities. Towards this end, a random network with 100 nodes is generated, and Algorithm 1 is run to obtain the network operation schedules. Next, the protocol is simulated using Monte-Carlo runs, assuming that the links fail independently with specified erasure probabilities. Fig. 5 depicts the average throughput, given by the average number of linear combinations received by the sinks, per time slot. It can be seen that the throughput performance degrades only gradually with erasures.

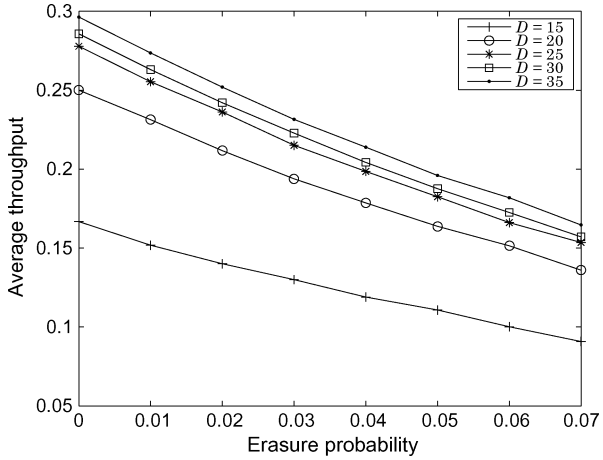

Fig. 5. Degradation of throughput with packet erasures for different values of D.

VIII. CONCLUSION

This paper considered network-coded multicast with deadline constraints. Since popular generation-based approaches do not handle delay constraints, a joint scheduling and network coding approach is introduced to maximize the average throughput while respecting the wireless constraints and packet-deadlines. The novel algorithm relies on a time-unwrapped graph expansion in order to construct linear-periodic time-varying network codes. The approach draws from the well-known augmenting-path algorithm, and is therefore both distributed and scalable. For networks with primary interference constraints, the algorithm was shown to have a constant-factor bounded worst-case performance. The setup was also analyzed from an integer programming perspective, and a set of valid inequalities was developed and used to obtain a linear programming based upper bound on the throughput.

APPENDIX A
PROOF OF LEMMA 1

First, using contradiction, we show that paths $Q^{(t_a)}(\ell)$ and $Q^{(t_b)}(\ell+2)$ do not conflict. If the two said paths indeed conflict, it would imply that there exist $e_a \in Q^{(t_a)}(\ell)$ and $e_b \in Q^{(t_b)}(\ell+2)$ such that one of the following holds:

C1) Edges e_a and e_b violate the half-duplex constraint. This means that there exists a node $v \in V$ and a time slot $\ell \leq k \leq \min(d_{t_a}, d_{t_b})$ such that either a) $e_b \in I_{v^r(k)}$ and $e_a \in O_{v^t(k)}$; or b) $e_a \in I_{v^r(k)}$ and $e_b \in O_{v^t(k)}$.

C2) There exists a node $v \in V$ and time slot k such that $e_a \in I_{v^r(k)}$ and $e_b \in I_{v^r(k)}$.

C3) The two edges are the same, i.e., $e_a = e_b$.

We begin by assuming that C1a) holds for some time slot k and node v. Since node $v^r(k+1)$ lies on the QS path $Q^{(t_a)}(\ell+2)$, it implies that a subnode in the wireless node v can be reached in $k-\ell-1$ time slots if the path along $Q^{(t_a)}(\ell+2)$ is taken. Note however that $v^r(k)$ also lies on $Q^{(t_b)}(\ell)$, which would imply that it must take at least $k-\ell$ time slots if a path along $Q^{(t_b)}(\ell)$ is taken. Therefore, the path $Q^{(t_a)}(\ell)$ reaches node v earlier than the path $Q^{(t_b)}(\ell)$ starting at the same slot. This is a contradiction since both paths were already assumed to be QS paths. The intuition is that starting at time slots ℓ and $\ell+2$, the time slots at which two QS paths reach a node differ by at least two.

The complementary case C1b) yields an even stronger contradiction as it implies that the QS path starting at a later time slot reaches a node at an earlier one. Similarly, the other cases C2) and C3) also follow from the aforementioned argument. Specifically, both C2) and C3) imply that two QS paths, starting at different time slots ℓ and $\ell+2$, reach a node at the same time slot k, which is not possible. It can be seen that the argument holds if the path $Q^{(t_a)}(\ell)$ is replaced by a shortest path $\mathcal{P} \leq Q^{(t_a)}(\ell)$ since that would again imply a stronger contradiction.

Note that it is not possible to provide similar guarantees for the SI model since, unlike the PI model, two QS paths starting at the same time slot may conflict with each other.

REFERENCES

[1] R. W. Yeung, S.-Y. R. Li, N. Cai, and Z. Zhang, "Network coding theory," *Found. Trends in Commun. Inf. Theory*, vol. 2, no. 4 and 5, pp. 241–381, 2005.

[2] S.-Y. Li, R. Yeung, and N. Cai, "Linear network coding," *IEEE Trans. Inf. Theory*, vol. 49, no. 2, pp. 371–381, 2003.

[3] P. Chou and Y. Wu, "Network coding for the Internet and wireless networks," *IEEE Signal Process. Mag.*, vol. 24, no. 5, pp. 77–85, Sep. 2007.

[4] C. Fragouli and E. Soljanin, "Network coding applications," *Found. Trends in Netw.*, vol. 2, no. 2, pp. 135–269, 2007.

[5] C. Fragouli, D. Katabi, A. Markopoulou, M. Médard, and H. Rahul, "Wireless network coding: Opportunities and challenges," in *Proc. MILCOM Conf.*, Orlando, FL, Oct. 2007.

[6] Y. Sagduyu and A. Ephremides, "On joint MAC and network coding in wireless *ad hoc* networks," *IEEE Trans. Inf. Theory*, vol. 53, no. 10, pp. 3697–3713, 2007.

[7] D. Traskov, M. Heindlmaier, M. Médard, R. Kötter, and D. Lun, "Scheduling for network coded multicast: A conflict graph formulation," in *Proc. GLOBECOM Workshops*, New Orleans, LA, Nov.–Dec. 2008.

[8] P. Chaporkar and A. Proutiere, "Adaptive network coding and scheduling for maximizing throughput in wireless networks," in *Proc. MobiCom Conf.*, New York, Sep. 2007, pp. 135–146.

[9] D. S. Lun, N. Ratnakar, M. Médard, R. Kötter, D. R. Karger, T. Ho, E. Ahmed, and F. Zhao, "Minimum-cost multicast over coded packet networks," *IEEE Trans. Inf. Theory*, vol. 52, no. 6, pp. 2608–2623, Jun. 2006.

[10] T. Ho and H. Viswanathan, "Dynamic algorithms for multicast with intra-session network coding," *IEEE Trans. Inf. Theory*, vol. 55, no. 2, pp. 797–815, Feb. 2009.

[11] P. A. Chou, Y. Wu, and K. Jain, "Practical network coding," in *Proc. 41st Ann. Allerton Conf. on Commun., Contr., Comput.*, Monticello, IL, Oct. 2003, pp. 40–49.

[12] E. Erez, M. Effros, and T. Ho, "Network codes with deadlines," in *Proc. 46th Ann. Allerton Conf. Commun., Contr., Comput.*, Monticello, IL, Sep. 2008, pp. 339–346.

[13] E. Drinea, C. Fragouli, and L. Keller, "Delay with network coding and feedback," in *Proc. Int. Symp. Inf. Theory*, Seoul, Korea, Jun. 2009, pp. 844–848.

[14] X. Li, C.-C. Wang, and X. Lin, "Throughput and delay analysis on uncoded and coded wireless broadcast with hard deadline constraints," in *Proc. INFOCOM Conf.*, San Diego, CA, 14–19, 2010, pp. 1–5.

[15] A. Eryilmaz, A. Ozdaglar, M. Médard, and E. Ahmed, "On the delay and throughput gains of coding in unreliable networks," *IEEE Trans. Inf. Theory*, vol. 54, no. 12, pp. 5511–5524, 2008.

[16] R. Costa, D. Munaretto, J. Widmer, and J. Barros, "Informed network coding for minimum decoding delay," in *Proc. Int. Conf. Mobile Ad hoc and Sensor Netw.*, Atlanta, GA, Sep. 2008, pp. 80–91.

[17] H. Seferoglu and A. Markopoulou, "Opportunistic network coding for video streaming over wireless," in *Proc. Packet Video Conf.*, Lausanne, Switzerland, Nov. 2007, pp. 191–200.

[18] J.-S. Park, M. Gerla, D. Lun, Y. Yi, and M. Médard, "Codecast: A network-coding-based *ad hoc* multicast protocol," *IEEE Trans. Wireless Commun.*, vol. 13, no. 5, pp. 76–81, Oct. 2006.

[19] L. Bui, R. Srikant, and A. Stolyar, "Novel architectures and algorithms for delay reduction in back-pressure scheduling and routing," in *Proc. INFOCOM Conf.*, Rio de Janeiro, Brazil, Apr. 2009, pp. 2936–2940.

[20] T. Cui, L. Chen, and T. Ho, "On distributed scheduling in wireless networks exploiting broadcast and network coding," *IEEE Trans. Commun.*, vol. 58, no. 4, pp. 1223–1234, Apr. 2010.

[21] M. Heindlmaier, D. Traskov, R. Kötter, and M. Médard, "Scheduling for network coded multicast: A distributed approach," in *Proc. GLOBECOM Workshops*, Honolulu, HI, Nov. 2009, pp. 1–6.

[22] E. Erez and M. Feder, "Convolutional network codes," in *Proc. Int. Symp. on Inf. Theory*, Jun. 2004, p. 146.

[23] Z. Zhang, "Linear network error correction codes in packet networks," *IEEE Trans. Inf. Theory*, vol. 54, no. 1, pp. 209–218, Jan. 2008.

[24] R. Kötter and F. Kschischang, "Coding for errors and erasures in random network coding," *IEEE Trans. Inf. Theory*, vol. 54, no. 8, pp. 3579–3591, Aug. 2008.

[25] A. Ramasubramonian and J. Woods, "Multiple description coding and practical network coding for video multicast," *IEEE Signal Process. Lett.*, vol. 17, no. 3, pp. 265–268, Mar. 2010.

[26] D. Silva and F. Kschischang, "Rank-metric codes for priority encoding transmission with network coding," in *Proc. Canad. Workshop on Inf. Theory*, Edmonton, AB, Jun. 2007, pp. 81–84.

[27] R. E. Burkard, K. Dlaska, and B. Klinz, "The quickest flow problem," *Math. Methods of Oper. Res.*, vol. 37, no. 1, pp. 31–58, Feb. 1993.

[28] R. Ahlswede, N. Cai, S.-Y. Li, and R. Yeung, "Network information flow," *IEEE Trans. Inf. Theory*, vol. 46, no. 4, pp. 1204–1216, Jul. 2000.

[29] S. Jaggi, P. Sanders, P. Chou, M. Effros, S. Egner, K. Jain, and L. Tolhuizen, "Polynomial time algorithms for multicast network code construction," *IEEE Trans. Inf. Theory*, vol. 51, no. 6, pp. 1973–1982, Jun. 2005.

[30] T. Ho, D. R. Karger, M. Médard, and R. Kötter, "Network coding from a network flow perspective," in *Proc. IEEE Int. Symp. on Inf. Theory*, London, U.K., May 2005.

[31] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. Cambridge, MA: MIT Press and McGraw-Hill, 2001.

[32] D. Peleg, Distributed Computing: A Locality-Sensitive Approach. Philadelphia, PA, Soc. Indust. Appl. Math., 2000.

[33] L. A. Wolsey, *Integer and Combinatorial Optimization*. New York: Wiley-Intersci., Sep. 1998.

[34] A. Schriver, *Combinatorial Optimization—Polyhedra and Efficiency*. Berlin, Germany: Springer-Verlag, 2003, vol. 1.

[35] F. A. Onat, I. Stojmenovic, and H. Yanikomeroglu, "Generating random graphs for simulation of wireless ad-hoc, actuator, and internet networks," *Pervasive and Mobile Computing (Elsevier)*, vol. 4, no. 5, pp. 597–615, 2008.

Ketan Rajawat (S'06) received the B.Tech. and M.Tech. degrees in electrical engineering from Indian Institute of Technology, Kanpur, in 2007.

Since August 2007, he has been working toward the Ph.D. degree at the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis. His current research focuses on network coding and optimization in wireless networks.

Georgios B. Giannakis (M'86–SM'91–F'97) received the Diploma in electrical engineering from the National Technology University of Athens, Greece, 1981. From 1982 to 1986, he was with the University of Southern California (USC), where he received the M.Sc. degree in electrical engineering in 1983, the M.Sc. degree in mathematics in 1986, and the Ph.D. degree in electrical engineering, in 1986.

Since 1999, he has been a professor with the University of Minnesota, where he now holds an ADC Chair in Wireless Telecommunications in the ECE Department, and serves as Director of the Digital Technology Center. His general interests span the areas of communications, networking and statistical signal processing—subjects on which he has published more than 300 journal papers, 500 conference papers, 20 book chapters, two edited books, and two research monographs. Current research focuses on compressive sensing, cognitive radios, network coding, cross-layer designs, wireless sensors, social and power grid networks. He is the (co-) inventor of 20 patents issued.

Dr. Giannakis is the (co-) recipient of seven paper awards from the IEEE Signal Processing (SP) and Communications Societies, including the G. Marconi Prize Paper Award in Wireless Communications. He also received Technical Achievement Awards from the SP Society (2000), from EURASIP (2005), a Young Faculty Teaching Award, and the G. W. Taylor Award for Distinguished Research from the University of Minnesota. He is a Fellow of EURASIP, and has served the IEEE in a number of posts, including that of a Distinguished Lecturer for the IEEE-SP Society.