A Low-Complexity Congestion Control and Scheduling Algorithm for Multihop Wireless Networks With Order-Optimal Per-Flow Delay

Po-Kai Huang, *Member, IEEE*, Xiaojun Lin, *Member, IEEE*, and Chih-Chun Wang, *Member, IEEE*

***Abstract*—Quantifying the end-to-end delay performance in multihop wireless networks is a well-known challenging problem. In this paper, we propose a new joint congestion control and scheduling algorithm for multihop wireless networks with fixed-route flows operated under a general interference model with interference degree $\mathcal{K}$. Our proposed algorithm not only achieves a provable throughput guarantee (which is close to at least $1/\mathcal{K}$ of the system capacity region), but also leads to explicit upper bounds on the end-to-end delay of every flow. Our end-to-end delay and throughput bounds are in simple and closed forms, and they explicitly quantify the tradeoff between throughput and delay of every flow. Furthermore, the per-flow end-to-end delay bound increases linearly with the number of hops that the flow passes through, which is order-optimal with respect to the number of hops. Unlike traditional solutions based on the back-pressure algorithm, our proposed algorithm combines window-based flow control with a new rate-based distributed scheduling algorithm. A key contribution of our work is to use a novel stochastic dominance approach to bound the corresponding per-flow throughput and delay, which otherwise are often intractable in these types of systems. Our proposed algorithm is fully distributed and requires a low per-node complexity that does not increase with the network size. Hence, it can be easily implemented in practice.**

***Index Terms*—Low-complexity and distributed algorithms, order-optimal delay bound, rate-based scheduling algorithms, supermodular ordering, window-based flow control.**

I. Introduction

THE JOINT congestion control and scheduling problem in multihop wireless networks has been extensively studied in the literature [1], [2]. Often, each user is associated with a nondecreasing and concave utility function of its rate, and a cross-layer utility maximization problem is formulated to maximize the total system utility subject to the constraint that the rate vector can be supported by some scheduling algorithm. One optimal solution to this problem is known to be the max-weight back-pressure scheduling algorithm combined with a congestion control component at the source [1], [2]. Furthermore, significant progress has been made in designing distributed scheduling algorithms with provable throughput and lower complexity than the back-pressure algorithm [3]–[8]. However, most of the existing works on joint congestion control and scheduling have only considered the throughput performance metric and have not accounted for delay performance issues. Although for flows with congestion control (e.g., file transfer) the throughput is often the most critical performance metric, packet delay is important as well because practical congestion control protocols need to set retransmission timeout values based on the packet delay, and such parameters could significantly impact the speed of recovery when packet loss occurs. Packet delay is also important for multimedia traffic, some of which have been carried on congestion-controlled sessions.

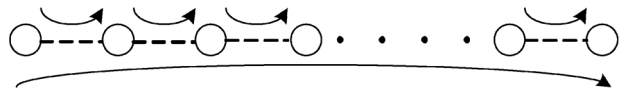


Fig. 1. Wireless network with linear topology.

There are two major issues on the delay-performance of the back-pressure algorithm. First, for long flows, the end-to-end delay may grow quadratically with the number of hops. The reason can be best explained by the following example [9]. Consider a long flow traverses a fixed route with H hops. For each link that the long flow traverses, there is a competing short flow as shown in Fig. 1. Under the back-pressure algorithm, if a link schedules the long flow, the queue difference of the long flow must be larger than the queue length q of the competing short flow. Therefore, when the joint congestion and scheduling algorithm converges, the queue length of the long flow at each hop must be around $Hq, (H-1)q, \cdots, q$, and the total end-to-end backlog is of order $\mathcal{O}(H^2)$. By Little's law, the end-to-end delay will also be of order $\mathcal{O}(H^2)$. Note that a packet needs at least H time-slots to reach the destination. Hence, the optimal order should have been $\mathcal{O}(H)$. This implies that the back-pressure algorithm may have significantly larger end-to-end delay for long flows. Second, under the back-pressure algorithm, it is difficult to control the end-to-end delay of each flow. The main parameter to tune a joint congestion control and scheduling algorithm based on the back-pressure algorithm is the step size in the queue update. A larger step size may lead to smaller queue length. However, a smaller step size is needed to ensure that the

Manuscript received October 21, 2011; revised March 23, 2012 and May 21, 2012; accepted May 22, 2012; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor L. Andrew. Date of publication October 02, 2012; date of current version April 12, 2013. This work was supported in part by the NSF under Grants CNS-0721477, CNS-0721484, CNS-0643145, CNS-0813000, CNS-0905331, and CCF-0845968 and the Purdue Research Foundation. An earlier version of this paper appeared in the 30th IEEE International Conference on Computer Communications (INFOCOM), Shanghai, China, April 10–15, 2011.

The authors are with the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907 USA (e-mail: huang113@purdue.edu; linx@purdue.edu; chihw@purdue.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/TNET.2012.2213343

joint congestion control and scheduling algorithm converges to close-to-optimal system throughput. Although one may use the step sizes to tune the throughput–delay tradeoff, a change of the step size on one node will likely affect all flows passing through the node. Hence, it is difficult to tune the throughput–delay tradeoff on a per-flow basis.

In this paper, we will provide a new class of joint congestion control and scheduling algorithms[1] that can achieve both provable throughput and provable per-flow delay. Consider m flows in a multihop network operating under a general interference model with the interference degree $\mathcal{K}$ (the notion of the interference degree will be given in Section II), and each flow m is given a fixed route with H_m hops. Our algorithm consists of three main components: window-based flow control, virtual-rate computation, and scheduling. The main ideas of our algorithm to improve the end-to-end delay are as follows. First, by using window-based flow control, we can tightly control the number of packets inside the network. Second, by using a *rate-based* scheduling algorithm with the computed virtual rate as input to schedule packets, we do not need to wait for the packets to accumulate before making scheduling decisions. However, the key difficulty in analyzing the end-to-end throughput and delay under this algorithm is that *the services at different links are correlated*. Hence, a Markov chain analysis will no longer provide a closed-form solution. We employ a novel stochastic dominance technique to circumvent this difficulty and derive closed-form bounds on the per-flow throughput and delay. Specifically, for any $\epsilon, \epsilon_m \in (0,1)$, by appropriately choosing the number of backoff mini-slots for the scheduling algorithm and the window size of flow m, our algorithm can guarantee that each flow m will achieve a throughput no less than $r_m(1-\epsilon)(1-\epsilon_m)$, where the total utility of the virtual rate allocation vector $\vec{r} = [r_m]$ is no smaller than the total utility of any rate vector within $\Omega/\mathcal{K}$, where Ω is the capacity region. Furthermore, the end-to-end expected delay of flow m can be upper-bounded by $H_m/(r_m(1-\epsilon)\epsilon_m)$. Therefore, with a reasonable choice of the parameters of the algorithm, our scheme can utilize a provable fraction of the total system utility with per-flow expected delay that increases linearly with the number of hops. Since a flow-m packet requires at least H_m time-slots to reach the destination, the order of the per-flow delay upper bound is optimal with respect to the number of hops. Our proposed algorithm is fully distributed and can be easily implemented in practice. Furthermore, the delay–throughput tradeoff of each flow can be individually controlled. To the best of our knowledge, this is the first fully distributed cross-layer control solution that can both guarantee order-optimal per-flow delay and a minimum throughput utilization close to $1/\mathcal{K}$ of the system capacity under a general interference model.

Recently, there have been a number of papers that quantify the delay performance of wireless networks with or without congestion control [9], [11]–[20]. In [9] and [11], the authors propose methods to reduce the delay of the back-pressure algorithm. The algorithm proposed in [9] is a shadow back-pressure algorithm, which maintains a single first-in–first-out (FIFO) queue at each link and uses multiple shadow queues to schedule the transmissions. This method decouples the control information from the real queues and hence reduces the delay. In our simulation, this algorithm seems to achieve linear delay after the algorithm converges. However, at the transient period, the real queues will still follow the shadow queues, which leads to a large queue backlog (see Fig. 4(c) in this paper and figures in [9]). In [11], the authors propose another mechanism, FQLA, to decouple the control signal from the real queues by injecting place holder bits into the queues. However, the FQLA algorithm requires an initial period to determine the place holder bits for each queue. In this initial period, normal BP algorithm is used. Hence, the delay performance during this transient phase would be comparable to that of the BP algorithm at best. At the end of this initial period, a significant fraction of packets (that correspond to the placeholder bits) must be dropped. In contrast, our proposed algorithm does not drop packets that are admitted, and the delay performance in the transient phase does not deviate significantly from the value after convergence. Finally, neither [9] nor [11] provides closed-form bounds on the per-flow end-to-end delay.

Our result is also different from other works in providing a *per-flow* end-to-end delay bound. First, [12] and [13] only prove delay bounds for single-hop flows rather than multihop flows. Second, [14]–[16] consider the delay among all the flows rather than the per-flow delay. Similarly, the results in [17] can be used to construct a bound on the delay averaged over all flows. However, it is still not a per-flow delay bound. In contrast, our per-flow delay bound scales with the number of hops of the flow itself; hence, it is usually much tighter. A per-flow delay bound is provided in [18], but the bound scales with the size of the network. Finally, a single-flow end-to-end delay analysis is given in [19] based on an approximation of the departure process for each hop. However, it is unclear how to extend the analysis to multiple flows.

Our result is perhaps most comparable to that in [20], where the authors provide a per-flow delay bound that scales with the number of hops without considering congestion control. However, the algorithm in [20] has a factor of 5 loss of throughput even under the one-hop interference constraint, where the interference degree is 2. If the network operates under the two-hop interference constraint, the throughput loss factor is $4\Delta^2$, where Δ is the maximum degree of the vertices. In contrast, our algorithm only has a factor of 2 loss of throughput under the one-hop interference constraint. Under the two-hop interference constraint, the throughput loss factor of our algorithm is 2Δ. Moreover, the algorithm in [20] is considerably more complicated, e.g., the per-node complexity is $\mathcal{O}(N)$, where N is the number of nodes. In contrast, our algorithm only requires $\mathcal{O}(1)$ complexity per node.

Our contributions can be summarized as follows.

- We provide a new joint congestion control and scheduling algorithm that can guarantee a minimum throughput utilization close to $1/\mathcal{K}$ of the system capacity, i.e., at most a factor of $\mathcal{K}$ loss of throughput, and guarantee an upper bound on the per-flow end-to-end expected delay that increases linearly with the number of hops.
- The congestion control algorithm is based on window-based flow control, which deterministically bounds

[1]A related delay bound can be shown for the scheduling algorithm without congestion control [10]. However, the delay analysis for joint congestion control and scheduling in this paper is more difficult due to the closed-loop feedback.

each flow's end-to-end backlog within the network and prevents buffer overflows. Furthermore, each flow's throughput–delay tradeoff can be individually controlled.
- Our algorithm is fully distributed and can be easily implemented in practice with a low per-node complexity that does not increase with the network size.
- We use a novel stochastic dominance method to analyze the end-to-end delay. This method is new and could be of independent value.

The remainder of this paper is organized as follows. The system model is presented in Section II. In Section III, we propose the joint congestion control and scheduling algorithm and present the main analytical results on per-flow throughput and delay. Section IV is dedicated to the proof of a key proposition by a novel stochastic dominance method. Implementation issues are discussed in Section V, and simulation results are reported in Section VI. Then, we conclude.

II. System Model

We model a wireless network by a graph $G = (V, E)$, where V is the set of nodes, and E is the set of links. We use the notation $|E|$ to denote the total number of links. Each link $\ell \in E$ consists of a transmitter node $b(\ell)$ and a receiver node $d(\ell)$. Two nodes are one-hop neighbors if they are the endpoints of a common link. Two links are one-hop neighbors if they share a common node. Two links are two-hop neighbors if they have a common one-hop neighboring link.

We assume a time-slotted wireless system, where packet transmissions occur within time-slots of unit length. Let c_ℓ denote the capacity of link ℓ, which represents the number of packets that link ℓ can transmit in one time-slot. We assume that c_ℓ is an integer for all link ℓ. We say two links interfere with each other if they cannot transmit data at the same time-slot. The set of all links that interfere with link ℓ is called the interference set of link ℓ and is denoted as $\mathcal{E}_\ell$. We adopt the convention that $\ell \in \mathcal{E}_\ell$, i.e., $\mathcal{E}_\ell = \{\ell\} \cup \{\ell' : \ell' \in E \text{ and } \ell' \text{ interferes with } \ell\}$. Assume that the interference relationship is symmetric, i.e., if $k \in \mathcal{E}_\ell$, then $\ell \in \mathcal{E}_k$. The interference degree of a link ℓ is the maximum number of links within its interference range that can be activated simultaneously without interfering with each other. The interference degree $\mathcal{K}$ of a network is the maximum interference degree over all links [6]–[8]. This interference model is very general, and it includes the one-hop interference constraint, i.e., a link will only interfere with all its one-hop neighboring links, and the two-hop interference constraint. Note that one-hop interference constraint has been used to model FH-CDMA systems in [5], [21], and [22], and the two-hop interference constraint has been used to model the interference relationship in IEEE 802.11 Distributed Coordination Function (DCF) [6]. Under one-hop interference constraint, the interference degree is at most 2. Under the two-hop interference constraint, define the in-degree and out-degree of a node v as the number of links in E that ends in v and originates from v, respectively. Define the directed degree of a link ℓ as the sum of the out-degree of transmitting node and in-degree of the receiving node. The maximum directed link degree in G, Δ_G, is the maximum directed degree of any link in E. It has been shown that the value of $\mathcal{K}$ can be upper-bounded by $\max(\Delta_G - 2, 1)$ [6]. Since $\Delta_G \leq 2\Delta$, where Δ is the maximum degree of the vertices, $\mathcal{K}$ is at most 2Δ.

There are M flows in the system, and each flow is associated with a source node, a destination node, and a fixed route between them. The routes are given by the matrix $[L_m^\ell]$, where $L_m^\ell = 1$ if flow m passes through link ℓ, and $L_m^\ell = 0$ otherwise. We assume that each flow always has packets to transmit. The congestion control algorithm will then determine the rate with which packets are injected into the network [1]. Each flow is associated with a utility function $U_m(R_m)$ [23], which reflects the "satisfactory level" of user m with injection rate R_m. We assume that $U_m(\cdot)$ is strictly concave, nondecreasing, and continuously differentiable. The capacity region Ω of a wireless network is the set of all rate vectors $\vec{R} = [R_m]$ such that there exists a network control policy to stabilize the network. We then model the joint congestion control and scheduling problem as

$$\max_{R_m \geq 0} \sum_{m=1}^{M} U_m(R_m), \qquad \vec{R} \in \Omega. \tag{1}$$

The exact capacity region Ω is often difficult to characterize. On the other hand, it is well known that $\Psi_0/\mathcal{K} \subseteq \Omega \subseteq \Psi_0$ [8], where

$$\Psi_0 = \left\{ \vec{r} \middle| \sum_{\ell \in \mathcal{E}_k} \sum_{m=1}^{M} \frac{L_m^\ell r_m}{c_\ell} \leq \mathcal{K}, \qquad \text{for all links } k \right\}. \tag{2}$$

Note that (2) simply states that, for each link ℓ, the total normalized load in an interference set must be no larger than $\mathcal{K}$. This is because at most $\mathcal{K}$ links can be scheduled in any interfere set at a time-slot. In Section III, we will describe how we utilize the relationship between Ω and Ψ_0 to approximately solve (1). Since we assume infinite backlog, the delay of a packet is computed from the time it is injected to the network to the time it reaches the destination. We are interested in both the throughput and the expected end-to-end delay of each flow.

III. Joint Congestion Control and Scheduling Algorithm

As we discussed in Section I, there are many approaches available in the literature to solve (1), and most of them do not consider delay performance. A typical optimal solution can be obtained by a duality approach that results into the back-pressure algorithm and a congestion-control component at the source node [1], [2]. Furthermore, a considerable amount of effort has focused on developing low-complexity and distributed scheduling algorithms that can replace the centralized back-pressure algorithm and yet still achieve provably good throughput performance [3]–[8]. Like the back-pressure algorithm, these low-complexity scheduling algorithms are usually also queue-length-based. The drawback of these approaches, however, is that the end-to-end delay of the resulting queue-length-based scheduling algorithm is very difficult to quantify, and as we described in Section I, under certain cases the back-pressure algorithm can have poor delay performance [9], [24]. In this paper, we will use a window-based flow control algorithm and a rate-based scheduling algorithm that are very different from the back-pressure algorithm. Our solution strategy is to first approximately solve (1) and compute

the decision vector $\vec{r} = [r_m]$. However, the decision variables r_m are *not* directly used as the rates to inject flow-m packets. For this reason, we refer to these variables r_m as "virtual rates." We will use these virtual rates as the control variables in a new class of rate-based scheduling algorithms. The actual end-to-end throughput under our algorithm will be denoted as R_m. As readers will see, for each flow, this new joint congestion control and scheduling algorithm will guarantee both provable throughput (close to r_m) and provably low delay. Also, they are fully distributed and easy to implement in real systems.

A. Virtual-Rate Computation

We first briefly describe how to approximately solve (1). Since the true capacity region Ω is of a complex form, instead of solving (1) directly, we solve the following optimization problem: (we will make precise the relationship between optimization problems (1) and (3) in Section III-D)

$$\max_{r_m \geq 0} \sum_{m=1}^{M} U_m(r_m), \qquad \vec{r} \in \frac{\Psi_0}{\mathcal{K}}. \tag{3}$$

Note that the optimization problem (3) is very similar to the standard convex-optimization problem in wireline network with linear constraints [25], [26]. Therefore, it is easy to apply the approaches in [25] and [26] to (3). Instead of elaborating on all the possible approaches to solve problem (3), we only present one well-known distributed solution. Specifically, associate a Lagrange multiplier (the dual variable) $\lambda_k \geq 0$ to each constraint in (3). The objective function of the dual problem of (3) becomes

$$D(\vec{\lambda}) := \max_{r_m \geq 0} \sum_{m=1}^{M} U_m(r_m) - \sum_{k=1}^{|E|} \lambda_k \left(\sum_{\ell \in \mathcal{E}_k} \sum_{m=1}^{M} \frac{L_m^\ell r_m}{c_\ell} - 1 \right).$$

We can then use the following gradient algorithm to minimize $D(\vec{\lambda})$ and compute the optimal virtual-rates.

Virtual-Rate Computation Algorithm: At each time t:

1) The source node of flow m updates r_m by equation

$$r_m(t) = U_m'^{-1} \left(\sum_{k=1}^{|E|} \lambda_k(t) \sum_{\ell \in \mathcal{E}_k} \frac{L_m^\ell}{c_\ell} \right).$$

2) Each link updates the dual variables by equation

$$\lambda_k(t+1) = \left[\lambda_k(t) + \gamma_k \left(\sum_{\ell \in \mathcal{E}_k} \sum_{m=1}^{M} \frac{L_m^\ell r_m}{c_\ell} - 1 \right) \right]^+$$

where $\gamma_k > 0$ is the step size, and $[\cdot]^+$ denotes the projection to $[0, \infty)$.

Using similar techniques as [25], one can show that as long as γ_k are sufficiently small, the above algorithm will converge to the optimal solution of (3). Note that as in [25], this algorithm requires passing λ_k and r_m among nodes in the network. We will give a simple protocol to exchange such information in Section V. As we emphasized earlier, the variables r_m are "virtual rates," and they are not directly used to inject flow-m packets under our proposed algorithm. We choose not to directly use the virtual rates as the real injection rates due to the following reasons. First, optimization problems (1) and (3) are formulated as if the rates are immediately passed to all links at the same time. In reality, a packet must traverse the links in a hop-by-hop fashion. In order to control the end-to-end delay, an additional flow-control algorithm is needed to regulate this hop-by-hop packet flow. Second, the low-complexity virtual-rate computation algorithm did not produce the schedule for link transmission. We still need a scheduling algorithm to compute the schedule that can support the virtual rate vector $\vec{r} = [r_m]$.

Readers who are familiar with the literature will realize that the back-pressure algorithm can again be used to answer the above flow control and scheduling questions. However, we would then return to our starting point that the end-to-end delay of the back-pressure algorithm is difficult to quantify and may be poor [9], [24]. Hence, in the sequel, we will use very different scheduling and flow-control components, for which we can quantify both the throughput and the end-to-end delay on a per-flow basis.

B. Scheduling Algorithm

We now present the scheduling algorithm, which is a modification of the low-complexity distributed scheduling algorithm in [8]. Each time-slot consists of an initial scheduling slot, which is further divided into F mini-slots. The links that are to be scheduled are selected in the scheduling slot, and the selected links transmit their packets in the rest of the time-slot. Let $a_\ell(t) = \sum_{m=1}^{M} L_m^\ell r_m(t)/c_\ell$, which is the normalized sum of the virtual rate over link ℓ, and let $x_\ell(t) = \max_{i \in \mathcal{E}_\ell} (\sum_{k \in \mathcal{E}_i} a_k(t))$.

Rate-Based Scheduling Algorithm: At each time-slot t:

1) Each link ℓ first computes $P_\ell = a_\ell(t) \log F / x_\ell(t)$.
2) In the scheduling slot, each link then randomly picks a backoff mini-slot (B) with distribution: $P\{B = F+1\} = e^{-P_\ell}$ and $P\{B = f\} = e^{-P_\ell((f-1)/F)} - e^{-P_\ell(f/F)}$, $f = 1, \cdots, F$. If $F+1$ is picked by link ℓ, it will not attempt to transmit in this time-slot.
3) When the backoff timer for a link expires in the scheduling slot, it begins transmission unless it has already heard a transmission from one of its interfering links. If two or more links that interfere begin transmissions simultaneously, a collision occurs, and both transmissions fail.
4) When a link begins transmission, it will randomly choose a passing flow m to serve with probability $r_m(t)/(a_\ell(t)c_\ell)$.

Note that this scheduling algorithm only uses virtual rates to compute P_ℓ, which is different from the queue-length-based algorithm studied in [8]. For simplicity, our performance analysis will be based on this scheduling algorithm. On the other hand, note that this algorithm can be easily improved by letting each link attempt only if it has packets to transmit, and if it starts transmission, it will randomly serve a flow m with positive backlog (i.e., $Q_{m\ell}(t) > 0$) with probability $r_m(t) / \sum_{\{m: Q_{m\ell}(t) > 0\}} L_m^\ell r_m(t)$, where $Q_{m\ell}(t)$ is the number of flow-m packets at link ℓ at time t. It is easy to see that this improved version will lead to higher throughput. Hence, the bound we derive in Section IV also holds for this improved version.

C. Window-Based Flow Control Algorithm

Now, we describe the congestion control component. Our approach is to use the window-based flow control. For each flow, we maintain a window W_m at the source node, and we only inject new packets to the queue at the source node when the total number of packets for this flow inside the network is smaller than the window size. This can be achieved by letting the destination node send an acknowledgement (ACK) back to the source node whenever it receives a packet. There are two advantages for this approach. First, for each flow, we can tightly control the maximum number of packets in each intermediate node along the route. This will prevent buffer overflows, which is an important issue as addressed in [17]. Second, as we will show in Section IV, each flow's tradeoff between throughput and delay can be individually controlled by the window size. Note that when we present the analysis in Section IV, we assume that there is a feedback channel from the destination node to the source node at each time-slot. Through this feedback channel, the destination node can send the ACK to the source node, and the source node can then decide if it is possible to inject another packet at the next time-slot. In reality, in order to reach the source node, each ACK will also go through the entire route in a hop-by-hop fashion in the reverse direction. In Section V, we will discuss how this can be achieved by piggybacking the ACK after each packet transmission. As readers will see, this method can be analyzed with the same approach presented in Section IV, and this extra ACK delay does not change the delay order of our result.

D. Performance Analysis

In this section, we will present the main steps of the analysis and the bounds on the throughput and delay of the above proposed scheme. We first present a relationship between optimization problems (1) and (3). Let $[r_m^*]$ be the optimal solution of (3), and let $[r_m'^*]$ be the optimal solution of the following optimization problem:

$$\max_{r_m \geq 0} \sum_{m=1}^{M} U_m(r_m), \qquad \vec{r} \in \frac{\Omega}{\mathcal{K}}. \tag{4}$$

Lemma 1: $\sum_{m=1}^{M} U_m(r_m^*) \geq \sum_{m=1}^{M} U_m(r_m'^*)$.

Proof: Since r_m^* is the optimal solution of (3), the total utility of $[r_m^*]$ will be larger than any rate vector in $\Psi_0/\mathcal{K}$. Also, by $\Omega \subseteq \Psi_0$, we have that $\Omega/\mathcal{K} \subseteq \Psi_0/\mathcal{K}$. Thus, $[r_m'^*]$ is a rate vector in $\Psi_0/\mathcal{K}$, and $\sum_{m=1}^{M} U_m(r_m^*) \geq \sum_{m=1}^{M} U_m(r_m'^*)$. ■

In other words, if each flow achieves a throughput equal to the optimal virtual rate r_m^*, then the total system utility will be no less than the maximum utility within $\Omega/\mathcal{K}$. Furthermore, we can show the following property of the scheduling algorithm.

Lemma 2: If flow m passes through link ℓ, the probability that link ℓ will schedule flow m at time-slot t is no smaller than $r_m(t)(1-\epsilon)/(x_\ell(t)c_\ell)$, where $\epsilon = (\log F + 1)/F$.

Proof: The proof is omitted for brevity. Details can be found in the technical report [27]. ■

Note that this lemma holds at all time-slot t even before the virtual-rate computation algorithm converges. Next, we focus on the system performance after the virtual-rate computation algorithm converges.[2] After the virtual-rate converges, the value of $r_m(t)$ will be equal to the optimal solution r_m^*. Furthermore, we have from the constraints of optimization problem (3) that $x_\ell(t) \leq 1$. It then follows from Lemma 2 that every link ℓ along the path of flow m will serve flow m with probability no smaller than $r_m^*(1-\epsilon)/c_\ell$, independent across time-slots. Intuitively, if the window size of flow m is large, the end-to-end throughput of flow m, i.e., R_m, will be at least $r_m^*(1-\epsilon)$. However, a large window size will also lead to a large end-to-end packet delay. If we reduce the window size, although the delay will decrease, the throughput of flow m will also decrease. Clearly, the key is then to analyze the throughput and delay as a function of the window size. The following proposition, which will be proved in Section IV, is the key result of the paper. Note that this analysis is difficult because Lemma 2 only provides a lower-bounded marginal probability for services. Furthermore, the exact statistics of the correlation among links is hard to characterize because of the interference constraint.

Let $c_{\max}^m$ be the maximum capacity of any link along the route of flow m. For simplicity, in the following derivations, we assume that the window-size of flow m is chosen as a multiple of $c_{\max}^m$, i.e., $W_m = c_{\max}^m w$, where w is a positive integer.

Proposition 1: After the virtual-rate computation algorithm converges, for each flow m, our congestion control and scheduling algorithm can achieve average throughput no smaller than $r_m^*(1-\epsilon)w/(w+H_m-1)$, where ϵ is chosen as in Lemma 2. Moreover, the average delay is upper-bounded by $c_{\max}^m(w + H_m - 1)/(r_m^*(1-\epsilon))$.

This proposition has the following two implications. First, for any $\epsilon_m \in (0,1)$, let w be the smallest positive integer such that $w > (H_m-1)(1-\epsilon_m)/\epsilon_m$. This implies that $w/(w + H_m - 1) > (1-\epsilon_m)$. It then follows from Proposition 1 that the average throughput R_m will be lower-bounded by $r_m^*(1-\epsilon)(1-\epsilon_m)$, which can be arbitrary close to r_m^*. Note that by Lemma 1, the total utility of the rate vector $\vec{r}^* = [r_m^*]$ is no smaller than the total utility of any rate vector within $\Omega/\mathcal{K}$. Hence, our joint congestion control and scheduling algorithm guarantees a minimum throughput utilization close to $1/\mathcal{K}$ of the system capacity. Second, since w is the smallest positive integer such that $w > (H_m-1)(1-\epsilon_m)/\epsilon_m$, we have that $w \leq (H_m-1)(1-\epsilon_m)/\epsilon_m + 1$. Thus

$$\frac{w+H_m-1}{r_m^*(1-\epsilon)} \leq \frac{H_m+\epsilon_m-1}{r_m^*(1-\epsilon)\epsilon_m} < \frac{H_m}{r_m^*(1-\epsilon)\epsilon_m}.$$

It then follows from Proposition 1 that the delay will be upper-bounded by $c_{\max}^m H_m/(r_m^*(1-\epsilon)\epsilon_m)$. As discussed in Section I, this implies that our per-flow delay upper bound is order optimal with respect to the number of hops.

Note that the assumption that the window size is a multiple of $c_{\max}^m$ is needed only for proving the explicit expression for end-to-end throughput and delay. In practice, any integer value of window size can be used for the algorithm. Our simulation results show that, by choosing the window size to be $(c_{\min}^m +$

[2]We note that a comparable bound on the probability of scheduling flow m on link ℓ can also be obtained by assuming that $r_m(t)$ is within some small neighborhood of r_m^*. For ease of exposition, we do not pursue this direction further in this paper.

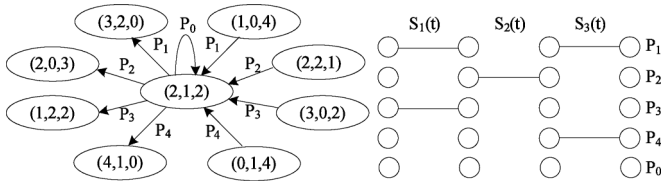


Fig. 2. *(left)* Incoming and outgoing transitions to and from state (2, 1, 2). *(right)* Distribution of the random schedule vector $\vec{S}(t)$. The capacities of links 1, 2, and 3 are 1, 1, and 2, respectively.

$c^m_{\max})H_m$,[3] where $c^m_{\min}$ is the minimum capacity of any link along the route of flow m, the algorithm can achieve both reasonable throughput and linear order delay.

IV. Proof of Proposition 1

Assume that the virtual-rate computation algorithm has converged at time t. Thus, $r_m(t) = r^*_m$ for the following time-slots. This implies that, for a particular flow, its service at every hop is identically and independently distributed (i.i.d.) across time. This observation allows us to isolate flow m out of the network and view the flow-m as passing through a virtual tandem network of H_m queues. Furthermore, for ease of presentation, we assume that there is a feedback channel from the destination node to the source node for the window-based flow control at each time-slot as discussed in Section III-C. (This assumption will be removed in Section V.) Now, we focus on a particular flow m. The analysis for other flows is the same. To ease the notation, we drop the index m from the notations W_m, H_m, and $c^m_{\max}$. We can then model this flow as an H-hop closed tandem network. Label the link along the route from 1 to H, where 1 is the link closest to the source node. Let the capacity of the ith link be c_i. By the discussion after Lemma 2, we know that

$$\mu_\ell c_\ell \geq \mu \triangleq r^*_m(1-\epsilon) \tag{5}$$

where μ_ℓ is the probability that link ℓ will serve a flow-m packet. Since we use the window-based flow control, and flow m always has packets to transmit, the number of flow-m packets in the network will be W at each time-slot. We can thus use discrete-time Markov chain (MC) analysis to study the closed tandem network for flow m. Specifically, let $\vec{Q}(t) = (Q_1(t), \cdots, Q_H(t))$ be the system state, where $Q_i(t)$ is the number of flow-m packets at the ith hop at the beginning of time t. Furthermore, let $\vec{S}(t) = (S_1(t), \cdots, S_H(t))$ be the random schedule vector for flow m at time t, where $S_i(t) = \mathbf{1}_{\{\text{link } i \text{ is scheduled at time } t\}}$. Since $\vec{S}(t)$ is i.i.d. across time-slots, the state at time $t+1$ will only depend on the current state $\vec{Q}(t)$ and the schedule $\vec{S}(t)$. It can be verified that this MC is ergodic, i.e., irreducible, positive recurrent, and aperiodic.

Fig. 2 illustrates an example of the MC for a 3-hop closed tandem network with five packets. The capacities of links 1, 2, and 3 are 1, 1, and 2, respectively. We know that the throughput of flow m is the average number of packets that leave the system, which is equivalent to the average number of packets that leave the last hop, i.e., link H. If the MC is in steady state, we can compute the actual throughput R_m by computing the average number of packets that leave the last hop as follows:

$$R_m = \sum_{i=1}^{\infty} \min(i, c_H)\mu_H P\{Q_H = i\}. \tag{6}$$

If we can compute the actual throughput R_m, then the delay can be obtained by Little's law. Unfortunately, it appears impossible to directly solve the MC. The reason is because the services of different links are correlated. For example, links 1 and 2 will never be scheduled together due to interference, and there is a good chance that links 1 and 3 will be served together. Furthermore, the exact statistics of such correlation are hard to characterize. Moreover, since the capacity of each link is different, it is possible that one link may not be able to use the full capacity to transmit the packets for some flow m. All that we know (from Lemma 2) is a lower-bounded marginal probability μ/c_ℓ that link ℓ is activated to send a flow m packet. Hence, it is difficult to solve the MC directly. To circumvent this difficulty, we next use a novel stochastic dominance approach. Specifically, we construct a sequence of systems such that one system stochastically dominates the other. Through this series of comparison, the original system is reduced to a system where throughput and delay can be directly computed.

A. Comparison Step 1

We start with some definitions and assumptions. In the rest of this section, when we refer to a particular *system*, we mean a version of the H-hop closed tandem network with window-based flow control and window size W. For each system, the random schedule vector is always i.i.d. across time. Furthermore, for different systems, the initial condition for the packet placement is the same, i.e., the number of packets in each queue is the same at time 0. However, within a time-slot, the distribution of the schedule vector is different depending on the system that we refer to. We abuse the notation and denote a system by $\vec{S}(\cdot)$ when the corresponding random schedule vector is denoted by $\vec{S}(t)$. Furthermore, we denote $T(\vec{S}(\cdot))$ and $D(\vec{S}(\cdot))$ as the throughput and delay of system $\vec{S}(\cdot)$.

Consider a system $\vec{S}^{(1)}(\cdot)$. Let the probability distribution of $\vec{S}^{(1)}(t)$ be $P\{\vec{S}^{(1)}(t) = \vec{x}_i\} = p'_i$, $i = 0, \cdots, I$, where $\vec{x}_i = (x_{i1}, \cdots, x_{iH})$ is the ith schedule vector, $x_{ij} = 1$ if link j is activated under the ith schedule vector, and $x_{ij} = 0$ otherwise. We use the convention that $\vec{x}_0 = \vec{0}$. Let $A_\ell = \{i | x_{i\ell} = 1\}$. Notice that $\mu'_\ell \triangleq \sum_{i \in A_\ell} p'_i$ is the marginal probability that link ℓ is scheduled at one time-slot. Assume that system $\vec{S}^{(1)}(\cdot)$ satisfies the following property: (we will discuss in Section IV-D how to treat the case when property (7) is not satisfied)

$$\sum_{\ell=1}^{H} \mu'_\ell = \sum_{\ell=1}^{H} \sum_{i \in A_\ell} p'_i \leq 1, \quad \mu'_\ell c_\ell \geq \mu'. \tag{7}$$

Recall that the key difficulty of analyzing the system is the correlation of the services among links. We now introduce a splitting procedure that converts a given system to another system where links are less likely to be scheduled together.

Construct system $\vec{S}^{(2)}(\cdot)$ as follows. First, pick a schedule vector of $\vec{S}^{(1)}(t)$ with positive probability such that at least

[3]The intuition behind this choice is the following. Suppose that the window size is $2\beta H_m$. Then, β could be used to tune the tradeoff between throughput and delay. If β is too small, the link with larger capacity may not fully utilize its capacity when it serves packets. If β is too large, packets start to pile up at the link with smaller capacity. Therefore, we let $\beta = (c^m_{\min} + c^m_{\max})/2$ to achieve good tradeoff.

two links are scheduled. Assume that this schedule vector is $\vec{x}_1$. Next, choose the smallest ℓ such that $x_{1\ell} = 1$. Let $\vec{e}_\ell$ be the schedule that only schedules link ℓ, and let $\vec{x}_1 - \vec{e}_\ell$ be the schedule that removes link ℓ from $\vec{x}_1$. The distribution of $\vec{S}^{(2)}(t)$ is

$$\begin{aligned} P\left\{\vec{S}^{(2)}(t) = \vec{x}_1 - \vec{e}_\ell\right\} &= p_1' \\ P\left\{\vec{S}^{(2)}(t) = \vec{e}_\ell\right\} &= p_1' \\ P\left\{\vec{S}^{(2)}(t) = \vec{x}_0\right\} &= p_0' - p_1' \\ P\left\{\vec{S}^{(2)}(t) = \vec{x}_i\right\} &= p_i', \qquad i \geq 2. \end{aligned}$$

Note that the schedule $\vec{x}_1$ is now split into two schedules $\vec{e}_\ell$ and $\vec{x}_1 - \vec{e}_\ell$. Let $|\vec{x}_i|$ be the number of links scheduled by $\vec{x}_i$, and recall that $|\vec{x}_1| \geq 2$. We can then show that

$$\begin{aligned} p_0' - p_1' &= 1 - \sum_{i\neq 0} p_i' - p_1' \geq 1 - \sum_{i\neq 0} |\vec{x}_i| p_i' \\ &= 1 - \sum_{i\neq 0}\sum_{\ell=1}^{H} 1_{\{i\in A_\ell\}} p_i' \\ &= 1 - \sum_{\ell=1}^{H}\sum_{i\neq 0} 1_{\{i\in A_\ell\}} p_i' \\ &= 1 - \sum_{\ell=1}^{H}\sum_{i\in A_\ell} p_i' \geq 0. \end{aligned}$$

Note that the last inequality follows from the fact that $\vec{S}^{(1)}(\cdot)$ has property (7). Thus, the probability distribution of $\vec{S}^{(2)}(t)$ is valid. We call $\vec{S}^{(2)}(\cdot)$ a split version of $\vec{S}^{(1)}(\cdot)$. The key intuition behind split $\vec{S}^{(1)}(\cdot)$ to $\vec{S}^{(2)}(\cdot)$ is that there is a stochastic ordering relation called supermodular ordering between $\vec{S}^{(1)}(\cdot)$ and $\vec{S}^{(2)}(\cdot)$. We review the basic definitions, and readers are referred to [28] and [29] for other definitions and basic properties of stochastic ordering.

Definition 1: (Supermodular Function): A function $\phi(\vec{x}) : \{0,1\}^n \to \mathcal{R}$ is said to be supermodular if, for any n-dimensional vectors $\vec{x}_1$ and $\vec{x}_2$, it satisfies that

$$\phi(\vec{x}_1) + \phi(\vec{x}_2) \leq \phi(\vec{x}_1 \wedge \vec{x}_2) + \phi(\vec{x}_1 \vee \vec{x}_2) \tag{8}$$

where $\wedge$ and $\vee$ mean componentwise minimum and maximum.

Definition 2: (Supermodular Ordering): Let $\mathcal{F}$ be the class of all supermodular functions from $\{0,1\}^n$ into $\mathcal{R}$. For two n-dimensional random vectors $\vec{X}$ and $\vec{Y}$, $\vec{X}$ is said to be smaller than $\vec{Y}$ in supermodular order (denoted by $\vec{X} \leq_{\text{sm}} \vec{Y}$) if $E[\phi(\vec{X})] \leq E[\phi(\vec{Y})]$, for all $\phi \in \mathcal{F}$.

Lemma 3: If we have a system $\vec{S}^{(1)}(\cdot)$ with property (7), and another system $\vec{S}^{(2)}(\cdot)$, which is the split version of system $\vec{S}^{(1)}(\cdot)$, then $\vec{S}^{(1)}(t) \geq_{\text{sm}} \vec{S}^{(2)}(t), \forall t$.

Proof: We omit the proof due to the space constraint. Details can be found in the technical report [27]. ■

Definition 2 and Lemma 3 only involve the comparison of the expected value of ϕ taking only one random vector as input. However, later we will compare the throughput that takes a sequence of random vectors as input. Define $\vec{S}^{(i)}(t_1, t_2)$ as the sequence of scheduling vectors $\vec{S}^{(i)}(t_1), \cdots, \vec{S}^{(i)}(t_2)$ for system $\vec{S}^{(i)}(\cdot)$. A useful lemma that reduces the complexity of comparison is as follows.

Lemma 4: Consider two systems $\vec{S}^{(1)}(\cdot)$ and $\vec{S}^{(2)}(\cdot)$ such that the number of packets in each queue is the same for both systems at time 0, i.e., the same initial condition for packet placement. If under any initial condition for packet placement, we have that $E[\phi(\vec{S}^{(1)}(1,t))|\vec{S}^{(1)}(2,t)] \geq E[\phi(\vec{S}^{(2)}(1), \vec{S}^{(1)}(2,t))|\vec{S}^{(1)}(2,t))]$, then we have that $E[\phi(\vec{S}^{(1)}(1,t))] \geq E[\phi(\vec{S}^{(2)}(1,t))]$.

Proof: We omit the proof due to the space constraint. Details can be found in the technical report [27]. ■

Remark: Note that $E[\phi(\vec{S}^{(1)}(1,t))|\vec{S}^{(1)}(2,t)] \geq E[\phi(\vec{S}^{(2)}(1), \vec{S}^{(1)}(2,t))|\vec{S}^{(1)}(2,t))]$ would hold if the function ϕ is a supermodular function with respect to the schedule at time 1 because the schedules from time 2 to time t are fixed to $\vec{S}^{(1)}(2,t)$ by the conditioning. Hence, according to Lemma 4, showing that ϕ is a supermodular function with respect to the schedule at time 1 is sufficient for $E[\phi(\vec{S}^{(1)}(1,t))] \geq E[\phi(\vec{S}^{(2)}(1,t))]$.[4]

Next, we treat the function ϕ as the throughput function f. Since system $\vec{S}^{(i)}(\cdot)$ is ergodic, we know that $T(\vec{S}^{(i)}(\cdot)) = \lim_{t\to\infty} E[f(\vec{S}^{(i)}(1,t))]/t$. By establishing that f is a supermodular function with respect to the schedules at time 1, we can then prove the following result.

Theorem 1: If we have an ergodic system $\vec{S}^{(1)}(\cdot)$ with property (7) and an ergodic system $\vec{S}^{(2)}(\cdot)$, which is the split version of system $\vec{S}^{(1)}(\cdot)$, then $T(\vec{S}^{(1)}(\cdot)) \geq T(\vec{S}^{(2)}(\cdot))$. Moreover, $\vec{S}^{(2)}(\cdot)$ has property (7).

Proof: The proof is given in the Appendix. ■

In other words, with the same window-based flow control, splitting will not increase the average throughput. To the best of our knowledge, this important relationship has not been reported in the literature. Clearly, if Theorem 1 holds, we can iteratively perform further splitting procedures on system $\vec{S}^{(2)}(\cdot)$. After a finite number of iterations, we will reach a system $\vec{S}^{(3)}(\cdot)$ such that each schedule vector only schedules one link. The distribution of $\vec{S}^{(3)}(t)$ is $P\{\vec{S}^{(3)}(t) = \vec{e}_\ell\} = \mu_\ell', \ell = 1, \cdots, H$ and $P\{\vec{S}^{(3)}(t) = \vec{x}_0\} = 1 - \sum_{\ell=1}^{H} \mu_\ell'$. Furthermore, we have that

$$T\left(\vec{S}^{(1)}(\cdot)\right) \geq T\left(\vec{S}^{(3)}(\cdot)\right). \tag{9}$$

B. Comparison Step 2

Although for system $\vec{S}^{(3)}(\cdot)$ each schedule vector only schedules one link, the capacity of each link is still different. Hence, it is still difficult to compute the throughput directly from the MC. We continue with the comparison to reach a system such that the capacity for each link is the same.

Consider a system $\vec{S}^{(4)}(\cdot)$. For this system, the capacity of link 1 is $c_{\max}$, and the capacities of every other link ℓ is still c_ℓ. The distribution of $\vec{S}^{(4)}(t)$ is given by $P\{\vec{S}^{(4)}(t) = \vec{e}_1\} = \mu_1''$, $P\{\vec{S}^{(4)}(t) = \vec{e}_\ell\} = \mu_\ell', \ell = 2, \cdots, H$, and $P\{\vec{S}^{(4)}(t) = \vec{x}_0\} = 1 - \mu_1'' - \sum_{\ell=2}^{H} \mu_\ell'$, where $\mu_1'' c_{\max} = \mu_1' c_1$. In other words, system $\vec{S}^{(3)}(\cdot)$ and $\vec{S}^{(4)}(\cdot)$ only differ in the capacity and scheduling probability of link 1. Furthermore, system $\vec{S}^{(4)}(\cdot)$

[4]Careful readers may raise the question that why do we not directly prove ϕ to be a supermodular function with respect to the entire sequence of schedules from 1 to t. It turns out that when we take the function ϕ as the throughput function f in the following derivation, such a function is *not* supermodular with respect to the entire sequence of schedules, even though it is supermodular with respect to each variable (see [27] for the detailed example.) Hence, the simplification offered by Lemma 4 is in fact quite essential to establish our main Theorem 1.

still satisfies property (7). It turns out that we can still prove the throughput order relation between $\vec{S}^{(3)}(\cdot)$ and $\vec{S}^{(4)}(\cdot)$ as illustrated in the following proposition.

Proposition 2: $T(\vec{S}^{(3)}(\cdot)) \geq T(\vec{S}^{(4)}(\cdot))$.

Proof: Due to page limits, we omit the proof, which can be found in the technical report [27]. ∎

This proposition says that when the product of the scheduling probability and the capacity remains fixed, if a link is scheduled more often but with less capacity, it will achieve higher throughput. The intuition is that in the system with higher capacity links, the service is more bursty, which will lead to smaller throughput.[5] It is not hard to see that using Proposition 2, we can iteratively show that

$$T\left(\vec{S}^{(3)}(\cdot)\right) \geq T\left(\vec{S}^{(5)}(\cdot)\right) \tag{10}$$

where for system $\vec{S}^{(5)}(\cdot)$, the capacity of all the links is $c_{\max}$, and the probability of scheduling each link ℓ is μ''_ℓ, where $\mu''_\ell c_{\max} = \mu'_\ell c_\ell$. As we will see in Section IV-C, the lower bound of $T(\vec{S}^{(5)}(\cdot))$ can be easily calculated.

C. Comparison Step 3

Let us take one more step. Consider another system $\vec{S}^{(6)}(\cdot)$. The capacity for all the links is $c_{\max}$. The distribution of $\vec{S}^{(6)}(t)$ is $P\{\vec{S}^{(6)}(t) = \vec{e}_\ell\} = \mu''_{\min}, \ell = 1, \cdots, H$ and $P\{\vec{S}^{(6)}(t) = \vec{x}_0\} = 1 - H\mu''_{\min}$, where $\mu''_{\min} = \min_\ell \mu''_\ell$. It can be verified that $T(\vec{S}^{(5)}(\cdot)) \geq T(\vec{S}^{(6)}(\cdot))$.

Lemma 5: $T(\vec{S}^{(5)}(\cdot)) \geq T(\vec{S}^{(6)}(\cdot))$.

Proof: Due to page limits, we omit the proof (which uses Lemma 8 in the Appendix). Details can be found in the technical report [27]. ∎

This result is intuitive because, for every link ℓ, $P\{\vec{S}^{(5)}(t) = \vec{e}_\ell\} \geq P\{\vec{S}^{(6)}(t) = \vec{e}_\ell\}$. Thus, the throughput of system $\vec{S}^{(5)}(\cdot)$ should be no smaller than the throughput of system $\vec{S}^{(6)}(\cdot)$. The throughput of system $\vec{S}^{(6)}(\cdot)$ has a closed-form solution as follows. Note that the MC for system $\vec{S}^{(6)}(\cdot)$ is similar to that of a closed tandem network of $M/M/1$ queues with identical service rates [31, p. 186].

Lemma 6: $T(\vec{S}^{(6)}(\cdot)) = c_{\max}\mu''_{\min} w/(w + H - 1)$.

Proof: Recall that $W = c_{\max} w$, i.e., the window size is a multiple of $c_{\max}$. Furthermore, each link has capacity $c_{\max}$. Hence, blocks of $c_{\max}$ packets must move from queues to queues. Consider the Markov chain of system $\vec{S}^{(6)}(\cdot)$ and a particular state $\vec{n} = (n_1, \cdots, n_H)$, where n_i is the number of blocks at queue i (a block consists of $c_{\max}$ packets). Since each possible schedule $\vec{e}_\ell$ for system $\vec{S}^{(6)}(\cdot)$ schedules only one link, there are the same number of incoming transitions and outgoing transitions for an arbitrary state $\vec{n}$. If we assume that $P(\vec{n}) = x$ for any state $\vec{n}$, then it will satisfy the global balance equation. Therefore, to obtain the solution of this stationary Markov chain, we only need to normalize $\sum P(\vec{n}) = 1$. Hence

$$x = \frac{1}{\text{number of possible states}} = \frac{1}{\binom{w+H-1}{w}}.$$

[5] A similar observation has been reported in the literature for continuous-time systems when the holding-time distribution is more bursty. There, a stochastic ordering relationship called increasing convex ordering was used [30]. Proposition 2 can be viewed as an extension to discrete-time systems with batch service.

We can then show that

$$P\{Q_H = 0\} = x\left(\sum_{\vec{n}} \mathbf{1}_{\{n_H = 0\}}\right) = \frac{\binom{w+H-2}{w}}{\binom{w+H-1}{w}} = \frac{H-1}{w+H-1}$$

and

$$T\left(\vec{S}^{(6)}(\cdot)\right) = c_{\max}\mu''_{\min}\left[1 - P\{Q_H = 0\}\right] = \frac{w c_{\max}\mu''_{\min}}{w+H-1}.$$

∎

Using (9) and (10) and Lemmas 5 and 6, we get a lower bound of $T(\vec{S}^{(1)}(\cdot))$.

D. Throughput Lower Bound and Delay Upper Bound

Until this point, we have assumed that system $\vec{S}^{(1)}(\cdot)$ satisfies property (7). We now discuss how to treat the case when the original system $\vec{S}(\cdot)$ does not satisfy property (7). Suppose that the distribution of $\vec{S}(t)$ is $P\{\vec{S}(t) = \vec{x}_i\} = p_i$, $i = 0, \cdots, I$. Define system $\vec{S}^{(1)}(\cdot)$ as follows. For $i \neq 0$, let $p'_i = p_i/(\sum_{\ell=1}^H \mu_\ell)$, and $p'_0 = 1 - \sum_{i\neq 0} p'_i$. The distribution of $\vec{S}^{(1)}(t)$ is $P\{\vec{S}^{(1)}(t) = \vec{x}_i\} = p'_i$, $i = 0, \cdots, I$. Recall that $A_\ell = \{i | x_{i\ell} = 1\}$. We can then show that

$$\sum_{\ell=1}^H \mu'_\ell = \sum_{\ell=1}^H \sum_{i\in A_\ell} p'_i = \frac{\sum_{\ell=1}^H \sum_{i\in A_\ell} p_i}{\sum_{j=1}^H \mu_j} = \frac{\sum_{\ell=1}^H \mu_\ell}{\sum_{j=1}^H \mu_j} \leq 1.$$

We also have from (5) that $\mu'_\ell c_\ell = \mu_\ell c_\ell / \sum_{i=1}^H \mu_i \geq \mu / \sum_{i=1}^H \mu_i \triangleq \mu'$. Thus, system $\vec{S}^{(1)}(\cdot)$ has property (7). The relationship between $T(\vec{S}(\cdot))$ and $T(\vec{S}^{(1)}(\cdot))$ is given by the following lemma.

Lemma 7: $T(\vec{S}(\cdot)) = (\sum_{\ell=1}^H \mu_\ell) T(\vec{S}^{(1)}(\cdot))$.

Proof: By (6), $T(\vec{S}(\cdot)) = \sum_{i=1}^\infty \min(i, c_H)\mu_H P_{\vec{S}}\{Q_H = i\}$, and $T(\vec{S}^{(1)}(\cdot)) = \sum_{i=1}^\infty \min(i, c_H)\mu'_H P_{\vec{S}^{(1)}}\{Q_H = i\}$. If we can show that system $\vec{S}(\cdot)$ and system $\vec{S}^{(1)}(\cdot)$ have the same stationary probability for all the states, this lemma can be proved by noting that $\mu_H = \mu'_H(\sum_{\ell=1}^H \mu_\ell)$. To show that system $\vec{S}(\cdot)$ and system $\vec{S}^{(1)}(\cdot)$ have the same stationary probability for all the states, notice that by changing system $\vec{S}(\cdot)$ to system $\vec{S}^{(1)}(\cdot)$, for each state we divide each incoming or outgoing transition probability by $\sum_{\ell=1}^H \mu_\ell$. Since the global balance equation will not change after the transition probability is divided by a constant, systems $\vec{S}(\cdot)$ and $\vec{S}^{(1)}(\cdot)$ will have the same steady-state distribution. ∎

It then follows from this lemma, $T(\vec{S}^{(1)}(\cdot)) \geq T(\vec{S}^{(6)}(\cdot)) = c_{\max}\mu''_{\min} w/(w + H - 1)$, and $c_{\max}\mu''_{\min} \geq \mu'$ that

$$T\left(\vec{S}(\cdot)\right) = \left(\sum_{\ell=1}^H \mu_\ell\right) T\left(\vec{S}^{(1)}(\cdot)\right) \geq \frac{\mu w}{w+H-1}.$$

By Little's law, $W = T(\vec{S}(\cdot))D(\vec{S}(\cdot))$. Thus

$$D\left(\vec{S}(\cdot)\right) \leq c_{\max}(w + H - 1)/\mu.$$

This completes the proof of Proposition 1.

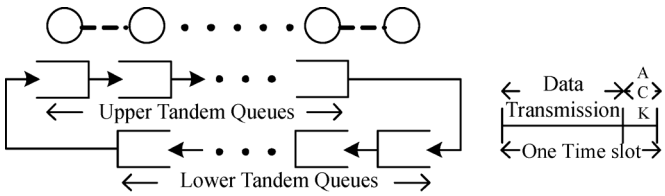


Fig. 3. Upper (resp. lower) tandem queues store packets (resp. ACKs).

V. Implementation Issues

In this section, we discuss some practical issues for implementing our algorithm. We will address three components of our scheme: window-based flow control, virtual-rate computation, and scheduling. First, the window-based flow control requires a backward channel for communicating the ACKs. This backward channel can be easily implemented as follows. Immediately after a link transmits a flow-m packet, the receiving node will respond with an acknowledgement, which piggybacks an ACK for flow m that it has received from the destination in the past. With this mechanism, each link can be modeled as an upper queue for the forward direction and a lower queue for the backward direction. The window-based flow control for a given flow m can then be modeled as a $2H_m$-hop closed queueing network in Fig. 3. Note that both the upper queue and the lower queue will be served with probability lower-bounded by Lemma 2. It is then easy to see that we can again use the technique of Section IV to derive the throughput and delay bounds. The main difference is that the number of hops is changed from H_m to $2H_m$. To maintain the throughput, we can increase the window size from W_m to $2W_m$. Then, by Proposition 1 the throughput is lower-bounded by $r_m^*(1-\epsilon)2w/(2w+2H-1)$, which is greater than or equal to $r_m^*(1-\epsilon)w/(w+H-1)$. Hence, the throughput performance is comparable to the case with instantaneous feedback. As for the delay, the round-trip delay bound is increased to $c_{\max}(2w+2H-1)/(r_m^*(1-\epsilon))$, which is roughly $2c_{\max}(w+H-1)/(r_m^*(1-\epsilon))$. Hence, the order-optimality of the delay bound is not affected.

Second, in the virtual-rate computation algorithm, each link ℓ needs to collect the total normalized virtual rate from links in the interference set $\mathcal{E}_\ell$, and each source node needs to collect the sum of the dual variables from all the links that interfere with at least one link along its route. In practice, such information exchange can be easily achieved by first piggybacking the most recent virtual rate information on each packet sent by the source node. Then, each link can check the packet from all the flows that pass through it and update the total virtual rate. When a link is scheduled to transmit, we can further piggyback the value of the most recent total normalized virtual rate in the packet. Hence, all the links that interfere with the scheduled link can overhear this information. Similarly, when a link is scheduled to transmit, we can also piggyback the most recent value of the dual variable. All the links that interfere with the scheduled link can also overhear this information. Each link along the route of flow m can piggyback the most recent sum of the normalized dual variables on each ACK sent by the destination node. Consequently, the source node of flow m can obtain the required information through the received ACK. Note that although the virtual rates and dual variables are updated asynchronously, our window-based flow control algorithm ensures that the delay of such information exchange will not be large. Hence, we expect that the virtual-rate computation algorithm will still converge with suitable choices of the step sizes [25], [32].

Finally, in the scheduling algorithm, each link i must first collect the total normalized virtual rate from the link in its interference set, i.e., $\sum_{k\in\mathcal{E}_i} a_k(t)$. As we described above, this information is already available when we perform the virtual-rate computation. Then, each link can again piggyback this value in the transmitting data packet, and every link that interferes with the transmitting link can overhear this information. As a result, each link ℓ can obtain $\max_{i\in\mathcal{E}_\ell}(\sum_{k\in\mathcal{E}_i} a_k(t))$. We note that each link may now attempt with outdated information, but it will not affect our delay bound. This is because after the virtual-rate computation algorithm converges, the virtual rate will not change significantly.

Readers can see that under our proposed algorithm, each node only needs to perform a constant number of operations with a constant time of F mini-slots. Note that larger F will lead to smaller ϵ as described in Lemma 2, but at the cost of larger overhead. In our simulations in Section VI, we choose the size of F to be 32. Assume that the length of a mini-slot is 20 μs; the overhead is then 640 μs. This amount of overhead is comparable to standard 802.11 protocols. When $F=32$, by Lemma 2, the value ϵ of our lower bound will be roughly 0.14, which is already small. When we run the simulations in Section VI, we use the improved algorithm as discussed in Section III-B, which will lead to further performance improvement. As the reader will see in Section VI, our simulation shows that $F=32$ is sufficient to make the actual throughput under our algorithm be larger than the calculated virtual rate. The impact of this overhead also depends on the packet length and the length of a time-slot. If the length of a time-slot is T_0, and the length of each mini-slot is T_1, then the backoff mini-slots will lead to another reduction factor of $(T_0-FT_1)/T_0$ on the throughput. Note that for a fixed F, the complexity of our algorithm is $\mathcal{O}(1)$, which is significantly lower than the $\mathcal{O}(N)$ per-node operations required by the algorithm in [20].

VI. Simulation Results

We start from simulating our proposed algorithm using the linear topology in Fig. 1 with H links under the one-hop interference constraint. The capacities of the leftmost four links are 5, 2, 3, and 4, respectively. Then, every four links repeat this pattern. We use the improved version of our scheduling algorithm as discussed in Section III-B, and we let the source node (resp. each link) collect the sum of the dual variables along the path (resp. virtual rate) asynchronously as described in the second paragraph of Section V. We set the number of backoff mini-slots $F=32$. The window size of each short flow is $2c_i$ if the short flow passes link i. The window size of the long flow is $(c_{\min}+c_{\max})H$. The utility function is $H\log(\cdot)$ for the long flow and $5\log(\cdot)$ for each short flow. Hence, when we increase the number of hops, the optimal rate assignment for the flows will be roughly the same. This will help us to observe how the average delay changes as the number of hops increases while the throughput is relatively fixed. We also use BP-α to represent the back-pressure algorithm with step size α and SBP-α to represent the shadow back-pressure algorithm with step size α [9].

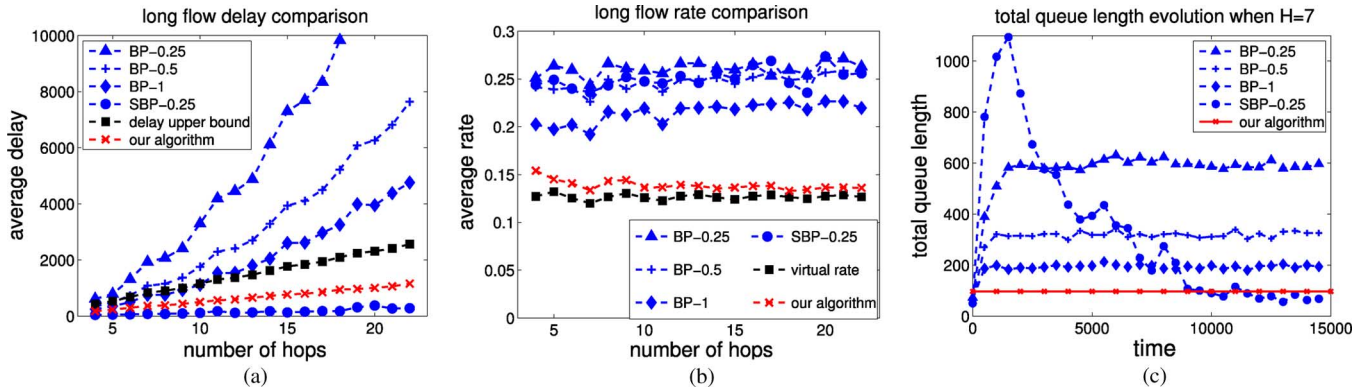


Fig. 4. Simulation results of the linear topology: (a) average delay of the long flow for different number of hops; (b) rate of the long flow for different number of hops; and (c) evolution of the total queue length when $H = 7$.

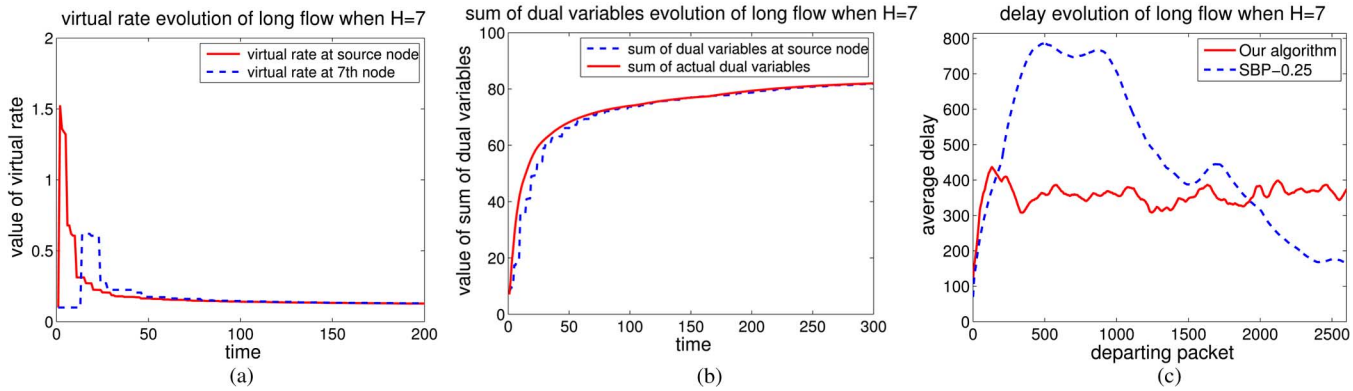


Fig. 5. Simulation results of the linear topology during the transient phase (all figures are for the long flow when $H = 7$): (a) evolution of the virtual rate; (b) evolution of the sum of dual variables; and (c) delay evolution.

We first compare the performance of our proposed algorithm with the standard back-pressure (BP) algorithm (for different step sizes). Fig. 4(a) shows that the average delay of our algorithm increases linearly with the number of hops. On the other hand, at all step sizes, the average delay of the BP algorithm increases quadratically with the number of hops, which is much larger than that of our algorithm. Note that our algorithm outperforms the BP algorithm in the delay performance even though the BP algorithm utilizes centralized computation. Moreover, our average delay curve is below the delay upper bound derived in Section IV. This verifies our delay analysis result. In Fig. 4(b), we plot the corresponding long-flow throughput of our algorithm versus BP algorithm. We can see that the throughput of our distributed algorithm is indeed more than half of the centralized and high-complexity BP algorithm. Another interesting observation is that when the step size is large (BP-1), the throughput differs significantly from those with smaller step sizes.[6] This indicates that the delay reduction [in Fig. 4(a)] of the BP algorithm at such a large step size is at the cost of losing its optimal control capability. In contrast, the step size in our proposed algorithm does not directly affect the delay.

Second, we compare the performance of our proposed algorithm with the *shadow back-pressure* (SBP) algorithm proposed in [9]. Note that the average delay curve of the SBP algorithm also shows linear scaling as shown in Fig. 4(a) (although this has not been formally proved in [9]). Furthermore, the corresponding long-flow throughput is roughly the same as the BP algorithm, which is optimal. However, like the BP algorithm, the SBP algorithm requires centralized computation to achieve maximum capacity region. Furthermore, we observe that SBP requires roughly 7000 time-slots for the whole algorithm to converge, and the total queue length inside the network will first rise to a very large value as shown in Fig. 4(c). In contrast, the control variables under our algorithm will converge after around 200 time-slots as shown in Fig. 5(a) and (b). Furthermore, because of window-based flow control, the total queue backlog of our algorithm is consistently the lowest at all time, even during the transient period. Finally, we plot the delay evolution of the departing packets when $H = 7$ as shown in Fig. 5(c). Specifically, we plot the average delay over the previous 200 departing packets right before each departing packet. The simulation result shows that the average delay of SBP will be significantly larger in the transient phase. In contrast, the average delay of our algorithm does not deviate significantly from the theoretical value even in the transient period.[7]

In Fig. 6, we demonstrate the per-flow controllability of our scheme by plotting the throughput-delay curve for the long flow. We first fix the window size of each short flow to be $2c_i$ if the short flow passes link i. We then vary the window size of the long flow. As the window size increases, the average throughput

[6]Note that the delay order of the BP algorithm is always quadratic for different step sizes. This implies that simply operating away from the boundary of the capacity region cannot change the order of the delay performance.

[7]Readers may note that even though the virtual rate of the long flow at the initial phase is higher, it has not led to more congestion. The reason is that the source node does not directly use the virtual rate to inject packets. Instead, our algorithm uses window-based flow-control, which controls the congestion level in the network all the time.

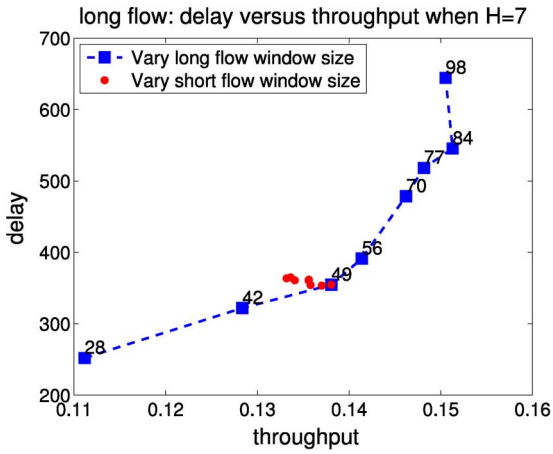


Fig. 6. Long flow delay versus throughput when $H = 7$ for the linear topology. Numbers along the curve are the window size of the long flow.

TABLE I
VALUE OF h THAT IS USED TO DETERMINE THE UTILITY FUNCTION OF THE LONG FLOW FOR DIFFERENT GRID SIZES

Grid Size n	3	4	5	6	7	8	9	10
h	2	2.3	3.2	4.3	6	7.5	9.7	11.5

of the long flow will approach to a limit, and the delay will increase linearly with window size. This curve shows that when the window size is equal to $(c_{\min} + c_{\max})H = 49$, we can achieve a good balance between throughput and delay. Next, we fix the window size of the long flow to be $(c_{\min} + c_{\max})H$ and vary the window size of the short flows. As shown in Fig. 6, all points are concentrated around a small region, which demonstrates that the performance of the long flow does not change when the window size of the short flows changes.

Next, we simulate our proposed algorithm using a much larger topology, i.e., the $n \times n$ grid topology in Fig. 7(a). For each $k = 1, \cdots, n-1$, there is one k-hop flow in each row (resp. column) from left to right (resp. top to bottom). Moreover, there is a $(2n-2)$-hop long flow from the northwest corner node to the southeast corner node. The capacity of each link that belongs to the ith row is 5, if i is an odd number, and 3, if i is an even number. The capacity of each link that belongs to the ith column is 4, if i is an odd number, and 2, if i is an even number. An example for the 3×3 grid topology is shown in Fig. 7(a). For the long flow, the utility function is $h\log(\cdot)$, where the value of h is chosen as shown in Table I. For all the other flows, the utility function is $\log(\cdot)$ for all grid sizes. As a result, when we increase the grid size, the rate assignment for the long flow will be roughly the same. For all flows except the long flow, the window size is two times the capacity of the first-hop link times the number of hops. The window size of the long flow is $(c_{\max} + c_{\min})$ times the number of hops. The delay of the long flow is presented in Fig. 7(b). When we increase the grid size from 3 to 10, the average delay of our algorithm still increases roughly linearly with the number of hops, and, at all step sizes, the average delay of the back-pressure algorithm increases quadratically with the number of hops. In Fig. 7(c), we plot the corresponding long-flow throughput of our algorithm versus the back-pressure algorithm. We can see that the throughput of our distributed algorithm is still more than half of the back-pressure algorithm.[8]

VII. CONCLUSION

In this paper, we propose a low-complexity and distributed algorithm for joint congestion control and scheduling in multihop wireless networks under a general interference model. The main ideas of the proposed algorithm are to control the congestion with window-based flow control and to use both virtual-rate information and queue information (rather than just queue information) to perform scheduling. Our scheduling algorithm is fully distributed and only requires a constant time (independent of network size) to compute a schedule [8]. We prove that our congestion control and scheduling algorithm can utilize nearly $1/\mathcal{K}$ of the capacity region and provide a per-flow delay bound that increases linearly with the number of hops. Our analysis uses a novel stochastic dominance approach to derive the per-flow throughput and delay bounds. In our future work, we will study how to extend this novel technique to the case with dynamic routing.

APPENDIX
PROOF OF THEOREM 1

Because the marginal probability of scheduling link ℓ in $\vec{S}^{(1)}(\cdot)$ and $\vec{S}^{(2)}(\cdot)$ is the same for each ℓ, it is easy to see that $\vec{S}^{(2)}(\cdot)$ still has property (7).

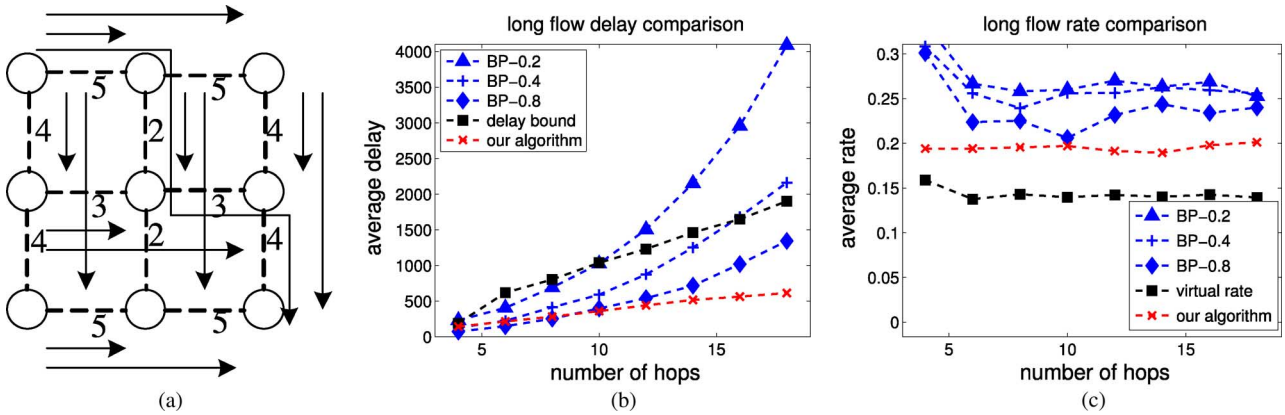
To show that $T(\vec{S}^{(1)}(\cdot)) \geq T(\vec{S}^{(2)}(\cdot))$, fix a packet placement at time 0. Under window-based flow control with window size W, let f be a function that maps a given sequence of schedule vectors to the total number of packets leaving queue H at the end of time t. We know that $\lim_{t\to\infty} E[f(\vec{S}^{(1)}(1,t))]/t = T(\vec{S}^{(1)}(\cdot))$, and $\lim_{t\to\infty} E[f(\vec{S}^{(2)}(1,t))]/t = T(\vec{S}^{(2)}(\cdot))$. Therefore, we only need to show that $E[f(\vec{S}^{(1)}(1,t))] \geq E[f(\vec{S}^{(2)}(1,t))], \forall t$. By Lemma 4, we can reduce the problem to proving that for systems $\vec{S}^{(1)}(\cdot)$ and $\vec{S}^{(2)}(\cdot)$, if the number of packets in each queue is the same at time 0, then $E[f(\vec{S}^{(1)}(1,t))|\vec{S}^{(1)}(2,t)] \geq E[f(\vec{S}^{(2)}(1))|\vec{S}^{(1)}(2,t)]$. It then comes from Definition 2 and Lemma 3 that we only need to prove that f is a supermodular function at time 1. In other words, consider the following four deterministic systems Y_i, $i = 1, \cdots, 4$. They have the same initial packet placement, i.e., the initial number of packets in each queue is the same for all four systems. For system Y_i, from time 1 to some time t, it uses a sequence of deterministic schedules, $\vec{z}_i, \vec{y}(2), \cdots, \vec{y}(t)$. It is easy to see that the only difference for these four deterministic systems is the schedule at the first time-slot. Let $\vec{z}_i, 1 \leq i \leq 4$ be schedules satisfying

$$\vec{z}_2 = \vec{z}_3 \wedge \vec{z}_4, \quad \vec{z}_1 = \vec{z}_3 \vee \vec{z}_4. \tag{11}$$

Notice that $\vec{z}_2 \preccurlyeq \vec{z}_3 \preccurlyeq \vec{z}_1$, and $\vec{z}_2 \preccurlyeq \vec{z}_4 \preccurlyeq \vec{z}_1$, where $\preccurlyeq$ means componentwise smaller. Let $T_i(t)$ be the throughput of system Y_i at time t, i.e., $T_i(t) = f(\vec{z}_i, \vec{y}(2), \vec{y}(3), \cdots, \vec{y}(t))$. To prove Theorem 1, we only need to show that

$$T_1(t) + T_2(t) \geq T_3(t) + T_4(t) \qquad \forall t. \tag{12}$$

[8]Due to space constraints, additional figures for the delay evolution of our algorithm are provided in the technical report [27].



(a) (b) (c)

Fig. 7. Simulation results of the grid topology: (a) network setting; (b) delay of the long flow for different number of hops; and (c) rate of the long flow for different number of hops.

We first prove two lemmas required for proving (12). Since we only care about the total number of departing packets for each system, without loss of generality, we can assume that each queue uses FIFO discipline. Let $D_k(i,j)$ be the departing time of the ith departing packet from queue j for system Y_k, and let $A_k(i,j)$ be the arriving time of the ith arriving packet at queue j for system Y_k. Note that $A_k(i,j+1) = D_k(i,j), 1 \le j \le H-1$ and $A_k(i,1) = D_k(i,H)$.

Lemma 8: Consider systems Y_k and Y_w. If $\vec{z}_k \succeq \vec{z}_w$, then $D_k(i,j) \le D_w(i,j)$ and $A_k(i,j) \le A_w(i,j)$.

The intuition of Lemma 8 is that system Y_k has more services than Y_w at time 1. Therefore, packets in system Y_k move faster to the next hop than those in system Y_w. Since the schedules for system Y_k and Y_w are the same after time 1, this trend will preserve after time 1. The rigorous proof is as follows.

Proof of Lemma 8: We prove this lemma by induction on time t. We will show the following induction hypothesis for any time $t = 1, 2, \cdots$.

Induction Hypothesis (for Time t): For all $i \ge 1$, $j = 1, 2, \cdots, H$, if $D_w(i,j) \le t$, then $D_k(i,j) \le D_w(i,j)$.

If this holds for all t, it will imply that $D_k(i,j) \le D_w(i,j)$ and $A_k(i,j) \le A_w(i,j)$. Notice that we only need to prove the inequality for the departing time because for any system Y_s, we have that $A_s(i,j+1) = D_s(i,j), 1 \le j \le H-1$ and $A_s(i,1) = D_s(i,H)$.

First, consider $t = 1$ and fix a queue $j = 1, 2, \cdots, H$. We have the following two cases.

Case 1: Queue j is not scheduled by $\vec{z}_w$. In this case, for any index i, $D_w(i,j) > 1$, and the induction hypothesis trivially holds for time 1.

Case 2: Queue j is scheduled by $\vec{z}_w$. If queue j is scheduled by $\vec{z}_w$, then queue j will also be scheduled by $\vec{z}_k$ because $\vec{z}_k \succeq \vec{z}_w$. Suppose that the number of initial packets at queue j is q_j. Since system Y_k and Y_w have the same initial packet placement, if $\min(c_j, q_j)$ packets are served by queue j for system Y_w, then $\min(c_j, q_j)$ packets are also served by queue j for system Y_k. Hence, for all $i \le \min(c_j, q_j)$, we have that $D_k(i,j) = D_w(i,j) = 1$. Furthermore, for all $i > \min(c_j, q_j)$, $D_w(i,j) > 1$. Thus, the induction hypothesis holds for time 1.

Next, assume that the induction hypothesis holds for time t. We next show that the induction hypothesis must also hold for time $t+1$. Note that the schedules for all systems are the same after time 1. Fix a queue $j = 1, 2, \cdots, H$. We have the following two cases.

Case 1: Queue j is not scheduled at time $t+1$ or queue j is scheduled at time $t+1$, but for system Y_w, no packets are served at queue j at time $t+1$. In this case, there is no index h such that $D_w(h,j) = t+1$. Hence, the set of indices i satisfying $D_w(i,j) \le t$ is equal to the set of indices i satisfying $D_w(i,j) \le t+1$, and the induction hypothesis trivially holds for time $t+1$.

Case 2: Queue j is scheduled at time $t+1$, and for system Y_w, at least one packet is served at queue j at time $t+1$. We prove by contradiction that the induction hypothesis will hold for time $t+1$. In this case, for system Y_w, queue j will serve packets at time $t+1$, and suppose the first one is the pth departing packet. From the induction hypothesis, we know that for all $h < p$

$$D_k(h,j) \le D_w(h,j) \le t. \tag{13}$$

Assume that one of the departing packets (say the ith departing packets) does not satisfy the induction hypothesis, i.e.,

$$D_k(i,j) > D_w(i,j) = t+1. \tag{14}$$

Since $D_w(i,j) = t+1$, for system Y_w, queue j must serve at least $i-p+1$ packets, i.e.,

$$c_j \ge i - p + 1. \tag{15}$$

We have the following two subcases.

Subcase 2.1: $i \le q_j$. This implies that the ith departing packet is stored at queue j at time 0. By (13), we have that, for system Y_k, there are at most $i-p$ packets in front of the ith departing packets at queue j at time $t+1$. It then follows from (15) that $D_k(i,j) = t+1$. This contradicts with (14).

Subcase 2.2: $i > q_j$. Let $i' = i - q_j$. This implies that the ith departing packet is the i'th arriving packet of queue j. From the induction hypothesis, for system Y_k, we can deduce that $A_k(i',j) \le A_w(i',j) \le t$. This implies that for system Y_k, the ith departing packet is stored at queue j at time $t+1$. By (13), we have that for system Y_k, there are at most $i-p$ packets in front of the ith departing packets at queue j at time $t+1$. It then follows from (15) that $D_k(i,j) = t+1$. This contradicts with (14). ■

This lemma implies that if $\vec{z}_k \succeq \vec{z}_w$, then $T_k(t) \geq T_w(t)$. We now prove another lemma. Recall that $\vec{z}_2 = \vec{z}_3 \wedge \vec{z}_4$.

Lemma 9: $D_2(i,j) \leq \max(D_3(i,j), D_4(i,j))$ and $A_2(i,j) \leq \max(A_3(i,j), A_4(i,j))$.

The intuition behind this lemma is that the packets in system Y_2 will move slower to the next hop than system Y_3 or Y_4 at time 1, but never both. Since the schedules for systems Y_2, Y_3, and Y_4 are the same after time 1, this trend will preserve after time 1. The rigorous proof is as follows.

Proof of Lemma 9: We prove this lemma by induction on time t. We will show the following induction hypothesis for any time $t = 1, 2, \cdots$.

Induction Hypothesis (for Time t): For all $i \geq 1$, $j = 1, 2, \cdots, H$, if $D_3(i,j) \leq t$ and $D_4(i,j) \leq t$ then $D_2(i,j) \leq \max(D_3(i,j), D_4(i,j))$.

If this holds for all t, it implies that $D_2(i,j) \leq \max(D_3(i,j), D_4(i,j))$ and $A_2(i,j) \leq \max(A_3(i,j), A_4(i,j))$. Notice that we only need to prove the inequality for the departing time because for any system Y_s, we have that $A_s(i, j+1) = D_s(i,j)$, $1 \leq j \leq H-1$ and $A_s(i,1) = D_s(i,H)$.

First, consider $t = 1$ and fix a queue $j = 1, 2, \cdots, H$. Recall that $\vec{z}_2 = \vec{z}_3 \wedge \vec{z}_4$. We have the following two cases.

Case 1: Queue j is not scheduled by $\vec{z}_2$. If queue j is not scheduled by $\vec{z}_2$ at time 1, then it will also not be scheduled by either $\vec{z}_3$ or $\vec{z}_4$. Hence, for any index i, $\max(D_3(i,j), D_4(i,j)) > 1$, and the induction hypothesis trivially holds for time 1.

Case 2: Queue j is scheduled by $\vec{z}_2$. If queue j is scheduled by $\vec{z}_2$, it will also be scheduled by $\vec{z}_3$ and $\vec{z}_4$. Suppose that the number of initial packets at queue j is q_j. Hence, if $\min(q_j, c_j)$ packets are served by queue j for system Y_3 and Y_4, $\min(q_j, c_j)$ packets are also served by queue j for system Y_2. Hence, for all $i \leq \min(c_j, q_j)$, we have that $D_2(i,j) = \max(D_3(i,j), D_4(i,j)) = 1$. Furthermore, for all $i > \min(c_j, q_j)$, $D_3(i,j) > 1$ and $D_4(i,j) > 1$. Thus, the induction hypothesis holds for time 1.

Next, assume that the induction hypothesis holds for time t. We next show that the induction hypothesis must also hold for time $t+1$. Note that the schedules for all systems are the same after time 1. Fix a queue $j = 1, 2, \cdots, H$, and assume that the total number of departing packets at queue j of system Y_3 and Y_4 between time 1 and time t is p_3 and p_4, respectively. Without loss of generality, assume that $p_3 \leq p_4$.

We have the following two cases.

Case 1: Queue j is not scheduled at time $t+1$ or queue j is scheduled at time $t+1$, but for system Y_3, no packets are served at queue j at time $t+1$. In this case, there is no index h with $D_3(h,j) = t+1$. Hence, the set of indices i satisfying $D_3(i,j) \leq t$ and $D_4(i,j) \leq t$ is equal to the set of indices i satisfying $D_3(i,j) \leq t+1$ and $D_4(i,j) \leq t+1$, and the induction hypothesis trivially holds for time $t+1$.

Case 2: Queue j is scheduled at time $t+1$, and for system Y_3, at least one packet is served at queue j at time $t+1$. We prove by contradiction that the induction hypothesis will hold for time $t+1$. Since we have assumed that the induction hypothesis holds for time t, we know that for all $h \leq p_3$

$$D_2(h,j) \leq \max(D_3(h,j), D_4(h,j)) \leq t. \tag{16}$$

Suppose that the induction hypothesis does not hold at queue j for time $t+1$. Then, there must exist an index i such that

$$\begin{aligned} &D_3(i,j) \leq t+1 \\ &D_4(i,j) \leq t+1 \\ &D_2(i,j) > \max(D_3(i,j), D_4(i,j)). \end{aligned} \tag{17}$$

Note that in this case we must have that $D_3(i,j) = t+1$. To see this, note that if $D_3(i,j) \leq t$, then $i \leq p_3$. Thus, (17) contradicts with (16). Hence, we must have that $D_3(i,j) = t+1$, which means that for system Y_3 the departing time of the ith departing packet from queue j is $t+1$. Furthermore, combined with (17), it implies that

$$D_2(i,j) > t+1 \tag{18}$$

i.e., for system Y_2, the departing time of the ith departing packet from queue j is larger than $t+1$. However, this leads to a contradiction as we show below. Without loss of generality, let i be the first such index [i.e., that satisfies $D_3(i,j) = t+1$, (17), and (18)]. Since $D_3(i,j) = t+1$, for system Y_3, queue j must serve at least $i - p_3$ packets, i.e.,

$$c_j \geq i - p_3. \tag{19}$$

We have the following two subcases.

Subcase 2.1: $i \leq q_j$. This implies that the ith departing packet is stored at queue j at time 0. By (16), we know that, for system Y_2, there are at most $i - p_3 - 1$ packets in front of the ith departing packets at queue j at time $t+1$. It then follows from (19) that $D_2(i,j) = t+1$. This contradicts with (18).

Subcase 2.2: $i > q_j$. Let $i' = i - q_j$. This implies that the ith departing packet is the i'th arriving packet of queue j. Since $D_3(i,j) = t+1$ and $D_4(i,j) \leq t+1$, we conclude that $A_3(i',j) \leq t$ and $A_4(i',j) \leq t$, i.e., the arrival time can only be smaller than or equal to t. From the induction hypothesis, we can deduce that $A_2(i',j) \leq \max(A_3(i',j), A_4(i',j)) \leq t$. This implies that for system Y_2, the ith departing packet is stored at queue j at time $t+1$. By (16), we have that, for system Y_2, there are at most $i - p_3 - 1$ packets in front of the ith departing packets at queue j at time $t+1$. It then follows from (19) that $D_2(i,j) = t+1$. This contradicts with (18). ■

Now, we prove (12).

Proof of Equation (12): At time t, without loss of generality, assume that $T_3(t) \leq T_4(t)$. This implies that $D_3(T_3(t), H) \leq t$ and $D_4(T_3(t), H) \leq t$. From Lemma 9, we have that $D_2(T_3(t), H) \leq t$. Hence, $T_2(t) \geq T_3(t)$. Since $\vec{z}_2 \preceq \vec{z}_3$, it follows from Lemma 8 that $T_2(t) \leq T_3(t)$, and we conclude that $T_2(t) = T_3(t)$. Also, $\vec{z}_1 \succeq \vec{z}_4$, and it follows from Lemma 8 that $T_1(t) \geq T_4(t)$. Finally, we have that $T_1(t) + T_2(t) \geq T_3(t) + T_4(t), \forall t$. ■

This ends the proof of Theorem 1.

REFERENCES

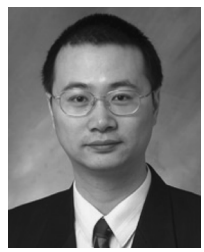
[1] L. Georgiadis, M. J. Neely, and L. Tassiulas, "Resource allocation and cross-layer control in wireless networks," *Found. Trends Netw.*, vol. 1, no. 1, pp. 1–144, 2006.

[2] X. Lin, N. B. Shroff, and R. Srikant, "A tutorial on cross-layer optimization in wireless networks," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 8, pp. 1452–1463, Aug. 2006.

[3] E. Modiano, D. Shah, and G. Zussman, "Maximizing throughput in wireless networks via gossiping," in *Proc. ACM SIGMETRICS*, 2006, pp. 27–38.
[4] S. Sanghavi, L. Bui, and R. Srikant, "Distributed link scheduling with constant overhead," in *Proc. ACM SIGMETRICS*, 2007, pp. 313–324.
[5] X. Lin and N. B. Shroff, "The impact of imperfect scheduling on cross-layer congestion control in wireless networks," *IEEE/ACM Trans. Netw.*, vol. 14, no. 2, pp. 302–315, Apr. 2006.
[6] P. Chaporkar, K. Kar, and S. Sarkar, "Throughput guarantees through maximal scheduling in wireless networks," *IEEE Trans. Inf. Theory*, vol. 54, no. 2, pp. 572–594, Feb. 2008.
[7] C. Joo, X. Lin, and N. B. Shroff, "Understanding the capacity region of the greedy maximal scheduling algorithm in multi-hop wireless networks," *IEEE/ACM Trans. Netw.*, vol. 17, no. 4, pp. 1132–1145, Aug. 2009.
[8] A. Gupta, X. Lin, and R. Srikant, "Low-complexity distributed scheduling algorithms for wireless networks," *IEEE/ACM Trans. Netw.*, vol. 17, no. 6, pp. 1846–1859, Dec. 2009.
[9] L. Bui, R. Srikant, and A. L. Stolyar, "Novel architectures and algorithms for delay reduction in back-pressure scheduling and routing," in *Proc. IEEE INFOCOM Mini-Conf.*, 2009, pp. 2936–2940.
[10] P.-K. Huang and X. Lin, "The end-to-end delay performance of a class of wireless scheduling algorithms," in *Proc. Allerton Conf. Commun., Control, Comput.*, 2010, pp. 951–952.
[11] L. Huang and M. J. Neely, "Delay reduction via Lagrange multipliers in stochastic network optimization," in *Proc. WiOpt*, 2009, pp. 1–10.
[12] M. J. Neely, "Delay-based network utility maximization," in *Proc. IEEE INFOCOM*, 2010, pp. 1–9.
[13] M. J. Neely, "Delay analysis for maximal scheduling in wireless networks with bursty traffic," in *Proc. IEEE INFOCOM*, 2008, pp. 6–10.
[14] L. B. Le, K. Jagannathan, and E. Modiano, "Delay analysis of maximum weight scheduling in wireless ad hoc networks," in *Proc. IEEE CISS*, Baltimore, MD, Mar. 2009, pp. 389–394.
[15] G. R. Gupta and N. B. Shroff, "Delay analysis for multi-hop wireless networks," in *Proc. IEEE INFOCOM*, 2009, pp. 2356–2364.
[16] L. Huang and M. J. Neely, "Delay efficient scheduling via redundant constraints in multihop networks," in *Proc. WiOpt*, 2010, pp. 142–151.
[17] L. B. Le, E. Modiano, and N. B. Shroff, "Optimal control of wireless networks with finite buffers," in *Proc. IEEE INFOCOM*, 2010, pp. 1–9.
[18] P. Jayachandran and M. Andrews, "Minimizing end-to-end delay in wireless networks using a coordinated EDF schedule," in *Proc. IEEE INFOCOM*, 2010, pp. 1–9.
[19] M. Xie and M. Haenggi, "Towards an end-to-end delay analysis of wireless multihop networks," *Ad Hoc Netw.*, vol. 7, pp. 849–861, Jul. 2009.
[20] S. Jagabathula and D. Shah, "Optimal delay scheduling in networks with arbitrary constraints," in *Proc. ACM SIGMETRICS*, Jun. 2008, pp. 395–406.
[21] B. Hajek and G. Sasaki, "Link scheduling in polynomial time," *IEEE Trans. Inf. Theory*, vol. 34, no. 5, pp. 910–917, Sep. 1988.
[22] S. Sarkar and L. Tassiulas, "End-to-end bandwidth guarantees through fair local spectrum share in wireless ad-hoc networks," in *Proc. IEEE CDC*, Maui, HI, Dec. 2003, vol. 1, pp. 564–569.
[23] F. P. Kelly, A. Maulloo, and D. Tan, "Rate control in communication networks: Shadow prices, proportional fairness and stability," *J. Oper. Res. Soc.*, vol. 49, pp. 237–252, 1998.
[24] A. L. Stolyar, "Large number of queues in tandem: Scaling properties under back-pressure algorithm," Bell Labs Tech. Memo, Oct. 2009.
[25] S. H. Low and D. E. Lapsley, "Optimization flow control—I: Basic algorithm and convergence," *IEEE/ACM Trans. Netw.*, vol. 7, no. 6, pp. 861–874, Dec. 1999.
[26] S. Shakkottai and R. Srikant, "Network optimization and control," *Found. Trends Netw.*, vol. 2, no. 3, pp. 271–379, 2007.
[27] P.-K. Huang, X. Lin, and C.-C. Wang, "A low-complexity congestion control and scheduling algorithm for multihop wireless networks with order-optimal per-flow delay," Purdue Univ., West Lafayette, IN, Tech. Rep., 2012 [Online]. Available: http://docs.lib.purdue.edu/ecetr/433/
[28] M. Shaked and J. G. Shanthikumar, *Stochastic Orders and Their Applications*. New York: Academic, 1994.
[29] A. Muller and D. Stoyan, *Comparison Methods for Stochastic Models and Risks*. New York: Wiley, 2002.
[30] F. Baccelli and Z. Liu, "Comparison properties of stochastic decision free petri nets," *IEEE Trans. Autom. Control*, vol. 37, no. 12, pp. 1905–1920, Dec. 1992.
[31] M. Schwartz, *Telecommunication Networks: Protocols, Modeling, and Analysis*. Reading, MA: Addison-Wesley, 1987.
[32] S. Athuraliya and S. H. Low, "Optimization flow control—II: Implementation," Univ. Melbourne, Melbourne, Australia, Tech. rep., 2000 [Online]. Available: http://netlab.caltech.edu/publications.php

Po-Kai Huang (M'12) received the B.S. and M.S. degrees in electrical and computer engineering from National Tsing Hua University, Taiwan, in 2004 and 2006, respectively, and has been pursuing the Ph.D. degree in electrical and computer engineering at Purdue University, West Lafayette, IN, since 2008.

His research interests are in the area of wireless networks including wireless cross-layer control, resource allocation, and delay performance analysis.



Xiaojun Lin (S'02–M'05) received the B.S. degree in electronics and information systems from Zhongshan University, Guangzhou, China, in 1994, and the M.S. and Ph.D. degrees in electrical and computer engineering from Purdue University, West Lafayette, IN, in 2000 and 2005, respectively.

He is currently an Associate Professor of electrical and computer engineering with Purdue University. He is currently serving as an Area Editor for *Computer Networks*, and has served as a Guest Editor for *Ad Hoc Networks*. His research interests are in the analysis, control, and optimization of wireless and wireline communication networks.

Dr. Lin was the Workshop Co-Chair for IEEE GLOBECOM 2007, the Panel Co-Chair for WICON 2008, the TPC Co-Chair for ACM MobiHoc 2009, and the Mini-Conference Co-Chair for IEEE INFOCOM 2012. He received the IEEE INFOCOM 2008 Best Paper Award and the 2005 Best Paper of the Year Award from the *Journal of Communications and Networks*. His paper was also one of two runner-up papers for the Best Paper Award at IEEE INFOCOM 2005. He received the NSF CAREER Award in 2007.

Chih-Chun Wang (M'06) received the B.E. degree from National Taiwan University, Taipei, Taiwan, in 1999, and the M.S. and Ph.D. degrees from Princeton University, Princeton, NJ, in 2002 and 2005, respectively, all in electrical engineering.

He is currently an Associate Professor with the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN. He worked with Comtrend Corporation, Taipei, Taiwan, as a Design Engineer in 2000 and spent the summer of 2004 with Flarion Technologies, Bedminster, NJ. In 2005, he held a Post-Doctoral Researcher position with the Department of Electrical Engineering, Princeton University. He joined Purdue University as an Assistant Professor in 2006. His current research interests are in the graph-theoretic and algorithmic analysis of iterative decoding and of network coding. Other research interests fall in the general areas of networking, optimal control, information theory, detection theory, and coding theory.

Dr. Wang received the National Science Foundation Faculty Early Career Development (CAREER) Award in 2009.