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

Po-Kai Huang, Xiaojun Lin, and Chih-Chun Wang
School of Electrical and Computer Engineering
Purdue University, West Lafayette, IN, USA
Email: {huang113, linx, chihw}@purdue.edu

***Abstract*—We consider the problem of designing a joint congestion control and scheduling algorithm for multihop wireless networks. The goal is to maximize the total utility and achieve low end-to-end delay simultaneously. Assume that there are M flows inside the network, and each flow m has a fixed route with H_m hops. Further, the network operates under the one-hop interference constraint. We develop a new congestion control and scheduling algorithm that combines a window-based flow control algorithm and a new distributed rate-based scheduling algorithm. 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 proposed algorithm can guarantee that each flow m achieves throughput no smaller than $r_m(1-\epsilon)(1-\epsilon_m)$, where the total utility of the rate allocation vector $\vec{r} = [r_m]$ is no smaller than the total utility of any rate vector within half of the capacity region. Furthermore, the end-to-end delay of flow m can be upper bounded by $H_m/(r_m(1-\epsilon)\epsilon_m)$. 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. To the best of our knowledge, this is the first fully-distributed joint congestion-control and scheduling algorithm that can guarantee order-optimal per-flow end-to-end delay and utilize close-to-half of the system capacity under the one-hop interference constraint. The throughput and delay bounds are proved by a novel stochastic dominance approach, which could be of independent value and be extended to general interference constraints. Our algorithm can be easily implemented in practice with a low per-node complexity that does not increase with the network size.**

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 non-decreasing 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]. Further, significant progresses have been made in designing distributed scheduling algorithms with provable throughput and lower complexity than the back-pressure algorithm [3]–[9]. However, most of the existing works on joint congestion control and scheduling have only considered the throughput performance metric and 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 very important as well because practical congestion control protocols need to set retransmission time-out 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.

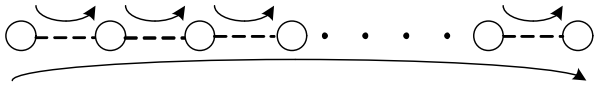


Fig. 1. A wireless network with linear topology.

There are two major issues on the delay-performance of the back-pressure algorithm. Firstly, 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 [10]. Consider a long flow traverses a fixed route with H hops. For each link that the long flow traverses, there is a short flow that competes with the long flow as shown in Fig. 1. Under the back-pressure algorithm, for the long flow to be scheduled on a link, 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 $Hq, (H-1)q, \cdots, q$, which leads to the total end-to-end backlog of order $\mathcal{O}(H^2)$. It then follows from Little's law that the end-to-end delay will also be of the 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 $\mathcal{O}(H)$. Secondly, 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 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 under the one-hop interference model, 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 are to tightly control the number of packets inside the network and to schedule the packets by a rate-based scheduling algorithm rather than a queue-length-based algorithm. The key difficulty in analyzing the end-to-end throughput and delay for window-based flow control 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 rate allocation vector $\vec{r} = [r_m]$ is no smaller than the total utility of any rate vector within half of the system capacity region. Further, 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. Further, 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 utilize close-to-half of the system capacity under the one-hop interference constraint.

Recently, there have been a number of papers that quantify the delay performance of wireless networks with or without congestion control [10], [12]–[21]. In [10], [12], the authors propose methods to reduce the delay of the back-pressure algorithm. The algorithm proposed in [10] is a shadow back-pressure algorithm, which maintains a single 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 [10]). In [12], the authors propose another mechanism to decouple the control signal from the real queues by injecting dummy packets into the queues. However, until the algorithm converges, it will be difficult to know the correct number of dummy packets. Hence, the backlog at the transient period is still difficult to control. In contrast, our proposed algorithm tightly controls the end-to-end backlog at all times.

Our result is also different from other works in providing a linear-order per-flow delay bound. First, [13], [14] only prove delay bounds for single-hop flows rather than multihop flows. Second, [15]–[17] consider the delay among all the flows rather than the per-flow delay. Similarly, the results in [18] can be used to construct a bound on the delay averaged over all flows. However, it is still not a per-flow delay bound. Third, a per-flow delay bound is provided in [19], but the bound scales with the size of the network. Fourth, a single flow end-to-end delay analysis is given in [20] 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 [21], where the authors provide a per-flow delay bound that scales with the number of hops without considering congestion control. However, the algorithm in [21] has a factor 5 loss of throughput under the one-hop interference constraint, and the algorithm is much 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 and can utilize close-to-half of the capacity.

Our contributions can be summarized as follows.

- We provide a new joint congestion control and scheduling algorithm that can utilize close to half of the system capacity, i.e., a factor 2 loss of throughput, and guarantee a per-flow expected delay upper bound that increases linearly with the number of hops.
- The congestion control algorithm is based on window flow control. For each flow, this method deterministically bounds the end-to-end backlog within the network and prevents buffer overflows. Further, 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 can be applied to general interference constraints [22].

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.

[1]A related delay bound can also be shown for the scheduling algorithm without congestion control, which will appear as a one-page abstract in [11]. However, the delay for joint congestion control and scheduling in this paper is more difficult due to the closed-loop feedback.

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. 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 end-points of a common link. Two links are one-hop neighbors if they share a common node. For each node v, let $N(v)$ denote the set of links that connect to the one-hop neighbors of node v.

We assume a time-slotted wireless system, where packet transmissions occur within time slots of unit length. The capacity of any link is normalized to 1. We say two links interfere with each other, if they can not transmit data at the same time slot. For ease of exposition, we assume that a link will interfere with all its one-hop neighboring links (i.e., one-hop interference constraint). This constraint has been used to model wireless networks in [5], [6], [23], [24]. (We note that our approach can be extended to a more general interference model, where an interference set for each link is given, and a link interferes with any other link in its interference set.) In our system, there are M flows, 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 $U_m(R_m)$ [25], which reflects the "satisfactory level" of user m with injection rate R_m. We assume that $U_m(\cdot)$ is strictly concave, non-decreasing, 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:

$$\max_{R_m \geq 0} \sum_m U_m(R_m), \ \vec{R} \in \Omega. \tag{1}$$

The exact capacity region Ω is often difficult to characterize. It is also well known that, under the one-hop interference model, $\Psi_0/2 \subseteq \Omega \subseteq \Psi_0$, where

$$\Psi_0 = \left\{ \vec{R} \left| \sum_{\ell \in N(v)} \sum_m L_m^\ell R_m \leq 1, \text{ for all nodes } v \right. \right\}. \tag{2}$$

Note that equation (2) simply states that, for each node v, the total load must be no larger than 1. This is because a wireless node can only communicate with one other node at a time slot. In Section III, we will describe how we utilize the relationship between Ω and Ψ_0 to approximately solve problem (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 the per-flow average delay.

III. JOINT CONGESTION CONTROL AND SCHEDULING ALGORITHM

As we discussed in Section I, there are many approaches available in the literature to solve problem (1), and most of them do not consider delay performance. A typical optimal solution can be obtained by a duality approach which results into the back-pressure algorithm and a congestion-control component at the source node [1], [2]. Further, 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 provable good throughput performance [3]–[9]. 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 there are evidence that, under certain cases, the back-pressure can have poor delay performance [10], [26]. In this paper, we will use a window-based flow control algorithm and a rate-based scheduling algorithm that are very different from back-pressure. Our solution strategy is to first approximately solve problem (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 problem (1). Since the true capacity region Ω is of a complex form, instead of solving problem (1) directly, we solve the following optimization problem: (we will make precise the relationship between optimization problems (1) and (3) in Section III-D.)

$$\max_{r_m \geq 0} \sum_m U_m(r_m), \ \vec{r} \in \Psi_0/2. \tag{3}$$

Note that the optimization problem (3) is very similar to the standard convex-optimization problem in wireline network with linear constraints [27], [28]. Therefore, it is easy to apply the approaches in [27], [28] to problem (3). We will not elaborate on all the possible approaches to solve problem (3). Instead, we only present one well-known distributed solution. Specifically, associate a Lagrange multiplier (the dual variable) $\lambda_v \geq 0$ to each constraint in (3). Let $c_{vm} = 2$ if v is an intermediate node of flow m, $c_{vm} = 1$ if v is the source node or destination node of flow m, and $c_{vm} = 0$ otherwise. The objective function of the dual problem of (3) becomes:

$$D(\vec{\lambda}) := \max_{r_m \geq 0} \sum_m U_m(r_m) - \sum_v \lambda_v \left(\sum_m r_m c_{vm} - \frac{1}{2} \right).$$

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:

$$r_m(t) = U_m'^{-1}(\sum_{v \in m} \lambda_v(t) c_{vm}),$$

where $v \in m$ indicates that node v is on flow m's route.

2) Each node updates the dual variables by equation:

$$\lambda_v(t+1) = \left[\lambda_v(t) + \gamma_v \left(\sum_{m:v \in m} r_m(t) c_{vm} - \frac{1}{2}\right)\right]^+,$$

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

Using similar techniques as [27], one can show that as long as γ_v are sufficiently small, the above algorithm will converge to the optimal solution of (3). Note that as in [27], this algorithm requires passing λ_v 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, and a 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 back-pressure is difficult to quantify and may be poor [10], [26]. 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 [9]. Each time slot consists of a scheduling slot and a transmission slot. The links that are to be scheduled are selected in the scheduling slot, and the selected links transmit their packets in the transmission slot. The scheduling slot is further divided into F mini-slots. Let $a_\ell(t) = \sum_m L_m^\ell r_m(t)$, which is the sum of the virtual rate over link ℓ, and let $x_\ell(t) = \max\left(\sum_{e \in N(b(\ell))} a_e(t), \sum_{e \in N(d(\ell))} a_e(t)\right)$.

Rate-based Scheduling Algorithm: At each time t,

1) Each link ℓ first computes $P_\ell = \log(F) a_\ell(t)/x_\ell(t)$.
2) Each link then randomly picks the number of backoff mini-slots (B) with distribution: $P\{B = F+1\} = e^{-P_\ell}$ and $P\{B = f\} = e^{-P_\ell \frac{f-1}{F}} - e^{-P_\ell \frac{f}{F}}, f = 1, \cdots, F$. 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, 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) If a link begins transmission, it will randomly choose a passing flow m to serve with probability $r_m(t)/a_\ell(t)$.

Note that this scheduling algorithm only uses virtual rates to compute P_ℓ, which is different from the queue-length-based algorithm studied in [9]. 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 $\frac{r_m(t)}{\sum_{\{m:Q_{m\ell}(t)>0\}} L_m^\ell r_m(t)}$, 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 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 [18]. 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, each ACK will also go through the entire route hop-by-hop in the reverse direction to reach the source node. In Section V, we will discuss how this can be achieved by piggy-backing 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 subsection, we will present the main steps of the analysis and the bounds on the throughput and delay of the

above proposed scheme. Due to space constraints, we will omit some proofs in this section and in Section IV. Details of these proofs can be found in our online technical report [22]. 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:

$$\max_{r_m \geq 0} \sum_m U_m(r_m), \ \vec{r} \in \Omega/2. \tag{4}$$

Lemma 1: $\sum_m U_m(r_m^*) \geq \sum_m U_m(r_m'^*)$.

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 $\Omega/2$. Further, we can show the following property of the scheduling algorithm.

Lemma 2: For any $\epsilon \in (0,1)$, if flow m passes through link ℓ, we can choose F large enough such that the probability that link ℓ will schedule flow m at time t is no smaller than $r_m(t)(1-\epsilon)/(2x_\ell(t))$.

This lemma implies that the scheduling algorithm will start serving flow-m packets even before the virtual-rate computation algorithm converges. After the virtual-rate computation algorithm converges[2], 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 $x_\ell(t) \leq 1/2$. It then follows from Lemma 2 that every hop along the path of flow m will serve flow m with probability no smaller than $r_m^*(1-\epsilon)$ independent across time slots. This observation allows us to isolate flow m out of the network and view the flow-m packets as passing through a virtual tandem network of H_m queues. 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, the end-to-end packet delay will also be large. 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. Further, the exact statistics of the correlation among links is hard to characterize because of the interference constraint.

Proposition 3: After the virtual-rate computation algorithm converges, for each flow m, our congestion control and scheduling algorithm can achieve average throughput $\frac{r_m^*(1-\epsilon)W_m}{W_m+H_m-1}$, where W_m is the window size of flow m, and ϵ is chosen as in Lemma 2. Moreover, the average delay is upper bounded by $\frac{W_m+H_m-1}{r_m^*(1-\epsilon)}$.

This proposition has the following two implications. First, for any $\epsilon_m \in (0,1)$, let W_m be the smallest positive integer such that $W_m > (H_m-1)(1-\epsilon_m)/\epsilon_m$. This implies that $W_m/(W_m+H_m-1) > (1-\epsilon_m)$. It then follows from Proposition 3 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 half of the capacity region. Second, since W_m is the smallest positive integer such that $W_m > (H_m-1)(1-\epsilon_m)/\epsilon_m$, we have that $W_m \leq (H_m-1)(1-\epsilon_m)/\epsilon_m + 1$. Thus,

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

It then follows from Proposition 3 that the delay will be upper bounded by $H_m/(r_m^*(1-\epsilon)\epsilon_m)$. As discussed in Section I, this implies that our per-flow delay upper-bound is order optimal.

IV. Proof of Proposition 3

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. Furthermore, for ease of presentation, we assume that there is a feedback channel from the destination node to the source node for the window 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 and H_m. We can then model this flow as a 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. By the discussion after Lemma 2, we know that

$$\mu_\ell \geq \mu \triangleq r_m^*(1-\epsilon), \tag{5}$$

where μ_ℓ is the probability that link ℓ will serve a flow-m packet. Since we use window 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 the 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 i^{th} 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 [22].

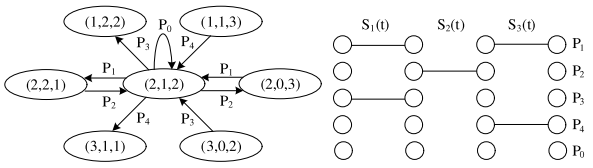


Fig. 2. Left: The incoming and outgoing transitions to and from state $(2,1,2)$. Right: The distribution of the random schedule vector $\vec{S}(t)$.

Fig. 2 illustrates an example of the MC for a 3-hop closed tandem network with 5 packets. If the MC is in steady state,

[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 fo r_m^*. For ease of exposition, we do not pursued this direction further in this paper.

we can compute the actual throughput R_m as follows:

$$R_m = \mu_H[1 - P\{Q_H = 0\}]. \quad (6)$$

If we can compute the throughput, 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, link 1 and link 2 will never be scheduled together due to interference, and there is a chance that link 1 and link 3 will be served together. Further, the exact statistics of such correlation is hard to characterize. All that we know (from Lemma 2) is a lower-bounded marginal probability μ 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 to derive the throughput lower-bound.

A. Overview of the Approach

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 flow control and window size W. For each system, the random schedule vector is always i.i.d. across time. Further, for different systems, the initial condition for the packet placement is the same. However, within a time slot, the distribution of the schedule vector is different depending on the system that we refer to. Since the only difference between two systems is the distribution of the schedule, 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 i^{th} schedule vector, $x_{ij} = 1$ if link j is activated under the i^{th} 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-C how to treat the case when property (7) is not satisfied.)

$$\sum_\ell \mu'_\ell = \sum_\ell \sum_{i \in A_\ell} p'_i \leq 1, \text{and } \mu'_\ell \geq \mu'. \quad (7)$$

Recall that the key difficulty of analyzing the system is the correlation of the services among links. We now introduce a splitting procedure that convert 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 more than 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:

$$P\{\vec{S}^{(2)}(t) = \vec{x}_1 - \vec{e}_\ell\} = p'_1, \quad P\{\vec{S}^{(2)}(t) = \vec{e}_\ell\} = p'_1,$$
$$P\{\vec{S}^{(2)}(t) = \vec{x}_0\} = p'_0 - p'_1, \quad P\{\vec{S}^{(2)}(t) = \vec{x}_i\} = p'_i, i \geq 2.$$

Note that the schedule $\vec{x}_1$ is now splitted 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

$$\begin{aligned} p'_0 - p'_1 &= 1 - \textstyle\sum_{i \neq 0} p'_i - p'_1 \geq 1 - \sum_{i \neq 0} |\vec{x}_i| p'_i \\ &= 1 - \textstyle\sum_\ell \sum_{i \in A_\ell} p'_i \geq 0, \end{aligned}$$

where 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 relationship between $\vec{S}^{(1)}(\cdot)$ and $\vec{S}^{(2)}(\cdot)$ is as follows.

Theorem 4: 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).

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 4 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 $P\{\vec{S}^{(3)}(t) = \vec{x}_0\} = 1 - \sum_\ell \mu_\ell$. By applying Theorem 4 in every iteration, we then have that $T(\vec{S}^{(1)}(\cdot)) \geq T(\vec{S}^{(3)}(\cdot))$. As we will see in Section IV-C, the lower bound of $T(\vec{S}^{(3)}(\cdot))$ can be more easily calculated.

B. Proof of Theorem 4

To prove Theorem 4, we use a specific stochastic ordering called supermodular ordering. We review the basic definitions used in our proof, and readers are referred to [29], [30] for other definitions and basic properties of stochastic ordering.

Definition 5: **(Supermodular Function)** A function $\phi(\vec{x}) : \mathcal{R}^n \to \mathcal{R}$ is said to be supermodular if, for any n-dimensional vectors $\vec{x}_1, \vec{x}_2$, it satisfies that

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

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

Definition 6: **(Supermodular Ordering)** Let $\mathcal{F}$ be the class of all supermodular functions from $\mathcal{R}^n$ 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 the supermodular order (denoted by $\vec{X} \leq_{sm} \vec{Y}$) if $E[\phi(\vec{X})] \leq E[\phi(\vec{Y})]$, for all $\phi \in \mathcal{F}$.

An important relationship between $\vec{S}^{(1)}$ and its split version $\vec{S}^{(2)}$ is the following. (The proof is available in [22].)

Lemma 7: 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 $\vec{S}^{(1)}(t) \geq_{sm} \vec{S}^{(2)}(t)$.

Sketch of the Proof of Theorem 4: Fix a packet placement at time 0. Under window 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. To prove Theorem 4, we first show that f is a supermodular function with respect to the schedule vector at time

1.[3] In other words, let $T_i(t) = f(\vec{z}_i, \vec{y}(2), \vec{y}(3), \cdots, \vec{y}(t))$, $i = 1, \cdots, 4$, where $\vec{z}_1 = \vec{z}_3 \vee \vec{z}_4$, $\vec{z}_2 = \vec{z}_3 \wedge \vec{z}_4$, and $\vec{y}(2), \vec{y}(3), \cdots, \vec{y}(t)$ are a sequence of deterministic schedules. We would like to show that

$$T_1(t) + T_2(t) \geq T_3(t) + T_4(t). \tag{9}$$

Notice that $\vec{z}_1 \succcurlyeq \vec{z}_i \succcurlyeq \vec{z}_2$, $i = 3, 4$, where $\succcurlyeq$ means componentwise larger. It is then intuitive that $T_1(t) \geq T_i(t) \geq T_2(t)$, $i = 3, 4$, because the throughput can only be higher if more services are provided. (Details can be found in [22].) The nontrivial result is the following:

Lemma 8: For any time t, $2T_2(t) + 1 \geq T_3(t) + T_4(t)$.

The intuition behind this lemma can be explained as follows. Suppose that the service discipline of each queue is FIFO (Notice that the service discipline does not change the total number of departing packets from Q_H). We label packets from 1 to W according to their distance to the destination and their position in the current queue. A packet has a smaller index if it is closer to the destination or it will be served earlier at the current queue. We will use Y_k, $k = 1, \cdots, 4$ when we refer to the dynamics of the system under the sequence of schedules $\vec{z}_k, \vec{y}(2), \cdots, \vec{y}(t)$. Our first observation is that if $\vec{z}_k \succcurlyeq \vec{z}_w$, then packet i in system Y_k may be ahead of packet i in system Y_w. However, because the rest of the schedules are the same for both systems, packet i (correspondingly packet 1) in system Y_k can never be ahead of packet $i-1$ (correspondingly packet W) in system Y_w. This observation can be used to show that the difference between $T_k(t)$ and $T_w(t)$ is at most 1. Second, because $\vec{z}_2 = \vec{z}_3 \wedge \vec{z}_4$, for any given link ℓ either $\vec{z}_3$ or $\vec{z}_4$ will have the same service as $\vec{z}_2$. Again, because the rest of the schedules are the same for all systems, we can then use this fact to show that even though a packet i in Y_2 may be behind packet i in Y_3 or Y_4, but never both. With these two observations, we can rigorously show that Lemma 8 holds. Readers can refer to [22] for details. Now, we can prove (9).

Proof of inequality (9): Prove by contradiction. Assume that $T_1(t) + T_2(t) < T_3(t) + T_4(t)$. From $T_3(t) + T_4(t) \leq 2T_2(t) + 1$, we know that $T_1(t) < T_2(t) + 1$. Since $T_1(t) \geq T_2(t)$, we have that $T_1(t) = T_2(t)$. It then follows from $T_1(t) \geq T_3(t) \geq T_2(t)$ that $T_3(t) = T_2(t)$. Similarly, we can show that $T_4(t) = T_2(t)$. Thus, $T_1(t)+T_2(t) = T_3(t)+T_4(t)$. This contradicts the assumption. ∎

Continue the proof of Theorem 4. By Lemma 7, $\vec{S}^{(1)}(1) \geq_{sm} \vec{S}^{(2)}(1)$. We can then use conditional expectation and the fact that f is a supermodular function with respect to the first variable to show that $E[f(\vec{S}^{(1)}(1), \cdots, \vec{S}^{(1)}(t))] \geq E[f(\vec{S}^{(2)}(1), \vec{S}^{(1)}(2) \cdots, \vec{S}^{(1)}(t))]$. Similarly, we can also show that $E[f(\vec{S}^{(2)}(1), \vec{S}^{(1)}(2), \vec{S}^{(1)}(3) \cdots, \vec{S}^{(1)}(t))] \geq E[f(\vec{S}^{(2)}(1), \vec{S}^{(2)}(2), \vec{S}^{(1)}(3) \cdots, \vec{S}^{(1)}(t))]$ because, by conditional expectation, both sides have the same packet placement at the beginning of time 2, and the only difference is the schedule at time 2. Iteratively, we then prove that

$$E[f(\vec{S}^{(1)}(1), \cdots, \vec{S}^{(1)}(t))] \geq E[f(\vec{S}^{(2)}(1), \cdots, \vec{S}^{(2)}(t))].$$

By ergodic theory, $\lim_{t\to\infty} E[f(\vec{S}^{(i)}(1), \cdots, \vec{S}^{(i)}(t))]/t = T(\vec{S}^{(i)})$, $i = 1, 2$. (For details, please refer to [22].) Thus, $T(\vec{S}^{(1)}) \geq T(\vec{S}^{(2)})$. Finally, it is easy to see that $\vec{S}^{(2)}(\cdot)$ has property (7). This ends the proof of Theorem 4. ∎

C. Throughput Lower Bound and Delay Upper Bound

As discussed in Section IV-A, we can use Theorem 4 to show that $T(\vec{S}^{(1)}(\cdot)) \geq T(\vec{S}^{(3)}(\cdot))$. Consider another system $\vec{S}^{(4)}(\cdot)$. The distribution of $\vec{S}^{(4)}(t)$ is $P\{\vec{S}^{(4)}(t) = \vec{e}_\ell\} = \mu', \ell = 1, \cdots, H$ and $P\{\vec{S}^{(4)}(t) = \vec{x}_0\} = 1 - H\mu'$. It can be verified that $T(\vec{S}^{(3)}(\cdot)) \geq T(\vec{S}^{(4)}(\cdot))$ [22]. This result is intuitive because, for every link ℓ, $P\{\vec{S}^{(3)}(t) = \vec{e}_\ell\} \geq P\{\vec{S}^{(4)}(t) = \vec{e}_\ell\}$. Thus, the throughput of system $\vec{S}^{(3)}(\cdot)$ should be no smaller than the throughput of system $\vec{S}^{(4)}(\cdot)$. The throughput of system $\vec{S}^{(4)}(\cdot)$ has a closed-form solution, i.e., $T(\vec{S}^{(4)}(\cdot)) = \mu' W/(W + H - 1)$ [22]. Therefore, we have a throughput lower bound for system $\vec{S}^{(3)}(\cdot)$. Note that the MC for system $\vec{S}^{(4)}(\cdot)$ is similar to that of a closed tandem $M/M/1$ queues with identical service rates [31].

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 $p'_i = p_i/(\sum_\ell \mu_\ell)$, 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

$$\begin{aligned} \textstyle\sum_\ell \mu'_\ell &= \textstyle\sum_\ell \sum_{i\in A_\ell} p'_i = \sum_\ell \sum_{i\in A_\ell} p_i/(\sum_j \mu_j) \\ &= \textstyle\sum_\ell \mu_\ell/(\sum_j \mu_j) \leq 1. \end{aligned}$$

We also have from (5) that $\mu'_\ell = \mu_\ell/\sum_i \mu_i \geq \mu/\sum_i \mu_i \triangleq \mu'$. Thus, system $\vec{S}^{(1)}(\cdot)$ has property (7). The relationship between $T(\vec{S}(\cdot))$ and $T(\vec{S}^{(1)}(\cdot))$ is $(\sum_\ell \mu_\ell)T(\vec{S}^{(1)}(\cdot)) = T(\vec{S}(\cdot))$ [22]. This can be shown by noting that MC for $\vec{S}(\cdot)$ and $\vec{S}^{(1)}(\cdot)$ have the same steady state distribution. It then follows from $T(\vec{S}^{(1)}(\cdot)) \geq T(\vec{S}^{(4)}(\cdot)) = \mu' W/(W + H - 1)$ that

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

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

$$D(\vec{S}(\cdot)) \leq (W + H - 1)/\mu.$$

This finishes the proof of Proposition 3.

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.

[3] Readers may wonder why not show directly that f is a supermodular function with respect to the entire sequences. Unfortunately, we can construct counter examples to show that this is not true [22]. However, as readers will see towards the end of Section IV-B, for Theorem 4 to hold, it is sufficient to show that f is a supermodular function with respect to the first variable.

Immediately after a link transmits a flow m packet, the receiving node will respond with an acknowledgement, which piggy-backs an ACK for flow m that it has received from the destination in the past. With this mechanism, each link can be modelled 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 modelled 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 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 only difference is that the number of hops is changed from H_m to $2H_m$, which does not affect our order-optimal delay result.

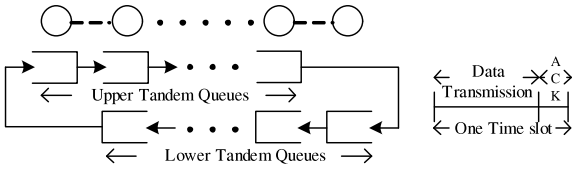


Fig. 3. Upper (resp. Lower) tandem queues store packets (resp. ACKs).

Second, in the virtual-rate computation algorithm, each node needs to collect the virtual-rate of each flow that passes through itself, and each source node needs to collect the sum of the dual variables along its route. In practice, such information exchange can be easily achieved by piggy-backing the virtual-rate information on each packet sent by the source node and piggy-backing the sum of the dual-variables on each ACK sent by the destination node. Note that although the virtual rates and dual variables are updated asynchronously, our window-based flow control algorithm guarantees an upper bound on the expected delay of such information exchange. Hence, we expect that the virtual-rate computation algorithm will still converges with suitable choices of the step sizes [27], [32].

Finally, in the scheduling algorithm, each transmitting node must collect the virtual-rates around the receiving node. Again, this information can be piggy-backed on data packets and ACKs. We note that the transmitter 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. This complexity is significantly lower than the algorithm in [21], which requires $\mathcal{O}(N)$ per-node operations.

VI. Simulation Results

We simulate our proposed algorithm using the linear topology in Fig. 1 with H links under the one-hop interference constraint. The simulation results using the grid topology can be found in [22]. We use the improved version of our scheduling algorithm as discussed in Section III-B and set the number of backoff mini-slots $F = 32$. The window sizes of each short flow and the long flow are 2 and $2H$, respectively. 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 the same. This will help us to observe the relationship between average delay and the number of hops at a fixed throughput.

We first compare the performance of our proposed algorithm with the standard back-pressure algorithm (for different step sizes) and the *shadow back-pressure* algorithm proposed in [10]. 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 back-pressure algorithm increases quadratically with the number of hops. Therefore, our algorithm outperforms the back-pressure algorithm in the delay performance when $H \geq 7$ even though the back-pressure also 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 back-pressure and SBP. We can see that the throughput of our distributed algorithm is indeed more than half of the centralized and high-complexity back-pressure algorithm. Another interesting observation is that when the step size is large (BP-16), the throughput differs significantly from those with smaller step sizes. This indicates that the delay reduction (in Fig. 4(a)) of the back-pressure 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. Finally, note that although the average delay curve of shadow back-pressure algorithm also shows linear-scaling, it requires roughly 10000 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). Therefore, the average delay of the first 1000 outgoing packets of the long flow is nearly 1000. In contrast, because of the window-based flow control, the total queue backlog of our algorithm is consistently the lowest throughout the simulation at all time (even at the transient period).

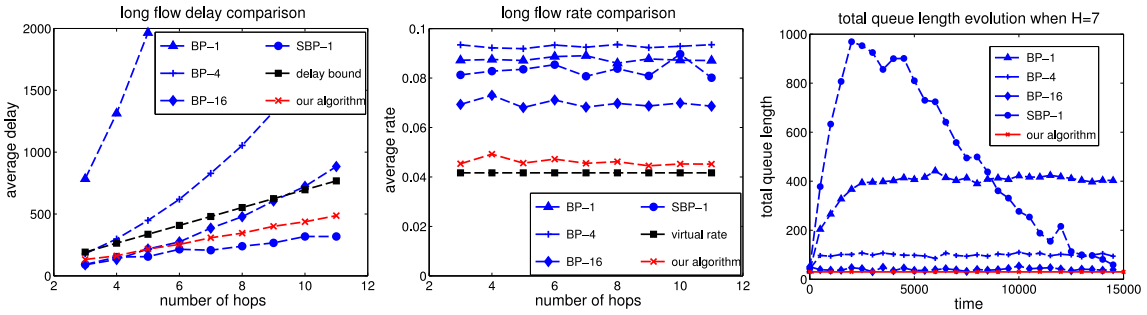
In Table I, we demonstrate the per-flow controllability of our scheme. We let the window size of the long flow be $2H$, $H = 7$, and vary the window size of the short flows. As shown in Table I, the performance of the long flow does not change when the window size of the short flows changes.

TABLE I

short flow window size	2	4	6	8
long flow delay	290.37	291.53	290.97	294.62
long flow throughput	0.0480	0.0475	0.0478	0.0478

VII. Conclusion

In this paper, we propose a low-complexity and distributed algorithm for joint congestion control and scheduling in multihop wireless networks under the one-hop interference constraint. The main ideas of the proposed algorithm are to control the congestion with window-based flow control and



(a) (b) (c)

Fig. 4. BP-α represents the back-pressure algorithm with step size α. SBP-α represents the shadow back-pressure algorithm with step size α.

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 [9]. We prove that our congestion control and scheduling algorithm can utilize nearly half 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 compare the throughput of our system with another system, which we can compute the exact throughput. We then use Little's law to derive a per-flow delay upper-bound. The methodology in this paper can also be extended to more general interference models defined by interference sets (see [22] for details). In our future work, we will study how to extend this novel technique to the case with dynamic routing.

Acknowledgement: This work has been partially supported by the NSF through grants CNS-0721477, CNS-0643145, CNS-0813000 and the Purdue Research Foundation.

REFERENCES

[1] L. Georgiadis, M. J. Neely, and L. Tassiulas, "Resource Allocation and Cross-Layer Control in Wireless Networks," *Foundations and Trends in Networking*, 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 Journal on Selected Areas in Communications*, vol. 24, no. 8, Aug. 2006.

[3] E. Modiano, D. Shah, and G. Zussman, "Maximizing Throughput in Wireless Networks via Gossiping," in *ACM SIGMETRICS*, 2006.

[4] S. Sanghavi, L. Bui, and R. Srikant, "Distributed Link Scheduling with Constant Overhead," in *ACM SIGMETRICS*, 2007.

[5] X. Lin and N. B. Shroff, "The Impact of Imperfect Scheduling on Cross-Layer Congestion Control in Wireless Networks," *IEEE/ACM Trans. on Networking*, vol. 14, no. 2, pp. 302–315, 2006.

[6] X. Lin and S. Rasool, "Constant-Time Distributed Scheduling Policies for Ad Hoc Wireless Networks," *IEEE Trans. on Automatic Control*, vol. 54, no. 2, pp. 231–242, February 2009.

[7] P. Charporkar, K. Kar, and S. Sarkar, "Throughput Guarantees Through Maximal Scheduling in Wireless Networks," *IEEE Trans. on Inf. Theory*, vol. 54, no. 2, pp. 572–594, Feb. 2008.

[8] C. Joo, X. Lin, and N. B. Shroff, "Understanding the Capacity Region of the Greedy Maximal Scheduling Algorithm in Multi-hop Wireless Networks," in *IEEE INFOCOM*, 2008.

[9] A. Gupta, X. Lin, and R. Srikant, "Low-Complexity Distributed Scheduling Algorithms for Wireless Networks," *IEEE/ACM Trans. on Networking*, December 2009.

[10] L. Bui, R. Srikant, and A. L. Stolyar, "Novel Architectures and Algorithms for Delay Reduction in Back-pressure Scheduling and Routing," in *IEEE INFOCOM Mini-Conference*, 2009.

[11] P.-K. Huang and X. Lin, " The End-to-End Delay Performance of A Class of Wireless Scheduling Algorithms," in *Allerton Conference on Communication, Control, and Computing, invited talk*, 2010.

[12] L. Huang and M. J. Neely, "Delay Reduction via Lagrange Multipliers in Stochastic Network Optimization," in *WiOpt*, 2009.

[13] M. J. Neely, "Delay-Based Network Utility Maximization," in *IEEE INFOCOM*, 2010.

[14] ——, "Delay Analysis for Maximal Scheduling in Wireless Networks with Bursty Traffic," in *IEEE INFOCOM*, 2008.

[15] L. B. Le, K. Jagannathan, and E. Modiano, "Delay Analysis of Maximum Weight Scheduling in Wireless Ad Hoc Networks," in *IEEE CISS*, Baltimore, MD, March 2009.

[16] G. R. Gupta and N. B. Shroff, "Delay Analysis for Multi-hop Wireless Networks," in *IEEE INFOCOM*, 2009.

[17] L. Huang and M. J. Neely, "Delay Efficient Scheduling Via Redundant Constraints in Multihop Networks," in *WiOpt*, 2010.

[18] L. B. Le, E. Modiano, and N. B. Shroff, "Optimal Control of Wireless Networks with Finite Buffers," in *IEEE INFOCOM*, 2010.

[19] P. Jayachandran and M. Andrews, "Minimizing End-to-End Delay in Wireless Networks Using a Coordinated EDF Schedule," in *IEEE INFOCOM*, 2010.

[20] M. Xie and M. Haenggi, "Towards an End-to-End Delay Analysis of Wireless Multihop Networks," *Elsevier Ad Hoc Networks*, vol. 7, pp. 849–861, July 2009.

[21] S. Jagabathula and D. Shah, "Optimal Delay Scheduling in Networks with Arbitrary Constraints," in *ACM SIGMETRICS*, June 2008.

[22] 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," *Technical Report, Purdue University*, 2010. [Online]. Available: http://web.ics.purdue.edu/%7ehuang113/

[23] B. Hajek and G. Sasaki, "Link Scheduling in Polynomial Time," *IEEE Trans. on Inf. Theory*, vol. 34, no. 5, pp. 910–917, September 1988.

[24] S. Sarkar and L. Tassiulas, "End-to-end Bandwidth Guarantees Through Fair Local Spectrum Share in Wireless Ad-hoc Networks," in *Proc. of the IEEE CDC*, Maui, Hawaii, December 2003.

[25] F. P. Kelly, A. Maulloo, and D. Tan, "Rate Control in Communication Networks: Shadow Prices, Proportional Fairness and Stability," *Journal of the Operational Research Society*, vol. 49, pp. 237–252, 1998.

[26] A. Stolyar, "Large number of queues in tandem: Scaling properties under back-pressure algorithm," *Bell Labs Technical Memo*, October 2009.

[27] S. H. Low and D. E. Lapsley, "Optimization Flow Control–I: Basic Algorithm and Convergence," *IEEE/ACM Trans. on Networking*, vol. 7, no. 6, pp. 861–874, December 1999.

[28] S. Shakkottai and R. Srikant, *Network Optimization and Control*. Foundations and Trends in Networking, 2007.

[29] M. Shaked and J. G. Shanthikumar, *Stochastic Orders and Their Applications*. Academic Press, 1994.

[30] A. Muller and D. Stoyan, *Comparison Methods for Stochastic Models and Risks*. John Wiley & Son, 2002.

[31] M. Schwartz, *Telecommunication Networks: Protocols, Modeling, and Analysis*. Addison Wesley, 1987.

[32] S. Athuraliya and S. H. Low, "Optimization Flow Control–II: Implementation," *Technical report, University of Melbourne, http://netlab.caltech.edu/publications.php*, 2000.