A Universal Stabilization Algorithm for Multicast Flows with Network Coding

Ching-Min Lien, *Member, IEEE,* Cheng-Shang Chang, *Fellow, IEEE,* and Duan-Shin Lee, *Senior Member, IEEE*

***Abstract*—In this paper, we consider a network that supports multiple multicast flows with network coding. It is well-known that network coding can be used to increase the throughput of a delay-free network. However, as there are multiple multicast flows in the network, packets might be delayed due to contention. As packets from the same flow have to be synchronized for network coding, additional buffers, known as *synchronization buffers* in the literature, are required. In certain networks, such as multistage interconnection networks in switch fabrics [17], the size of internal buffers could be quite limited. One of the key contributions of the paper is to propose a packet scheduling algorithm so that synchronization buffers can be bounded by a *finite* constant that only depends on the maximum hop count of the flows. We also show that our scheduling algorithm does not cause any throughput degradation as it can stabilize the network for any *admissible* traffic. Moreover, our algorithm is *universal* and it does not require to know the rates of the flows in the network. The main idea of our algorithm is to introduce *fictitious* packets in the dynamic frame sizing algorithm recently proposed in [4], [19] for stabilizing switches and *wired* networks (without network coding). By so doing, packets that might be stalled in synchronization buffers can be flushed out. We show that the number of fictitious packets in each frame is also bounded by a finite constant that only depends on the maximum hop count.**

***Index Terms*—Scheduling, Stability, Networks, Network Coding.**

I. Introduction

As mentioned in numerous research results in the past decade [1], [18], [16], network coding can be used to increase the throughput of a delay-free network. This can be best explained by using the classical butterfly diagram in Fig. 1(a). Assume that each link can send at most one packet in a single time slot, and the network is delay-free. Then, by using the traditional store-and-forward routing network, the source cannot send packets a and b to both sinks at the same time. On the other hand, as shown in Fig. 1(a), one can combine the two arrival packets at node 4 by using the XOR operation so that nodes 6 and 7 can receive and decode both packets a and b simultaneously.

In practice, for network coding, all the networks may not be delay-free and the problem of synchronization is introduced. That is, a packet transmitted in an arbitrary link might be a mixture of packets that follow different routes and thus experience different delays. As such, additional buffers, known as *synchronization buffers* in the literature, are needed in the network. Again, we take the multicast traffic flow in Fig. 1(a) as an example. Assume that packet a has arrived at node 4 in Fig. 1(a). Then, node 4 cannot transmit packet a directly to node 5 even though the link between them (i.e., link $(4,5)$) is available to transmit. Instead, one has to wait for packet b to arrive at node 4, combine these two packets (a and b), and transmit the packet $a+b$ until the next time link $(4,5)$ is available. Such a synchronization problem also occurs in fork-join queues for modelling parallel processing systems (see e.g., [22], [2]).

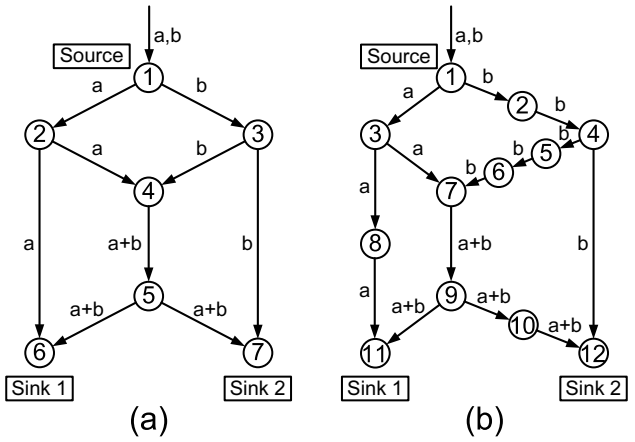

Fig. 1. (a) A classical butterfly diagram (b) A unsymmetrical butterfly diagram.

For networks with delay, it is still *theoretically* feasible to apply linear network coding as shown in [16]. However, it is difficult to find the inverse of the polynomial matrix with delay variable D in practice. As such, random network coding was proposed in [10], [5]. For random network coding, each node chooses its encoding coefficients randomly from a large enough field, and appends the coefficients in the header of all the packets it sends out. Then, all the nodes are able to decode the incoming packets once it receives an enough number of independent packets. It was analyzed theoretically [12] and simulated [6] that random network coding guarantees 100% throughput for networks with delay if the synchronization buffers are unlimited. However, it might still suffer from throughput degradation, such as removing outdated packets, if the synchronization buffers are finite.

In certain networks, such as multistage interconnection networks in switch fabrics, the size of internal buffers could be quite limited. As pointed out in [8], it is thus crucial to mitigate the problem of implementing unlimited internal buffers. For the setting with one *single* traffic flow and linear

C. -M. Lien, C. -S. Chang, and D. -S. Lee are all with the Institute of Communications Engineering, No. 101, Section 2, Kuang-Fu Road, Hsinchu, Taiwan.

E-mail: keiichi@gibbs.ee.nthu.edu.tw, cschang@ee.nthu.edu.tw, lds@cs.nthu.edu.tw.

network coding, Wu, Zhao and You [28] showed that the additional delay experienced by packets (and thus the sizes of additional synchronization buffers) could be upper bounded by the difference of path lengths of all incoming packets for each node. Moreover, Prasad and Sundar Rajan [24] discussed how the synchronization buffer distributes over all the nodes (along with the packets traversing through the network) for single traffic flow case. To see the intuition for the setting with a single traffic flow, let us consider the example in Fig. 1(b), where the path lengths for packets a and b to node 7 differ from each other. If each link is of one unit delay, then it is clear that the time difference between the arrivals of packets a and b to node 7 is simply the difference between the lengths of the two paths for these two packets to get to node 7. Thus, the difference of paths lengths can be used as an upper bound for synchroniation buffers for the setting with a single traffic flow as shown in [28].

The question that we would like to address in this paper is then

whether it is possible to support multiple (single-source) multicast flows with network coding for networks with limited internal buffers.

Our approach for this problem is to extend the dynamic frame sizing (DFS) algorithm that was previously proposed in [4] for CICQ switches and [19] for wired networks without network coding. As mentioned in these two papers, the key idea of the DFS algorithm is to emulate an ideal fluid network by providing guaranteed rate services inside a network so that internal buffers can be bounded. This is done by implementing a smooth packet schedule in [9]. Such an emulation is *dynamic* in every frame and it is based on the traffic load observed at the beginning of each frame. As such, it can stabilize all the admissible traffic without the need to know the arrival rates of flows. However, such an emulation is not perfect and there are a finite number of packets (at most two) that might be stalled in the internal buffers. These packets can only be pushed out of the network by future arrivals from the same flow.

The main difference between wired networks *without* network coding and wired networks *with* network coding is that multicast flows in networks *with* network coding are *directed acyclic graphs*, while those in networks *without* network coding are *fanout trees*. Notice that, even though one can apply convolutional network coding for multicast flows with directed cycles as shown in the literature [11], we consider only acyclic flows in this paper. Extension from trees to directed acyclic graphs requires *synchronization* in nodes where network coding is performed. This poses two challenges: (i) bounding the size of synchronization buffers, and (ii) flushing out packets stalled in internal buffers. The main contribution of this paper is to address these two challenges and show that multiple multicast flows with network coding can still be supported by using the DFS algorithm for networks with limited internal buffers. Specifically, for the problem of bounding the size of synchronization buffers, we conduct a worst case analysis. We show that the size of synchronization buffers can be bounded by a finite constant that only depends on the maximum hop count of flows. For the second challenge, our idea is to introduce a finite number of *fictitious* packets to flush out the stalled packets. We show that the network can still be stabilized for any admissible traffic and that the number of fictitious packets in every frame is also bounded by a finite constant that only depends on the maximum hop count of flows.

We now compare our work with some related ones in the literature. For fork-joined queues or so-called processing networks, Jiang et al. [15] and Huang et al. [13] proposed the Deficit Maximum Weight (DMW) and Perturbed Max-Weight (PMW) algorithms, respectively, to solve the buffer underflow problems. Both algorithms are variants of Maximum-Weighted Matching (MWM) algorithm and can be shown to be throughput-optimal. Compared to their algorithms, our dynamic frame sizing algorithm guarantees not only the stability of the networks for all the admissible traffic but also the finiteness of all the internal buffers. Moreover, in contrast with the MWM-based scheduling algorithms, the queue length information is needed only once a frame for our DFS algorithm, and communication and computation overheads are thus reduced considerably. On the other hand, Cogill et al. [7] and Parag et al. [23] studied the enlargement of the capacity region of random network coding scheme relative to the pure-routing scheme by providing the queueing analysis. In contrast, we focus on whether it is possible to support multicast traffic flows with network coding with limited internal buffers or not. Thus, in this paper, we consider the intra-flow network coding and linear network coding only, which can be shown to maximize the capacity region for single-source multicast traffic flows in the literature [29].

The rest of this paper is organized as follows. First, we describe our mathematical model in Section II. In Section III, we propose our DFS algorithm with intra-flow linear network coding. Later, we prove the finiteness of all internal buffers in Section IV-A and the finiteness of the expected frame size in Section IV-B, respectively. Then, in Section V, we perform quantitative delay analysis by computer simulations for our DFS algorithm. In Section. VI, we conclude this paper by addressing some further extensions.

II. Modeling

In this section, we first introduce the mathematical model and the notations that will be used in the paper.

A. Network as a Directed Graph

In this paper, a network is modeled as a finite directed graph $G(V, E)$, where V and E are the sets of nodes and links, respectively. For an arbitrary link $\ell \in E$, its tail end and its head end are represented as $Tail(\ell)$ and $Head(\ell)$, respectively. Link ℓ can be also represented as (v, w) if $Tail(\ell) = v \in V$ and $Head(\ell) = w \in V$. For an arbitrary node v, link ℓ is called an *incident outgoing link* (resp., *incident incoming link*) of node v if $v = Tail(\ell)$ (resp., $v = Head(\ell)$). Moreover, we denote $Out(v)$ and $In(v)$ as the collection of incident outgoing and incoming links for node v, respectively. In the rest of the paper, it is sometimes more convenient to index the links numerically (e.g., links

$1, 2, \ldots, L$) rather than as node-pairs (e.g., link (v, w) for $v, w \in V$), where L is the total number of links in the network, namely, $|E| = L$.

In this paper, we consider the usual discrete-time setting by assuming that packets are of the same size and that time is slotted so that one packet can be transmitted within a time slot in a link.

B. Multicast Traffic Flows with Intra-flow Network Coding

In this section, we introduce our assumptions for multicast traffic flows.

(A1) (External arrival processes) Assume that there are J single-source multicast flows with network coding, indexed from 1 to J. These J flows form independent compound Bernoulli processes when they arrive at the network. Specifically, there are two rates for each flow: the arrival rate λ_j and the multicast rate m_j, $j = 1, 2, \ldots, J$. With probability λ_j, a batch of m_j packets arrive at the network in every time slot. This is independent of everything else. These m_j packets are then distributed from the source node to a set of sink nodes following a specific intra network coding scheme that can be characterized by an *acyclic* subgraph $G_j(V_j, E_j)$ of $G(V, E)$, where exactly one packet needs to be transmitted in every link in G_j. Recall that, in this paper, we consider intra-flow network coding scheme only, and thus the packets could not be coded across different traffic flows.

Let $a_j(t)$ be the indicator random variable for the event that there is a batch of m_j flow j packets arriving at the network at time t. Under (A1), the sequence of random variables $\{a_j(t), t = 1, 2, \ldots\}$ are independent Bernoulli random variables with parameter λ_j.

Note that E_j is the set of links traversed by flow j. Let $S_\ell = \{j : \text{link } \ell \in E_j\}$ be the set of flows traversing link ℓ. Hence, for an arbitrary link ℓ and flow j, it is clear that $\ell \in E_j$ if and only if $j \in S_\ell$.

(A2) (Admissible traffic) We assume that the arrival rates $\{\lambda_j\}_{j=1}^J$ satisfy the following inequality

$$\rho = \max_{1 \le \ell \le L} \sum_{j \in S_\ell} \lambda_j < 1. \tag{1}$$

That is, the aggregate arrival rate in each link does not exceed its capacity.

Traffic that satisfies the rate condition in (1) is often called *admissible* in the literature.

For all the links traversed by the same traffic flow, we call the links traversed right before and after a particular link as its upstream and downstream links. Specifically, for an arbitrary link ℓ traversed by flow j, link ℓ' is called an upstream link of link ℓ for flow j if and only if ℓ' is traversed by flow j and link ℓ' is an incident incoming link for the tail end of link ℓ. Let $U(j, \ell)$ (resp., $D(j, \ell)$) be the collection of upstream links (resp., downstream links) of link ℓ for flow j. Then, we have that $\ell' \in U(j, \ell)$ if and only if $\ell' \in E_j$ and $\ell' \in In(Tail(\ell))$, or equivalently, $U(j, \ell) = E_j \cap In(Tail(\ell))$. Similarly, $D(j, \ell)$ is defined as $D(j, \ell) = E_j \cap Out(Head(\ell))$. If $|U(j, \ell)| > 1$, then synchronization is needed for flow j in link ℓ as it has to wait for all the packets from its upstream links to arrive before network coding can be performed. We now take the acyclic graph for the classical butterfly diagram in Fig. 1(a) as an example. For this acyclic graph, the multicast rate is 2 and there are two packets a and b in a batch. Since the packet traversing link $(4, 5)$ is a combination of packets traversing on both links $(2, 4)$ and $(3, 4)$, both links $(2, 4)$ and $(3, 4)$ are the upstream links for link $(4, 5)$.

C. Two-Level Queueing Mechanism

Assume that per flow queueing is used in every link, i.e., every flow has its own queues in every link it traverses. As mentioned before, to perform network coding, additional buffers are placed for synchronization. In this paper, we adopt a two-level queueing mechanism, which is similar to those used in [27], [28]. More specifically, as flow j traverses through link ℓ, namely, $j \in S_\ell$, both the tail and head ends of link ℓ have one queue for flow j. The queues placed at the tail and head ends of link ℓ for flow j are thus called the tail queue and head queue for flow j in link ℓ, and they are denoted as $q_{j,\ell}^T$ and $q_{j,\ell}^H$, respectively. The tail queue $\{q_{j,\ell}^T\}$ stores packets that wait for being transmitted through link ℓ, and the head queue $\{q_{j,\ell}^H\}$ stores packets that wait for being mixed and transmitted to the tail queues of its downstream links later. In this paper, all the tail queues of the incident outgoing links of source nodes are called *ingress queues*, and all the other queues (including head and tail queues) are called *internal queues*.

We assume that each queue is started from an empty system at time 0. We now define $x_{j,\ell}^T(t)$ as the number of packets in the tail queue $q_{j,\ell}^T$ at time t. Then, the governing equation for an arbitrary tail queue $q_{j,\ell}^T$ can be written as

$$x_{j,\ell}^T(t+1) = x_{j,\ell}^T(t) + a_{j,\ell}^T(t+1) - b_{j,\ell}^T(t+1), \tag{2}$$

where $a_{j,\ell}^T(t)$ and $b_{j,\ell}^T(t)$ are the number of packets arriving at $q_{j,\ell}^T$ and departing from $q_{j,\ell}^T$ at time t, respectively. Also, for an arbitrary head queue $q_{j,\ell}^H$, $a_{j,\ell}^H(t)$, $b_{j,\ell}^H(t)$ and $x_{j,\ell}^H(t)$ are defined similarly. In this paper, we assume that the number of ingress queues for flow j is m_j, which coincides with the size of the batch of the arrival packets for flow j as in the assumption (A1). Then, whenever a batch of m_j packets arrives at the network, each one of the m_j ingress queues would receive exactly one of these packets. Thus, if $q_{j,\ell}^T$ is an ingress queue for flow j, then $a_{j,\ell}^T(t) = 1$ when there is a batch of m_j flow j packets arriving at the network at time t. In other words, for an ingress queue for flow j, we have $a_{j,\ell}^T(t) = a_j(t)$.

D. Implementation by Linear Network Coding

In this section, we discuss how one finds the acyclic graph $G_j(V_j, E_j)$ that characterizes the network coding scheme for flow j in (A1). If the multicast rate m_j is not greater than the minimum cut from the source node to each sink node, then it is well-known (see e.g., [14]) that there is a polynomial time algorithm for finding a linear network coding scheme

that provides the multicast rate m_j from the source node to each sink node. The algorithm in [14] starts from finding m_j link disjoint paths from the source node to each sink node. The union of these link disjoint paths then forms an acyclic graph. Following the topological order of the acyclic graph, a linear network coding scheme can be found from the source node to each sink node by selecting local coding coefficients at each node (over some Galois field F_q for q larger than the number of sinks for all the flows) so that the m_j link disjoint paths from the source node to each sink node contain m_j linearly independent packets. To illustrate this, suppose that the source node of flow j is $s_j \in V$. For a batch of m_j packets from flow j, we denote $X_{j,k}$ as the k^{th} packet in that batch. Also, denote by $Y_{j,\ell}$ as the packet transmitted on link $\ell \in E_j$. Then, $Y_{j,\ell}$ can be represented as

$$Y_{j,\ell} = \begin{cases} \sum_{k=1}^{m_j} \alpha_{j,k,\ell} X_{j,k}, & \text{if } Tail(\ell) = s_j, \\ \sum_{k \in U(j,\ell)} \beta_{j,k,\ell} Y_{j,k}, & \text{otherwise,} \end{cases} \tag{3}$$

where $\{\alpha_{j,k,\ell}\}$ and $\{\beta_{j,k,\ell}\}$ are *local coding coefficients* for flow j. These local coding coefficients are chosen in the way that there are m_j linearly independent packets to each sink node. That is, the packets $Y_{j,\ell}$ transmitted on the incident outgoing links of source node s_j are linear combinations of the m_j arriving packets $\{X_{j,k}\}$, $k = 1, 2, \ldots, m_j$, and the packet $Y_{j,\ell}$ transmitted on other links are linear combinations of the traffic transmitted on their corresponding upstream links $U(j, \ell)$. Moreover, each sink node v can decode all the m_j arriving packets $\{X_{j,k}\}$, $k = 1, 2, \ldots, m_j$, by using the packets transmitted in its incident incoming links.

We note that for the synchronization problem, if linear network coding is used, then it is possible that some local encoding coefficients at a certain link are 0. Thus, it may not be necessary to wait for all the packets from the upstream links to arrive before network coding can be performed. In this paper, we do not distinguish this case from the general setting as we will show later that our algorithm can still stabilize the network for any admissible traffic.

An implementation like the one discussed above is known as the generation-based network coding in the literature (see e.g., Chou and Wu in [6]). For the j^{th} (multicast) traffic flow, the external arriving packets are in batches of m_j packets. All the coded packets related to the same batch are said to be in the same generation, and the packets are only mixed within the same generation. In this paper, we will not discuss cross-generation network coding schemes.

III. Dynamic Frame Sizing Algorithm

In [4], [19], the dynamic frame sizing (DFS) algorithm has been used for stabilizing queues in switches and wired networks without knowing the arrival rates. In the DFS algorithm, time is partitioned into frames, where the frame size is not fixed and is determined at the beginning of each frame. As described in [4], [19], the main idea of the DFS algorithm is to determine the minimum frame size at the beginning of a frame so that the backlog observed at the ingress queue(s) of each flow at the beginning of the frame can be cleared by the end of the frame. To ensure the number of packets of each flow inside the network is bounded above by a finite constant, the DFS algorithm then provides each flow in a frame a guaranteed rate that is proportional to the backlog observed at the ingress queue(s) at the beginning of that frame. As long as the expected size of each frame is finite, the expected backlog at each queue remains finite.

As mentioned in the introduction, there are two challenges to apply the DFS algorithm for network coding: (i) bounding the size of synchronization buffers, and (ii) flushing out packets stalled in internal buffers. For the problem of bounding the size of synchronization buffers, we first derive the governing equations for synchronization in Section III-D and then conduct a worst case analysis in Section IV-A. For the second challenge, our idea, as described in Section III-A below, is to adopt gated services and introduce a finite number of *fictitious* packets to flush out the stalled packets.

A. Gated Services and Fictitious Packets

Unlike the DFS algorithm in [4], [19], we adopt gated services in this paper. Specifically, packets that arrive during a frame are stored in the ingress queues and can only be served in the next frame. As mentioned before, one problem for the DFS algorithm in [4], [19] is that there are a finite number of packets (at most two) that might be stalled in the internal buffers. These packets can only be pushed out of the network by future arrivals from the same flow. To solve this problem, our idea is to introduce fictitious packets in the DFS algorithm to flush out all the real packets in the network. By doing so, there are no real packets in the internal buffer at the beginning of a frame and we can simply drop all the remaining packets in the internal buffers as they are all fictitious packets. Let h_j be the maximum hop count of the acyclic graph $G_j(V_j, E_j)$, i.e., the length of the longest path from the source node to any sink node. Then, prior to determining the new frame size, additional $h_j - 1$ fictitious packets are appended to the end of the ingress queues of flow j if there are real packets in the ingress queues. We will show later this is enough to flush out all the real packets in the internal buffers.

B. Determine the Frame Size

The size of the frame is determined by the minimum clearance time at the beginning of each frame. Denote by τ_n the last time slot of the $(n-1)^{th}$ frame and by $\tau_n + 1$ the beginning time slot of the n^{th} frame. Since we adopt gated services, the number of real packets stored in the ingress queues at the beginning of the n^{th} frame are those arriving in the $(n-1)^{th}$ frame. This implies that all the ingress queues for flow j have the same number of real packets at the beginning of a frame. As such, we can use any ingress queue for determining the size of a new frame. Now suppose that there are $x_j(\tau_n)$ packets (including the fictitious packets) stored in an ingress queue for flow j at the beginning of the n^{th} frame. Recall that S_ℓ is the collection of all the flows traversing link ℓ, namely, $S_\ell = \{j | \ell \in E_j\}$. Then, the size of the n^{th} frame is set to be

$$T_n = \max_{1 \le \ell \le L} \sum_{j \in S_\ell} x_j(\tau_n). \tag{4}$$

Note that T_n is the minimum time T such that $\sum_{j\in S_\ell} x_j(\tau_n) \le T$ for all $\ell = 1, 2, \ldots, L$. If we view the backlogs at the ingress queues as fluids, then T_n is the minimum time to clear all the backlogs under the constraint that each link is of capacity one (i.e., one packet per time slot). If there are no real packets in any of the ingress queues, we simply set $T_n = 1$. Thus, we have $T_1 = 1$ as we assume the network is started from an empty network.

C. Smooth Scheduling

As in the original DFS algorithm, we schedule packets in the per-flow queues with the rates proportional to their sizes at the beginning of a frame. Specifically, for each tail queue $q^T_{j,\ell}$ with $\ell \in E_j$, we generate $x_j(\tau_n)$ tokens in the n^{th} frame (for sending packets to downstream links or to sink nodes), and the k^{th} token in the n^{th} frame is assigned with the eligible time $\tau_n + 1 + \lfloor (k-1)T_n/x_j(\tau_n) \rfloor$ and the deadline $\tau_n + \lceil kT_n/x_j(\tau_n) \rceil$. To schedule under the constraint of eligible times and deadlines, there is a token arbitrator at each link for sending packets from the tail of the link to the head of the link. In each time slot of the n^{th} frame, i.e., the time interval $[\tau_n + 1, \tau_n + T_n]$, the ℓ^{th} link selects one eligible token with the earliest deadline (the EDF policy in the literature) among all the remaining tokens for all the tail queues in link ℓ, and removes that token. Ties are broken arbitrarily. A tail queue with a selected token is then allowed to send a packet (through the link to its head queue) in that time slot.

As a direct consequence of the well-known result for a smooth schedule (see e.g., Lemma 3.1 in [19]), every token is selected not later than its deadline. In the following lemma, we use this result to derive an upper bound and a lower bound on the total number of tokens selected for a particular flow in a link traversed by that flow.

Lemma 1 *For each link ℓ traversed by flow j, let $C_{j,\ell}(t)$ be the total number of tokens selected for flow j in link ℓ in the time interval $[\tau_n, \tau_n + t]$. Under the smooth schedule described in this section, $C_{j,\ell}(t)$ is upper and lower bounded as follows:*

$$\left\lfloor \frac{t}{T_n} x_j(\tau_n) \right\rfloor \le C_{j,\ell}(t) \le \left\lceil \frac{t}{T_n} x_j(\tau_n) \right\rceil. \quad (5)$$

Proof: First, the number of tokens selected by time t cannot be greater than the number of tokens generated by time t. This shows the upper bound in (5). On the other hand, as every token is selected not later than its deadline, the number of tokens selected by time t is lower bounded by the number of tokens with deadlines not later than t. Hence, $C_{j,\ell}(t)$ is upper and lower bounded as shown in (5). ■

D. Synchronization for Network Coding

As we adopt gated services and drop all the packets in internal buffers at the beginning of a frame, it suffices to treat each frame *separately*, where every internal buffer is started from an empty queue and there is no future arrivals at any ingress queue (during that frame). In this paper, we assume that the time spent for network coding is negligible (comparing to the packet transmission). As such, the delay is mainly due to synchronization and queueing.

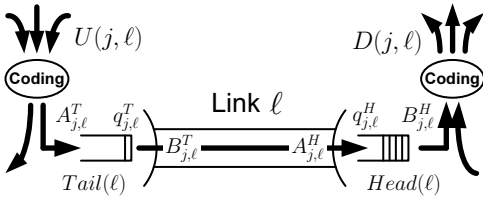

Fig. 2. Synchronization of the head queue and the tail queue of link ℓ for flow j.

Now we derive the governing equations for all the head queues and tail queues. For the ease of our presentation, let us reset the last time slot of the $(n-1)^{th}$ frame, τ_n, to 0 and set time t as the t^{th} time slot of the n^{th} frame. Suppose that link ℓ is traversed by flow j, i.e., $\ell \in E_j$. Let $A^T_{j,\ell}(t)$ and $B^T_{j,\ell}(t)$ (resp., $A^H_{j,\ell}(t)$ and $B^H_{j,\ell}(t)$) be the cumulative number of packets arriving at and departing from tail queue $q^T_{j,\ell}$ (resp., head queue $q^H_{j,\ell}$) by time t, respectively. Since the departure of a packet from a tail queue in link ℓ is an arrival to the head queue in link ℓ (see Fig. 2), we have

$$A^H_{j,\ell}(t) = B^T_{j,\ell}(t). \quad (6)$$

Let $c_{j,\ell}(t)$ be the indicator variable for the event that the token of the tail queue $q^T_{j,\ell}$ is selected. Then, the governing equation (2) for internal tail queue $q^T_{j,\ell}$ can be rewritten as

$$x^T_{j,\ell}(t+1) = \max[0, x^T_{j,\ell}(t) + a^T_{j,\ell}(t+1) - c_{j,\ell}(t+1)]. \quad (7)$$

Recursively expanding the governing equation (7) with the initial condition $x^T_{j,\ell}(0) = 0$ yields

$$x^T_{j,\ell}(t) = \max_{0\le s\le t} [A^T_{j,\ell}(t) - A^T_{j,\ell}(s) - (C_{j,\ell}(t) - C_{j,\ell}(s))]. \quad (8)$$

Since packets that have arrived at queue $q^T_{j,\ell}$ are either still in the buffer or have departed from the queue, we have $A^T_{j,\ell}(t) = x^T_{j,\ell}(t) + B^T_{j,\ell}(t)$. Using this in (8), we then have

$$B^T_{j,\ell}(t) = \min_{0\le s\le t} \left[A^T_{j,\ell}(s) + (C_{j,\ell}(t) - C_{j,\ell}(s))\right]. \quad (9)$$

Note from (9) that the tail queue $q^T_{j,\ell}$ is in fact a work conserving link with a time varying capacity defined in Example 2.3.2 of the book [3]. Moreover, since $B^T_{j,\ell}(s) \le A^T_{j,\ell}(s)$, it follows from (9) that for all $s \le t$

$$B^T_{j,\ell}(t) - B^T_{j,\ell}(s) \le C_{j,\ell}(t) - C_{j,\ell}(s). \quad (10)$$

As discussed before, network coding can only be performed for the k^{th} packet arriving at internal tail queue $q^T_{j,\ell}$ if all of its upstream head queues in $U(j,\ell)$ have received at least k packets. Since we assume the time to perform network coding is negligible, the time that the k^{th} packet arriving at tail queue $q^T_{j,\ell}$ is exactly the time that all of its upstream head queues in $U(j,\ell)$ have received at least k packets. Thus,

$$A^T_{j,\ell}(t) = \min_{\ell' \in U(j,\ell)} A^H_{j,\ell'}(t). \quad (11)$$

Note that network coding is done *simultaneously* for the k^{th} packet in every downstream tail queue of $U(j,\ell)$. Thus, the

k^{th} packet arrives at every downstream tail queue of $U(j,\ell)$ *synchronously*. Moreover, once network coding is done, the k^{th} packet in every upstream head queue in $U(j,\ell)$ can be removed from the buffer at the same time. Similarly, the k^{th} packet can be removed from head queue $q^H_{j,\ell}$ if all the head queues pointing to the same node as link ℓ, i.e., the set of head queues in $In(Head(\ell)) \cap E_j$, have received at least k packets. This then leads to

$$B^H_{j,\ell}(t) = \min_{\ell' \in In(Head(\ell)) \cap E_j} A^H_{j,\ell'}(t). \tag{12}$$

As mentioned before, synchronization for network coding is carried out for every upstream head queue even though some of the local coding coefficients in a linear coding scheme might be 0 for coding downstream packets. For example, suppose that flow j traverses through links 1 to 5 as shown in Fig. 3, where we set the local coding coefficients $\beta_{j,3,4} = \beta_{j,1,5} = 0$. Note that links 4 and 5 have the same set of upstream links for flow j as $U(j,4) = U(j,5) = \{1,2,3\}$ even though the packets traversed through either link 4 or link 5 are only linear combinations of the packets transmitted in a proper subset of link 1, 2 and 3. Suppose that there is a flow j packet that arrives at the head queue of link 3 at time t. Then, the head queues of all the upstream links for flow j (i.e., $q^H_{j,1}$, $q^H_{j,2}$ and $q^H_{j,3}$) are not empty at the end of time slot t (as shown in Fig. 3(a)). As a result, for flow j, the tail queues for links 4 and 5 can obtain a new packet as the linear combination of the packets in head queues $q^H_{j,1}$, $q^H_{j,2}$ and $q^H_{j,3}$ using the corresponding local coding coefficients. Then the head queues for links 1, 2 and 3 can remove one packet after network coding is done (as shown in Fig. 3(b)).

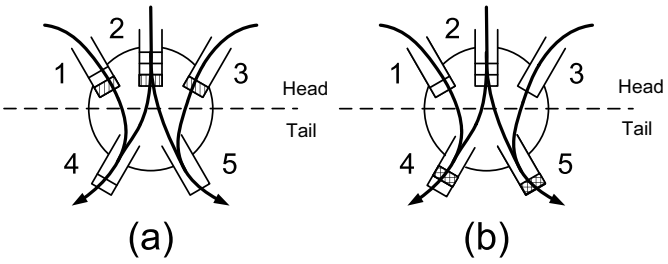

Fig. 3. An illustrating example for synchronization (a) Before (b) After

IV. Finite internal buffer and frame bound

In Section IV-A, we show the finiteness for both internal tail queues (for transmission) and head queues (for synchronization under the setup of network coding) under the DFS algorithm. Moreover, the proof for the finiteness for the expected frame size under the DFS algorithm will be given in Section IV-B.

A. Finite Internal Buffer

In this section, we show that each internal tail queue is upper bounded by two packets and the size of an arbitrary internal head queue is bounded by a constant that only depends on the maximum hop count, i.e., the length of the longest path length to the source. We now formally define the maximum hop count of link ℓ for flow j as follows.

Definition 2 *For each link ℓ traversed by flow j (i.e., $\ell \in E_j$), the maximum hop count $h_{j,\ell}$ is defined recursively as*

$$h_{j,\ell} = 1 + \max_{\ell' \in U(j,\ell)} h_{j,\ell'}, \tag{13}$$

where $h_{j,\ell} = 1$ if link ℓ is the first link traversed by flow j immediately after departing from the source node, namely, an incident outgoing link for source node s_j. Notice that (13) is well-defined according to the assumption that the graph $G_j(V_j, E_j)$ is acyclic.

Notice that, for fixed j, $h_{j,\ell}$ can be viewed as the number of hops of the longest path from the tail of link ℓ to the source node. We now take the multicast flow j from source node 1 to sink nodes 11 and 12 in Fig. 1(b) as an example. According to (13), one can verify that $h_{j,(3,7)} = 2$. Moreover, we have that $h_{j,(7,9)} = 6 \neq h_{j,(3,7)} + 1$ since the path along nodes $1, 2, 4, 5, 6, 7$ is the longer path from the source (i.e., node 1) to the tail end 7 of link $(7, 9)$. We now ready to show that the upper bounds of internal tail and head queues in the following theorem.

Theorem 3 *Let $h_j = \max_{\ell \in E_j} h_{j,\ell}$ be maximum hop count for flow j.*

(i) *There are at most two packets in an arbitrary internal tail queue, namely, $x^T_{j,\ell}(t) \leq 2$ for all $\ell \in E_j$, $j = 1, 2, \ldots, J$ and $t \geq 0$.*

(ii) *The cumulative number of departures from tail queue $q^T_{j,\ell}$ of link ℓ for flow j by time t can be lower bounded as follow:*

$$B^T_{j,\ell}(t) \geq \left\lfloor \frac{t}{T_n} x_j(\tau_n) \right\rfloor - (h_{j,\ell} - 1). \tag{14}$$

(iii) *The size of an arbitrary internal head queue is upper bounded by the maximum hop count for flow j, i.e.,*

$$x^H_{j,\ell}(t) \leq h_j \tag{15}$$

for all $\ell \in E_j$, $j = 1, 2, \ldots, J$, and $t \geq 0$,

(iv) *At the end of each frame, other than the packets arrived during this frame and gated in ingress queues, all the packets remaining in the network are all fictitious packets and are free to be removed at the beginning of the next frame.*

Proof: (i) First, from (8), the queue length $x^T_{j,\ell}(t)$ of internal tail queue $q^T_{j,\ell}$ can be represented as

$$x^T_{j,\ell}(t) = \max_{0 \leq s \leq t} [(A^T_{j,\ell}(t) - A^T_{j,\ell}(s)) - (C_{j,\ell}(t) - C_{j,\ell}(s))].$$

According to (11) and (6), we have that

$$A^T_{j,\ell}(t) - A^T_{j,\ell}(s) = \min_{\ell' \in U(j,\ell)} B^T_{j,\ell'}(t) - \min_{\ell'' \in U(j,\ell)} B^T_{j,\ell''}(s). \tag{16}$$

Assume that $\min_{\ell'' \in U(j,\ell)} B^T_{j,\ell''}(s)$ occurs at $\ell'' = \ell_1$ for some $\ell_1 \in U(j,\ell)$, namely,

$$\min_{\ell'' \in U(j,\ell)} B^T_{j,\ell''}(s) = B^T_{j,\ell_1}(s). \tag{17}$$

Then, it follows from (16) and (17) that

$$A^T_{j,\ell}(t) - A^T_{j,\ell}(s) = \min_{\ell' \in U(j,\ell)} B^T_{j,\ell'}(s) - B^T_{j,\ell_1}(s)$$
$$\leq B^T_{j,\ell_1}(t) - B^T_{j,\ell_1}(s) \leq C_{j,\ell_1}(t) - C_{j,\ell_1}(s),$$

where we use (10) in the last inequality. According to (5), we have that $\lfloor \frac{t}{T} x_j(\tau_n) \rfloor - \lceil \frac{s}{T} x_j(\tau_n) \rceil \leq C_{j,\ell'}(t) - C_{j,\ell'}(s) \leq \lceil \frac{t}{T} x_j(\tau_n) \rceil - \lfloor \frac{s}{T} x_j(\tau_n) \rfloor$ for all $\ell' \in E_j$. Hence, it yields that

$$\left(A^T_{j,\ell}(t) - A^T_{j,\ell}(s)\right) - \left(C_{j,\ell}(t) - C_{j,\ell}(s)\right) \leq 2,$$

for all $0 \leq s \leq t$, and thus $x^T_{j,\ell}(t) \leq 2$ for all $\ell \in E_j$ and $j = 1, 2, \ldots, J$.

(ii) We now prove (14) by induction. We first consider an arbitrary ingress queue $q^T_{j,\ell}$. As described in Section III-C for smooth scheduling, there are $x_j(\tau_n)$ tokens generated for $q^T_{j,\ell}$ for the n^{th} frame. Notice that, at the beginning of the n^{th} frame, there are totally $x_j(\tau_n)$ packets in $q^T_{j,\ell}$. Hence, every time a token is selected in an ingress queue there is a packet transmitted through link ℓ from $q^T_{j,\ell}$ to $q^H_{j,\ell}$. Recall that $C_{j,\ell}(t)$ is the cumulative number of tokens selected for flow j in link ℓ by time t (as we have reset the time τ_n back to 0 for the ease of our presentation). Then, there are at least $C_{j,\ell}(t)$ packets departing from $q^T_{j,\ell}$, and thus, from Lemma 1,

$$B^T_{j,\ell}(t) = C_{j,\ell}(t) \geq \left\lfloor \frac{t}{T_n} x_j(\tau_n) \right\rfloor = \left\lfloor \frac{t}{T_n} x_j(\tau_n) \right\rfloor - (h_{j,\ell} - 1),$$

where we use the fact that $h_{j,\ell} = 1$ for an arbitrary ingress queue $q^T_{j,\ell}$ in the last equality.

We now assume that (14) holds for all the tail queues with $h_{j,\ell} = 1, 2, \ldots, m$ as our induction hypothesis. Consider a tail queue $q^T_{j,\ell}$ with $h_{j,\ell} = m + 1$. According to (9), (11) and (6), we have that

$$B^T_{j,\ell}(t) = \min_{0 \leq s \leq t} [A^T_{j,\ell}(s) + (C_{j,\ell}(t) - C_{j,\ell}(s))]$$
$$= \min_{0 \leq s \leq t} \left[\left(\min_{\ell' \in U(j,\ell)} B^T_{j,\ell'}(s) \right) + (C_{j,\ell}(t) - C_{j,\ell}(s)) \right].$$

Note from (13) that $\max_{\ell' \in U(j,\ell)} h_{j,\ell'} = m$ and thus $h_{j,\ell'} \leq m$ for all $\ell' \in U(j,\ell)$. According to the induction hypothesis, we have that $B^T_{j,\ell'}(s) \geq \lfloor x_j(\tau_n) s / T_n \rfloor - (h_{j,\ell'} - 1)$ for all $\ell' \in U(j,\ell)$. Moreover, according to (5), we have that $C_{j,\ell}(t) - C_{j,\ell}(s) \geq \lfloor x_j(\tau_n) t / T_n \rfloor - \lceil x_j(\tau_n) s / T_n \rceil$ for all $\ell \in E_j$. Hence, for all $\ell' \in U(j,\ell)$ and $0 \leq s \leq t$, we have that,

$$\left(\min_{\ell' \in U(j,\ell)} B^T_{j,\ell'}(s) \right) + (C_{j,\ell}(t) - C_{j,\ell}(s))$$
$$\geq \left\lfloor \frac{s}{T_n} x_j(\tau_n) \right\rfloor - \left(\max_{\ell' \in U(j,\ell)} h_{j,\ell'} - 1 \right)$$
$$+ \left\lfloor \frac{t}{T_n} x_j(\tau) \right\rfloor - \left\lceil \frac{s}{T_n} x_j(\tau_n) \right\rceil.$$

Using the fact that $\lceil x \rceil - \lfloor x \rfloor \leq 1$ for all x yields (14).

(iii) Also, from (6), (10) and (5), we have that

$$A^H_{j,\ell}(t) = B^T_{j,\ell}(t) \leq C_{j,\ell}(t) \leq \left\lceil \frac{t}{T_n} x_j(\tau_n) \right\rceil.$$

On the other hand, it follow from (12) and (6) that

$$B^H_{j,\ell}(t) = \min_{\ell' \in In(Head(\ell)) \cap E_j} B^T_{j,\ell'}(t)$$
$$\geq \min_{\ell' \in In(Head(\ell)) \cap E_j} \left(\left\lfloor \frac{t}{T_n} x_j(\tau_n) \right\rfloor - (h_{j,\ell'} - 1) \right)$$
$$= \left\lfloor \frac{t}{T_n} x_j(\tau_n) \right\rfloor + 1 - \max_{\ell' \in In(Head(\ell)) \cap E_j} h_{j,\ell'}, \quad (18)$$

where we use (14) in the inequality. Hence, for the head queue $q^H_{j,\ell}$, we have that

$$x^H_{j,\ell}(t) = A^H_{j,\ell}(t) - B^H_{j,\ell}(t)$$
$$\leq \left\lceil \frac{t}{T_n} x_j(\tau_n) \right\rceil - \left\lfloor \frac{t}{T_n} x_j(\tau_n) \right\rfloor - 1 + \max_{\ell' \in In(Head(\ell)) \cap E_j} h_{j,\ell'}$$
$$\leq \max_{\ell' \in In(Head(\ell)) \cap E_j} h_{j,\ell'} \leq h_j,$$

where we use the fact that h_j is the maximum hop count for flow j in the last inequality.

(iv) Note that the head queues at the sink nodes are the last queues the packets traverse through the network. Thus, we now consider the departure process for the head queue $q^H_{j,\ell}$ of an arbitrary incident incoming link ℓ of a sink node for flow j. At the end of the n^{th} frame (i.e., the end of time slot T_n), we have from (18) that

$$B^H_{j,\ell}(T_n) \geq x_j(\tau_n) + 1 - \max_{\ell' \in In(Head(\ell)) \cap E_j} h_{j,\ell'}. \quad (19)$$

On the other hand, we recall that the last $h_j - 1$ packets for flow j are all fictitious packets. Thus, in the n^{th} frame, only the first $x_j(\tau_n) - (h_j - 1)$ packets sent out from the source node into the network are real packets. Then, it follows from (19) that all the non-fictitious packets are sent out and leave the network at the end of the n^{th} frame. In other words, all the packets remaining in the internal queues are all fictitious packets, and it makes no harm to remove them from the network at the beginning of the next frame.

■

B. Logarithm Frame Size

Our main result is that, for the input traffic satisfying the assumptions in (A1) and (A2), the expectation of frame size $E[T_n]$ is bounded for each n under the DFS algorithm, as shown in Theorem 4.

Theorem 4 *Assume that the input traffic satisfies (A1) and (A2). Let $S_{\max}$ be the maximum number of flows traversing a single link, i.e., $S_{\max} = \max_{1 \leq \ell \leq L} |S_\ell|$ and $h_{\max}$ be the maximum hop count among all the j flows, i.e., $h_{\max} = \max_{1 \leq j \leq J} h_j$. Then under the DFS algorithm, we have for $n > 1$*

$$\log E[e^{\theta^* T_n}] \leq \theta^* + \frac{2 \log L + 2\theta^* S_{\max}(h_{\max} - 1)}{1 - \rho}, \quad (20)$$

where θ^* is the unique positive solution of

$$\frac{e^\theta - 1}{\theta} = \frac{1 + \rho}{2\rho}. \quad (21)$$

As a result, the expectation of the frame size $E[T_n]$ is bounded by $1+\frac{2\log L+2\theta^* S_{\max}(h_{\max}-1)}{\theta^*(1-\rho)}$, and the DFS algorithm achieves 100% throughput.

As a direct consequence of Theorem 4, the expected number of packets in each ingress queue is also finite. In conjunction with the bound for an internal queue in Theorem 3, the DFS algorithm stabilizes the network for the compound Bernoulli traffic described in (A1) and (A2).

Proof: From Theorem 3 (iv), the number of real packets stored in ingress queue $q_{j,\ell}^T$ at the beginning of the n^{th} frame is the same as the number of batches that arrives during the $(n-1)^{th}$ frame. Adding the h_j-1 fictitious packets, we know that the queue length of an ingress queue $q_{j,\ell}^T$ at the beginning of the n^{th} frame is bounded by the sum of the number of batches that arrive during the $(n-1)^{th}$ frame and the h_j-1 fictitious packets, i.e.,

$$x_j(\tau_{n+1}) \le \sum_{t=\tau_n+1}^{\tau_{n+1}} a_j(t) + (h_j-1). \tag{22}$$

With (4) and (22), we have for $\theta>0$ that

$$\begin{aligned} e^{\theta T_{n+1}} &\le \max_{1\le\ell\le L} \exp\left[\theta\left(\sum_{t=\tau_n+1}^{\tau_{n+1}}\sum_{j\in S_\ell} a_j(t) + |S_\ell|(h_j-1)\right)\right] \\ &\le \sum_{\ell=1}^{L} \exp\left[\theta\left(\sum_{t=\tau_n+1}^{\tau_{n+1}}\sum_{j\in S_\ell} a_j(t) + |S_\ell|(h_j-1)\right)\right], \end{aligned}$$

where we use the inequality that $\max(x_1,x_2)\le x_1+x_2$ for $x_1,x_2\ge 0$. Since we assume that the arrival processes are independent compound Bernoulli processes in (A1), we know that $\{a_j(t)\}_{t=1}^{\infty}$ are i.i.d Bernoulli random variables with parameter λ_j. It then follows that

$$\begin{aligned} &E[e^{\theta T_{n+1}}|T_n] \\ &\le e^{\theta S_{\max}(h_{\max}-1)} \sum_{\ell=1}^{L}\left\{E\left[\exp\left(\theta\sum_{j\in S_\ell} a_j(1)\right)\right]\right\}^{T_n}, \end{aligned}$$

where we use the fact $S_{\max}=\max_{1\le\ell\le L}|S_\ell|$ and $h_{\max}=\max_{1\le j\le J} h_j$. Note that

$$\begin{aligned} &\log E\left[\exp\left(\theta\sum_{j\in S_\ell} a_j(1)\right)\right] = \sum_{j\in S_\ell}\log E\left[\exp\left(\theta a_j(1)\right)\right] \\ &= \sum_{j\in S_\ell}\log\left(\lambda_j e^{\theta}+1-\lambda_j\right) \le \sum_{j\in S_\ell}\lambda_j\left(e^{\theta}-1\right), \end{aligned} \tag{23}$$

where we use $\log(1+x)\le x$ for nonnegative x. According to (A2), we have that $\rho=\max_{1\le\ell\le L}\sum_{j\in S_\ell}\lambda_j<1$, and thus

$$E[e^{\theta T_{n+1}}|T_n] \le e^{\theta S_{\max}(h_{\max}-1)} L\exp(\rho T_n(e^{\theta}-1)). \tag{24}$$

Taking expectation on both sides of (24) yields

$$E[e^{\theta T_{n+1}}] \le e^{\theta S_{\max}(h_{\max}-1)} LE\left[\exp\left(\rho T_n(e^{\theta}-1)\right)\right]. \tag{25}$$

As θ^* is the unique positive solution of (21), we can rewrite (25) as

$$E[e^{\theta^* T_{n+1}}] \le e^{\theta^* S_{\max}(h_{\max}-1)} LE[e^{\theta^* T_n(1+\rho)/2}]. \tag{26}$$

Since $\phi(\theta)=\log E[e^{\theta T_n}]$ is convex in θ (see e.g., [3, Proposition 7.1.8]) and $\rho<1$, we have that

$$\log E[e^{\theta^* T_n(1+\rho)/2}] \le \frac{1+\rho}{2}\log E[e^{\theta^* T_n}]. \tag{27}$$

Using (27) and (26) yields

$$\begin{aligned} &\log E[e^{\theta^* T_{n+1}}] \\ &\le \log L + \theta^* S_{\max}(h_{\max}-1) + \frac{1+\rho}{2}\log E[e^{\theta^* T_n}]. \end{aligned} \tag{28}$$

Since $T_1=1$, it is easy to verify (20) from induction by using (28). Now we use (20) to show the bound of the frame size in Theorem 4. Since $e^{\theta x}$ is convex in x, it follows from Jensen's Inequality that

$$\begin{aligned} E[T_n] &\le \frac{1}{\theta^*}\log E[e^{\theta^* T_n}] \\ &\le 1+\frac{2\log L+2\theta^* S_{\max}(h_{\max}-1)}{\theta^*(1-\rho)}. \end{aligned} \tag{29}$$

Since the expected frame size is finite, it follows from the standard theory for regenerative processes [25] that the DFS algorithm achieves 100% throughput. ■

V. Simulation Results

In this section, we perform quantitative delay analysis for our DFS algorithm by computer simulations. For this, we consider the network as shown in Fig. 4, where all the links are of capacity one. The upper half of the network is basically the same as the example in [21, Fig. 1], where it was shown in [21] that it is impossible to achieve 100% throughput for certain admissible multicast traffic *without internal buffers in the network*. On the other hand, the lower half of the networks are the canonical butterfly diagrams in [18], where the network coding can be used to expand the capacity region. Similar to [21], we assume that there are 4 multicast traffic flows traversing on the network, labeled from 1 to 4. Flows 1 and 2 (resp. 3 and 4) are of the same source node, node 1 (resp., node 2), and the sets of their sink nodes are $\{3,4,5,6\}$ and $\{7,8,9,10\}$ (resp., {3,4,7,8} and $\{5,6,9,10\}$), respectively. Also, we assume that the arrival processes of all the 4 flows are compound Bernoulli processes with the same arrival rate, where each batch of arrival consists of two packets as considered in (A1), Section II-B. Since each link is of capacity one, the maximum arrival rate of each flow is thus 0.5.

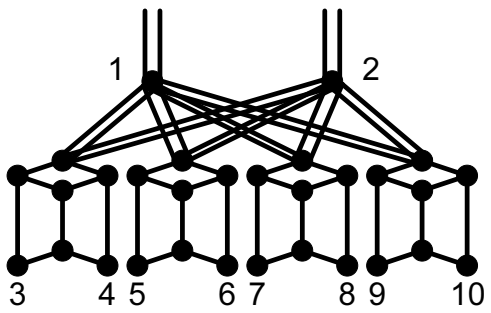

Fig. 4. The network considered in the simulation.

We now consider three different transmission schemes for the network: (i) our DFS algorithm, (ii) pure routing (without

network coding), and (iii) network coding without internal buffers. Clearly, for the pure routing scheme, the butterfly diagrams in the lower half network become the bottleneck when the arrival rate exceeds 0.25 (as illustrated in [18]). Thus, its maximum throughput is 50%. On the other hand, for the network coding scheme without internal buffer, the upper half of the network becomes the bottleneck when the arrival rate exceeds 0.375 (as illustrated in [21]). Thus, its maximum throughput is 75%. These bottlenecks are clearly shown in our simulations in Fig. 5. In contrast, even though the average packet delay of the DFS algorithm is slight higher than other two schemes in light traffic, our algorithm indeed can achieve 100% throughput and all the internal buffers are guaranteed to be finite as proved in Theorem 3.

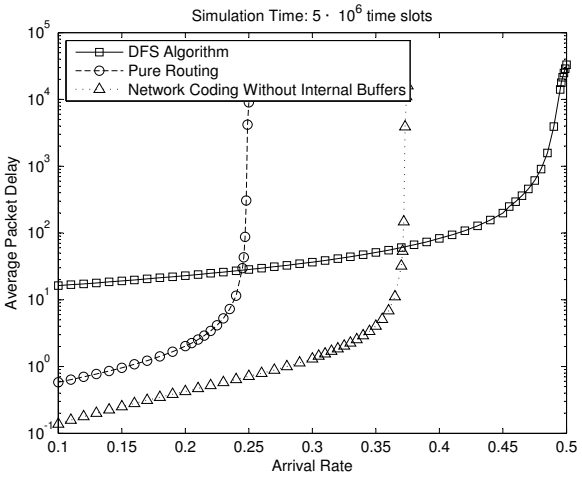

Fig. 5. The comparison of average packet delays for i) DFS algorithm, ii) pure routing (without network coding), and iii) network coding without internal buffers.

VI. Conclusion

One of the key problems for applying network coding to support multiple multicast traffic flows in a network is to deal with the issue of *synchronization*. To address this problem, we proposed in this paper a universal stabilization algorithm based on the dynamic frame sizing algorithm in [4], [19]. We showed that our algorithm indeed stabilizes the network for any admissible traffic and this is done without the need to know the arrival rates of the multicast flows. Moreover, the size of synchronization buffers can be bounded by the maximum hop count of flows. One main difference between our algorithm and the original DFS algorithm in [4], [19] is the use of *fictitious packets* to flush out real packets that might be stalled inside the network. By so doing, every frame can be treated *separately* and the proofs for the finiteness of internal buffers are much simpler.

One possible extension of our work is to support multiple multicast traffic flows with network coding in *wireless* networks, where only certain sets of links can transmit simultaneously. For such a setting, one will require to have a hierarchical scheduler as in [20] and the bound for the synchronization buffers might be much larger than the one obtained in this paper. Another interesting research problem is to investigate whether the maximum weighted matching algorithm [26], another universal stabilization algorithm, performs well in the setting of network coding. In the maximum weighted matching algorithm, queue lengths are often used as the weights. Thus, large queues tend to be served first. How one incorporates *synchronization* into weights requires further study.

References

[1] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network Information Flow, " *IEEE Transactions on Information Theory,* vol. 46, no. 4, Jul. 2000, pp. 1204-1216.

[2] F. Baccelli, W. A. Massey, and D. Towsley, "Acyclic Fork-Join Queueing Networks, " *Journal of the ACM,* vol. 36, no. 3, 1989, pp. 615-642.

[3] C. -S. Chang, *Performance Guarantees in Communication Networks,* London: Springer-Verlag, 2000.

[4] C. -S. Chang, Y. -H. Hsu, J. Cheng, and D. -S. Lee, "A Dynamic Frame Sizing Algorithm for CICQ Switches with 100% Throughput, " *IEEE INFOCOM 2009.*

[5] P. A. Chou, Y. Wu, and K. Jain, "Practical network coding, " *Proc. 41st Annual Allerton Conference on Communication, Control, and Computing, Oct. 2003.*

[6] P. A. Chou, and Y. Wu, "Network Coding for the Internet and Wireless Networks, " *IEEE Signal Processing Magazine,* vol. 24, no. 5, Sep. 2007, pp. 77-85.

[7] R. Cogill, B. Shrader, and A. Ephremides, "Stable Throughput for Multicast With Random Linear Coding," *IEEE Transactions on Information Theory,* vol. 57, no. 1, Jan. 2011, pp. 266-280.

[8] P. Giaccone, E. Leonardi, and D. Shah, "Throughput Region of Finite-buffered Networks, " *IEEE Transaction on Parallel and Distributed Systems,* vol. 18, no. 2, Feb. 2007, pp. 251-263.

[9] S. -M. He, S. -T. Sun, H. -T. Guan, Q. Zheng, Y. -J. Zhao, and W. Gao, "On Guaranteed Smooth Switching for Buffered Crossbar Switches, " *IEEE/ACM Transactions on Networking,* vol. 16, no. 3, Jun. 2008, pp. 718-731.

[10] T. Ho, R. Koetter, M. Medard, D. R. Karger, and M. Effros, "The Benefits of Coding over Routing in a Randomized Setting, " *IEEE International Symposium on Information Theory, Jun. 2003.*

[11] T. Ho, and D. S. Lun, *Network Coding: An Introduction,* Cambridge University Press, 2008.

[12] T. Ho, and H. Viswanathan, "Dynamic Algorithms for Multicast With Intra-Session Network Coding, " *IEEE Transaction on Information Theory,* vol. 55, no. 2, Feb. 2009, pp. 797-815.

[13] L. Huang, and M. J. Neely, "Utility Optimal Scheduling in Processing Networks, " Performance Evaluation, vol. 68, 2011, pp. 1002-1021.

[14] S. Jaggi, P. Sanders, P. A. Chou, M. Effros, S. Egner, K. Jain, and L. M. G. M. Tolhuizen, "Polynomial Time Algorithms for Multicast Network Code Construction, " *IEEE Transaction on Information Theory,* vol. 51, no. 6, Jun. 2005, pp. 1973-1982.

[15] L. Jiang, and J. Walrand, "Stable and Utility-Maximizing Scheduling for Stochastic Processing Networks, " *Forty-Seventh Annual Allerton Conference on Communication, Control, and Computing,* 2009.

[16] R. Koetter, and M. Medard, "An Algebraic Approach to Network Coding, ", *IEEE/ACM Transactions on Networking,* vol. 11, no. 5, Oct. 2003, pp. 782-795.

[17] S.-Y. R. Li. *Algebraic Switching Theory and Broadband Applications*. Academic Press, 2001.

[18] S.-Y. R. Li, R. W. Yeung, and N. Cai, "Linear Network Coding, " *IEEE Transaction on Information Theory,* vol. 49, no. 2, Feb. 2003, pp. 371-381.

[19] C. -M. Lien, and C. -S. Chang, "Generalized Dynamic Frame Sizing Algorithm for Finite-Internal-Buffered Networks, " *IEEE Communication Letters,* vol. 13, no. 9, Sep. 2009, pp. 714-716.

[20] C.-M. Lien, C.-S. Chang, J. Cheng and D.-S. Lee, "Maximizing Throughput in Wireless Networks with Finite Internal Buffers," *Proceedings of IEEE INFOCOM 2011.*

[21] M. A. Marsan, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri, "Multicast Traffic in Input-Queued Switches: Optimal Scheduling and Maximum Throughput, " *IEEE/ACM Transactions on Networking,* vol. 11, no. 3, Jun. 2003, pp. 465-477.

[22] R. Nelson, and A. N. Tantawi, "Approximate Analysis of Folk/Join Synchronization in Parallel Queues, " *IEEE Transactions on Computers,* vol. 37, no. 6, Jun. 1988, pp. 739-743.

[23] P. Parag, and J. -F. Chamberland, "Queueing Analysis of a Butterfly Network for Comparing Network Coding to Classical Routing," *IEEE Transactions on Information Theory,* vol. 56, no. 4, Apr. 2010, pp. 1890-1908.
[24] K. Prasad, and B. Sundar Rajan, "Single-Generation Network Coding for Networks with Delay, " *IEEE ICC 2010.*
[25] Sheldon M. Ross, *Stochastic Processes,* John Wiley & Sons, Inc., 1996.
[26] L. Tassiulas and A. Ephremides, "Stability Properties of Constrained Queueing Systems and Scheduling Policies for Maximum Throughput in Multihop Radio Networks," *IEEE Transactions on Automatic Control*, vol. 31, no. 12, pp. 1936–1948, 1992
[27] D. Traskov, M. Medard, P. Sadeghi, and R. Koetter, "Joint Scheduling and Instantaneously Decodable Network Coding, " *IEEE GLOBECOM 2009.*
[28] X. Wu, C. Zhao, and X. You, "Generation-Based Network Coding over Networks with Delay, " *2008 IFIP International Conference on Network and Parallel Computing.*
[29] R. W. Yeung, *Information Theory and Network Coding,* Springer, August 2008.

PLACE PHOTO HERE

Ching-Min Lien (S'09-M'11) received the B.S., M.E. degrees, both in Electrical Engineering, and Ph.D. degree in Communications Engineering from the National Tsing Hua University, Hsinchu, Taiwan, R.O.C., in 1999, 2001 and 2011, respectively. Since 1993, he has been with the Institute of Communications Engineering at National Tsing Hua University, Taiwan, R.O.C., where he is a Postdoctoral Research Fellow. He has been elected as an honorary member of the Phi Tau Phi Honor Society of the Republic of China in 2011. His current research interests are concerned with high speed switching, communication network theory, distributed resource scheduling, and large complex networks.

PLACE PHOTO HERE

Cheng-Shang Chang (S'85-M'86-M'89-SM'93-F'04) received the B.S. degree from the National Taiwan University, Taipei, Taiwan, in 1983, and the M.S. and Ph.D. degrees from Columbia University, New York, NY, in 1986 and 1989, respectively, all in Electrical Engineering. From 1989 to 1993, he was employed as a Research Staff Member at the IBM Thomas J. Watson Research Center, Yorktown Heights, N.Y. Since 1993, he has been with the Department of Electrical Engineering at National Tsing Hua University, Taiwan, R.O.C., where he is a Professor. His current research interests are concerned with network science, high speed switching, communication network theory, and mathematical modeling of the Internet. Dr. Chang received an IBM Outstanding Innovation Award in 1992, an IBM Faculty Partnership Award in 2001, and Outstanding Research Awards from the National Science Council, Taiwan, in 1998, 2000 and 2002, respectively. He also received Outstanding Teaching Awards from both the college of EECS and the university itself in 2003. He was appointed as the first Y. Z. Hsu Scientific Chair Professor in 2002 and elected to an IEEE Fellow in 2004. Dr. Chang received the Academic Award from the Ministry of Education in 2011. He is the author of the book "Performance Guarantees in Communication Networks" and the coauthor of the book "Principles, Architectures and Mathematical Theory of High Performance Packet Switches." He served as an editor for Operations Research from 1992 to 1999 and an editor for IEEE/ACM Transactions on Networking from 2007 to 2009. Dr. Chang is a member of IFIP Working Group 7.3.

PLACE PHOTO HERE

Duan-Shin Lee (S'89-M'90-SM'98) received the B.S. degree from National Tsing Hua University, Taiwan, in 1983, and the MS and Ph.D. degrees from Columbia University, New York, in 1987 and 1990, all in electrical engineering. He worked as a research staff member at the C&C Research Laboratory of NEC USA, Inc. in Princeton, New Jersey from 1990 to 1998. He joined the Department of Computer Science of National Tsing Hua University in Hsinchu, Taiwan, in 1998. Since August 2003, he has been a professor. He received a best paper award from the Y.Z. Hsu Foundation in 2006. His research interests are network science, switch and router design, and performance analysis of communication networks. He is a senior IEEE member.