Novel Architectures and Algorithms for Delay Reduction in Back-pressure Scheduling and Routing

Loc Bui
ECE Dept. and CSL
University of Illinois, USA
locbui@ifp.uiuc.edu

R. Srikant
ECE Dept. and CSL
University of Illinois, USA
rsrikant@illinois.edu

Alexander Stolyar
Bell Labs
Alcatel-Lucent, NJ, USA
stolyar@research.bell-labs.com

***Abstract*—The back-pressure algorithm is a well-known throughput-optimal algorithm. However, its delay performance may be quite poor even when the traffic load is not close to network capacity due to the following two reasons. First, each node has to maintain a separate queue for each commodity in the network, and only one queue is served at a time. Second, the back-pressure routing algorithm may route some packets along very long routes. In this paper, we present solutions to address both of the above issues, and hence, improve the delay performance of the back-pressure algorithm. One of the suggested solutions also decreases the complexity of the queueing data structures to be maintained at each node.**

I. Introduction

Resource allocation in wireless networks is complicated due to the shared nature of wireless medium. One particular allocation algorithm called the *back-pressure algorithm* which encompasses several layers of the protocol stack from MAC to routing was proposed by Tassiulas and Ephremides, in their seminal paper [1]. The back-pressure algorithm was shown to be *throughput-optimal*, i.e., it can support any arrival rate vector which is supportable by any other resource allocation algorithm. Recently, it was shown that the back-pressure algorithm can be combined with congestion control to fairly allocate resources among competing users in a wireless network [2]–[7], thus providing a complete resource allocation solution from the transport layer to the MAC layer. While such a combined algorithm can be used to perform a large variety of resource allocation tasks, in this paper, we will concentrate on its application to scheduling and routing.

Even though the back-pressure algorithm delivers maximum throughput by adapting itself to network conditions, there are several issues that have to be addressed before it can be widely deployed in practice. As stated in the original paper [1], the back-pressure algorithm requires centralized information and computation, and its computational complexity is too prohibitive for practice. Much progress has been made recently in easing the computational complexity and deriving decentralized heuristics. We refer the interested reader to [8], [9] and references within for some recent results along these lines. We do not consider complexity or decentralization issues in this paper; our proposed solutions can be approximated well by the solutions suggested in the above papers.

Besides complexity and decentralization issues which have received much attention recently, the back-pressure algorithm can also have poor delay performance. To understand that, we consider two different network scenarios: one in which the back-pressure algorithm is used to adaptively select a route for each packet, and the other in which a flow's route is chosen upon arrival by some standard multi-hop wireless network routing algorithm such as DSR or AODV and the back-pressure algorithm is simply used to schedule packets. We refer to the first case as *adaptive-routing* and the second case as *fixed-routing*, respectively.

We first discuss networks with fixed routing. It is easy to see that under the back-pressure algorithm, for a link to be scheduled, its weight should be slightly larger than zero. Now, let us consider a flow that traverses K links, and use an informal argument to show why it is very intuitive that the flow's total queue accumulation along its route should grow quadratically with the route length. The queue length at the destination for this flow is equal to zero. The queue length at the first upstream node from the destination will be some positive number, say, ϵ. The queue length at the second upstream node from the destination will be even larger and for the purposes of obtaining insight, let us say that it is 2ϵ. Continuing this reasoning further, the total queue length for the flow will be $\epsilon(1+2+\ldots+K)=\Theta(K^2)$. Thus, the total backlog on a path is intuitively expected to grow quadratically in the number of hops. On the other hand, suppose a fixed service rate is allocated to each flow on each link on its path, then the queue length at each hop will be roughly $O(1)$ depending on the utilization at that link. With such a fixed service rate allocation, the total end-to-end backlog should then grow linearly in the number of hops. However, such an allocation is possible only if the packet arrival rate generated by each flow is known to the network a priori. One of the contributions of this paper is to use counters called *shadow queues* introduced in [10] to allocate service rates to each flow on each link in an adaptive fashion without knowing the set of packet arrival rates.

We will also show that the concept of shadow queues can reduce the number of real queues maintained at each node significantly. In particular, we will show that it is sufficient to maintain per-neighbor queues at each node, instead of per-flow queues required by the back-pressure algorithm in the case of fixed routing. In large networks, the number of flows is typically much larger compared to the number of neighbors of each node, thus using per-neighbor queues can result in significant reduction in implementation complexity. A different idea to reduce the number of queues at each node has been proposed in [11], but the implementation using shadow queues has the additional benefit of delay reduction.

Next, we discuss the case of adaptive routing. The back-pressure algorithm tends to explore many routes to find sufficient capacity in the network to accommodate the offered traffic. Since the goal of the algorithm is to maximize throughput, without considering Quality of Service (QoS), back-pressure based adaptive routing can result in very long paths leading to unnecessarily excessive delays. In this paper, we propose a modification to the back-pressure algorithm which forces it to first explore short paths and only use long paths when they are really needed to accommodate the offered traffic. Thus, under our proposed modification, the back-pressure algorithm continues to be throughput-optimal, but it pays attention to the delay performance of the network. We also refer the reader to a related work in [12] where the authors use the same cost function as us, but their formulation is different and hence their solution is also different.

Due to the page limit, we only present the main ideas and results in this paper. The reader is referred to the associated technical report [13] for detailed proofs and extensive simulation results.

II. System model

Consider a network modeled by a graph, $\mathcal{G} = (\mathcal{N}, \mathcal{L})$, where $\mathcal{N}$ is the set of nodes and $\mathcal{L}$ is the set of links. We assume that time is slotted, with a typical time slot denoted by t. If a link (n, m) is in $\mathcal{L}$, then it is possible to transmit packets from node n to node m subject to the interference constraints which will be described shortly.

We let $\mathcal{F}$ be the set of flows that share the network resources. Packets of each flow enter the network at one node, travel along multiple hops (which may or may not predetermined), and then exit the network at another node. For each flow $f \in \mathcal{F}$, let $b(f)$ denote the begin (entering) node, and $e(f)$ denote the end (exiting) node of flow f.

We define a valid schedule $\pi = \left(c_1^\pi, c_2^\pi, \ldots, c_{|\mathcal{L}|}^\pi\right)$ to be a set of link rates (measured in terms of number of packets) that can be simultaneously supported. We make a natural and non-restrictive assumption that if π is a valid schedule, then replacing any subset of its components by zeros will produce a valid schedule as well. We also assume that c_l^π is upper-bounded by some c_{max} for any π and l. Let Γ be the set of all possible valid schedules, and $co(\Gamma)$ denote the convex hull of Γ.

Let Λ denote the network's *capacity region*, which is defined as the set of all flow rates which are supportable by the network. The traffic in the network can be *elastic* or *inelastic*. If the traffic is *inelastic*, i.e., the flows' rates are fixed (and within the capacity region), then the goal is to route/schedule the traffic through the network while ensuring that the queues in the network are stable. If the traffic is *elastic*, then the goal is to allocate the network's resources to all flows in some fair manner. More precisely, suppose that each flow has a utility function associated with it. The utility function of flow f, denoted by $U_f(\cdot)$, is defined as a function of the data rate x_f sent by flow f, and assumed to be concave and non-decreasing. The goal, in the case of elastic traffic, is to determine the optimal solution to the following resource allocation problem:

$$\begin{aligned} \max \quad & \sum_{f \in \mathcal{F}} U_f(x_f) \\ \text{s.t.} \quad & x \in \Lambda. \end{aligned} \tag{1}$$

It has been shown that, for *inelastic* traffic, the back-pressure algorithm is *throughput-optimal*, i.e., it can support any arrival rate vector which lies inside the capacity region Λ. Furthermore, for *elastic* traffic, a joint congestion control and back-pressure routing/scheduling algorithm has been shown to be able to solve the resource allocation problem (1). However, as we mentioned in Section I, the delay performance of such algorithms can be quite poor. In the subsequent sections, we describe our architectures and algorithms in detail.

III. The shadow algorithm

In this section, we consider networks with fixed routing, and propose an architecture to reduce delays and reduce the number of queues maintained at each node. The main idea is use a fictitious queueing system called the *shadow queueing* system to perform flow control and resource allocation in the network while using only a single physical FIFO queue for each outgoing link (also known as per-neighbor queueing) at each node. The idea of shadow queues was introduced in [10], but the main goal there was to extend the network utility maximization framework for wireless networks to include multicast flows. However, one of the main points of this paper is to show that shadow queues can be useful even in networks with unicast flows only for the purpose of delay reduction. Further, the idea of using per-neighbor queueing and establishing its stability is new here.

A. Description

The traditional back-pressure algorithm requires the queue length of every flow that passes through a node to perform resource allocation. The main idea of the shadow algorithm is to decouple the storage of this information from the queueing data structure required to store packets at each node. The details of the shadow algorithm are described as follows.

Queues and Counters: At each node, instead of keeping a separate queue for each flow as in the back-pressure algorithm, a FIFO (first-come-first-served) queue is maintained for each outgoing link. This FIFO queue stores packets for all flows going through the corresponding link. When a node receives a packet, it looks at the packet's header: if the node is not the final destination of that packet, it will send the packet to the FIFO queue of the next-hop link; otherwise, it will deliver the packet to the upper layer. We let $P_{nm}[t]$ denote the length of the queue maintained at link (n, m) and at the beginning of time slot t.

Each node maintains a separate *shadow* queue (i.e., a counter) for each flow going through it. Let $\tilde{Q}_n^f[t]$ be the length of the shadow queue (i.e., the value of the counter) of flow f at node n at the beginning of time slot t. The shadow queues and real queues are updated according to the scheduling algorithm described next. Note that each node still needs to keep a separate shadow queue for every flow going through it, but these are just counters, not actual physical queues. A counter is much easier to implement than a physical queue.

Back-pressure scheduling using the shadow queue lengths: At time slot t,

- Each link looks at the maximum *shadow* differential backlog of all flows going through that link:
$$w_{nm}[t] = \max_{f:(n,m)\in L(f)} \left(\tilde{Q}_n^f[t] - \tilde{Q}_m^f[t]\right). \qquad (2)$$
- Back-pressure scheduling:
$$\pi^*[t] = \max_{\pi\in\Gamma} \sum_{(n,m)} c_{nm}^{\pi} w_{nm}[t]. \qquad (3)$$
- A schedule $\pi^* = (c_1^{\pi}, c_2^{\pi}, \ldots, c_{|\mathcal{L}|}^{\pi})$ is interpreted by the network as follows: link (n,m) transmits c_{nm}^{π} shadow packets from the shadow queue of the flow whose differential backlog achieves the maximum in (2) (if the shadow queue has fewer than c_{nm}^{π} packets, then it is emptied); link (n,m) also transmits as many real packets as shadow packets from its real FIFO queue. Again, if the number of real packets in the queue is less than the number of transmitted shadow packets, then all the real packets are transmitted.

We recall that shadows queues are just counters. The action of "transmitting shadow packets" is simply the action of updating the counters' values. In other words, "transmitting" k shadow packets from $\tilde{Q}_n^f$ to $\tilde{Q}_m^f$ means that we subtract k from $\tilde{Q}_n^f$ and add k to $\tilde{Q}_m^f$. From the above description, it should be clear that the shadow packets can be interpreted as permits which allow a link to transmit. Unlike the traditional back-pressure algorithm, the permits are associated with just a link rather than with a link and a flow.

Congestion control at the source: At time slot t, the source of flow f computes the rate at which it injects packets into the ingress *shadow* queue as follows:

$$x_f[t] = \min\left\{U_f^{'-1}\left(\frac{\tilde{Q}_{b(f)}^f[t]}{M}\right), x_{max}\right\} \qquad (4)$$

where x_{max} is an upper-bound of the arrival rates, and M is a positive parameter. The source also generates real traffic at rate $\beta x_f[t]$ where β is a positive number less than 1. If x_f and βx_f are not integers, the actual number of shadow and real packets generated can be random variables with these expected values. Since the shadow packets are permits that allow real-packet transmission, from basic queueing theory, it follows that the actual packet arrival rate must be slightly smaller than the shadow packet arrival rate to ensure the stability of real queues. The parameter β is chosen to be less than 1 for this purpose. As we will see later in simulations, the queue backlog in the network would be smaller for smaller values of β.

The above description of the shadow algorithm applies to elastic traffic. For inelastic traffic, the same shadow algorithm can be used without congestion control. To ensure stability of the real queues, if the real arrival rate of an inelastic flow is λ_f, the shadow arrival rate for this flow must be larger than λ_f. For example, if we wish to make the shadow arrival rate larger than the real arrival rate by a factor of $(1+\epsilon)$, it can accomplished as follows: for every real packet arrival, generate a shadow packet. Generate an additional shadow packet for each real packet with probability ϵ. This procedure ensures that the shadow arrival rate will be $(1+\epsilon)$ times the real arrival rate. For the algorithm to be stable, the set of arrival rates $\{\lambda_f(1+\epsilon)\}_f$ must lie in the interior of capacity region.

We note that the concept of shadow queues here is different from the notion of virtual queues used in [14] for the Internet and in [5] for wireless networks. In networks with virtual queueing systems, the arrival rates to both the real and virtual queues are the same, but the virtual queue is drained at a slower rate than the real queue. Instead, here the arrival rates to the real queues are slightly smaller than the arrival rates to the corresponding shadow queues. This subtle difference is important in that it allows us to use per-neighbor FIFO queues and prove stability in a multihop wireless network in the next section.

B. Stability of the shadow algorithm

In this subsection, we establish the optimality and stability of the real and shadow queues. First, we note that the optimality of the resource allocation and the stability of shadow queues follow from previous results in the literature. In particular, we have the following theorem.

Theorem 1: The shadow-queue-based congestion control and scheduling algorithms described in Section III-A above asymptotically achieve the optimal rate allocation, i.e.,

$$\lim_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}[x[t]] = x^* + O(1/M), \qquad (5)$$

where x^* is the optimal solution to (1). Furthermore, the *shadow* queues are stable in the sense that the Markov chain of shadow queues $\tilde{Q}[t]$ is positive recurrent and the steady-state expected values of the shadow queue lengths are bounded as follows:
$$\sum_{n,f} \mathbb{E}(\tilde{Q}_n^f[\infty]) = O(M).$$

The remaining goal is to prove the stability of the real queues. Note that the sources are sending real traffic with smaller rates than shadow traffic, and we know that the shadow queues are stable. However, it does not automatically mean that the real queues are stable as well, since each of them is an aggregated FIFO queue storing packets for all flows going through its corresponding link. Fortunately, we can apply results from the stochastic networks literature to establish the following result.

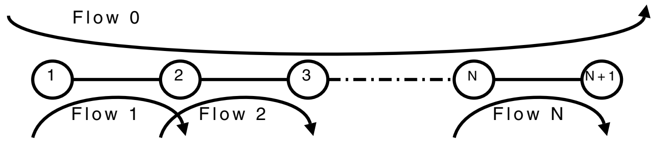

Fig. 1. The linear network with N links.

Theorem 2: The process describing the joint evolution of both shadow and real queues, $\left((\tilde{Q}_n^f[t])_{f\in\mathcal{F},n\in\mathcal{N}};(P_{nm}[t])_{(n,m)\in\mathcal{L}}\right)$, is an irreducible, aperiodic, positive recurrent Markov chain. Therefore, the real FIFO queues are also stable.

The proof is based on the fluid limit approach and a result by Bramson [15]. More details can be found in our technical report version [13] of the paper.

Note that the real traffic throughput will always be slightly smaller than the optimal solution to (1), but this difference from the optimal solution can be made arbitrarily small by adjusting the parameter β.

C. Performance comparison: back-pressure algorithm versus the shadow algorithm

We first present simple calculations to get some feel for the performance of the traditional back-pressure algorithm when it is used with congestion control. We confine our discussions to the case of a linear network with N links as in Figure 1. There are $N+1$ flows sharing this network: one flow (indexed 0) goes through all N links, and N other flows (indexed 1 to N) where each of them goes through each link. The capacity of each link is c, and for simplicity, we assume that there is no interference between links.

Proposition 1: Consider the resource allocation problem (1) for a linear network in Figure 1, and let q_i^* and $q_{0,i}^*$ be the optimal lengths of the queues maintained at node i for flow i and flow 0, respectively. For the following class of utility functions which model a large class of fairness concepts [16],

$$U_i(x)=\frac{x^{1-\alpha}}{1-\alpha},\quad \alpha>0,$$

we have that $q_i^*=q_{0,i}^*-q_{0,i+1}^*=\Theta(1),\ i=1,\ldots,N.$ Hence, the optimal end-to-end total queue length for flow 0 is $\sum_{i=1}^N q_{0,i}^*=\Theta\left(N^2\right).$

That is, the combined back-pressure and congestion control algorithm for elastic traffic can lead to *quadratic* end-to-end queueing delay in terms of the number of hops.

In the case of *inelastic* traffic, i.e., the flows' rates are fixed, the following theorem establishes an upper-bound on the end-to-end queue backlog of any flow when the back-pressure algorithm is used.

Theorem 3: Consider a general topology network accessed by a set of flows with fixed routes. Let K_{max} be the maximum number of hops in the route of any flow, i.e., $K_{max}=\max_f|L(f)|$. Suppose, the arrival rate vector λ is such that, for some $\epsilon>0$, $(1+\epsilon)\lambda$ lies in the interior of the capacity region of the network. Then, the expected value of the sum of queue lengths (in steady-state) along the route of any flow f is bounded as follows:

$$\mathbb{E}\left[\sum_{n\in R(f)}Q_n^f[\infty]\right]\ \leq\ \frac{1+\epsilon}{\epsilon}\frac{b}{\lambda_f}|\mathcal{F}|K_{max}^2\ ,\ \forall f\in\mathcal{F},$$

where constant $b>0$ depends only on c_{max}.

While the above result is only an upper bound, it suggests the *quadratic* growth of the total flow queue length on the flow route length, which is validated by our simulation [13].

Now, as mentioned in Section I, if a fixed rate (larger than its arrival rate) is allocated to each flow, then the total queue length of a flow is expected to increase *linearly* in terms of the number of hops instead of *quadratically*. In fact, that is the case of the shadow algorithm, since the shadow algorithm is "reserving" capacity between each source-destination pair (i.e., each flow), and the sources are sending data with rates less than the "reserved" capacities. The shadow algorithm, thus, yields a significant gain in delay performance (*linear* versus *quadratic*) at the expense of a small loss in throughput (represented by parameters β or ϵ in Section III-A). Our simulation results [13] validate this intuitive argument.

IV. MIN-RESOURCE ROUTING USING BACK-PRESSURE ALGORITHM

In this section, we consider wireless networks where each flow's route is not pre-determined, but is adaptively chosen by the back-pressure algorithm for each packet. As mentioned in Section I, the back-pressure algorithm explores all paths in the network and as a result may choose paths which are unnecessarily long which may even contain loops, thus leading to poor performance. We address this problem by introducing a cost function which measures the total amount of resources used by all the flows in the network. Specifically, the cost function is the sum of traffic loads on all links, i.e. it's packets $\times$ hops per unit time. In the case of inelastic flows, the goal is to minimize this cost subject to network capacity constraints; by the nature of the cost function, we obtain the *min-resource* (or, min-hop) routing problem. In the case of elastic flows, one can maximize the sum of flow utilities minus a weighted function of the cost described above, where the weight provides a tradeoff between network utility and resource usage. Obviously, the inelastic case (min-resource routing) problem is a special case of the elastic case.

It may seem surprising, but the above problem can be solved (asymptotically) exactly, by an extension of the conventional back-pressure algorithm. To simplify exposition, we only present the inelastic case algorithm here

Given a set of packet arrival rates that lie within the capacity region, our goal is to find the routes for flows such that use as few resources as possible in the network:

$$\begin{aligned} \min\quad & \sum_{(n,m)}\mu_{nm} \qquad (6)\\ s.t.\quad & \lambda_f\mathcal{I}_{\{n=b(f)\}}+\sum_{(k,n)}\mu_{kn}^f\leq\sum_{(n,m)}\mu_{nm}^f,\ f\in\mathcal{F},n\in\mathcal{N},\\ & \{\mu_{nm}\}_{(n,m)\in\mathcal{L}}\ \in\ co(\Gamma), \end{aligned}$$

where μ_{nm}^f is the rate that link (n,m) allocates to serve flow f, i.e., $\mu_{nm} = \sum_f \mu_{nm}^f$, and λ_f is the fixed rate of flow f. An algorithm that asymptotically solves the min-resource routing problem (6) is as follows. (It is a special case of the algorithm in [3], where the scaling parameter $1/M$ is called β.)

Min-resource routing by back-pressure: At time slot t,

- Each node n maintains a separate queue of packets for each destination d; its length is denoted $Q_n^d[t]$. Each link is assigned a weight

$$w_{nm}[t] = \max_{d \in D} \left(\frac{1}{M} Q_n^d[t] - \frac{1}{M} Q_m^d[t] - 1 \right), \quad (7)$$

  where $M > 0$ is a parameter (having the same meaning as earlier in this paper.)
- Scheduling/routing rule:

$$\pi^*[t] = \max_{\pi \in \Gamma} \sum_{(n,m)} \pi_{nm} w_{nm}[t].$$

Note that the above algorithm does not change if we replace the weights in (7) by the following, re-scaled ones:

$$w_{nm}[t] = \max_{d \in D} \left(Q_n^d[t] - Q_m^d[t] - M \right), \quad (8)$$

and therefore, compared with the traditional back-pressure scheduling/routing, the only difference is that each link weight is equal to the maximum differential backlog *minus parameter* M. ($M = 0$ reverts the algorithm to traditional.)

The performance of the stationary process which is "produced" by the algorithm with fixed parameter M is within $O(1/M)$ of the optimal (analogously to (5)):

$$\left| \mathbb{E}\left[\sum_{(n,m)} \pi_{nm}^*[t] \right] - \sum_{(n,m)} \mu_{nm}^* \right| = O(1/M),$$

where μ^* is an optimal solution to (6). However, larger M means larger, $O(M)$ queues and slower convergence to the (nearly optimal) stationary regime. On the other hand, "too small" M results in a stationary regime being "too far" from optimal, and queues being large for that reason. Therefore, a good value for M for a practical use should be neither too large nor too small. Our simulations [13] confirm these intuitions.

V. Conclusions

In this paper, we have proposed a new shadow architecture to improve the delay performance of back-pressure scheduling algorithm. The shadow queueing system allows each node to maintain a single FIFO queues for each of its outgoing links, instead of keeping a separate queue for each flow in the network. This architecture not only reduces the queue backlog (or, equivalently, delay by Little's law) but also reduces the number of actual physical queues that each node has to maintain. Next, we proposed an algorithm that forces the back-pressure algorithm to use the minimum amount of network resources while still maintaining throughput optimality. This results in better delay performance compared to the traditional back-pressure algorithm.

We presented the shadow algorithm for the case of fixed routing, i.e., the route for each flow is fixed. The shadow algorithm can also be used in the case of adaptive routing; however, a node cannot use just one FIFO queue for each neighbor, but still has to maintain a separate queue for each destination at each node. On the other hand, it would be interesting to study if a single per-neighbor FIFO queue can be maintained even in the case of adaptive routing, which is a topic for future research.

Acknowledgments

The work of the first two authors has been supported in part by DTRA Grant HDTRA1-08-1-0016, NSF Grant CNS 07-21286, and Army MURI 2008-01733.

References

[1] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE Transactions on Automatic Control*, vol. 37, pp. 1936–1948, December 1992.

[2] X. Lin and N. Shroff, "Joint rate control and scheduling in multihop wireless networks," in *Proceedings of IEEE Conference on Decision and Control*, vol. 2, Paradise Island, Bahamas, December 2004, pp. 1484–1489.

[3] A. Stolyar, "Maximizing queueing network utility subject to stability: Greedy primal-dual algorithm," *Queueing Systems*, vol. 50, no. 4, pp. 401–457, August 2005.

[4] ——, "Greedy primal-dual algorithm for dynamic resource allocation in complex networks," *Queueing Systems*, vol. 54, no. 3, pp. 203–220, 2006.

[5] A. Eryilmaz and R. Srikant, "Fair resource allocation in wireless networks using queue-length based scheduling and congestion control," in *Proceedings of IEEE INFOCOM*, vol. 3, Miami, FL, March 2005, pp. 1794–1803.

[6] ——, "Joint congestion control, routing and MAC for stability and fairness in wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 8, pp. 1514–1524, August 2006.

[7] M. Neely, E. Modiano, and C. Li, "Fairness and optimal stochastic control for heterogeneous networks," in *Proceedings of IEEE INFOCOM*, vol. 3, Miami, FL, March 2005, pp. 1723–1734.

[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 *Proceedings of IEEE INFOCOM*, Phoenix, AZ, April 2008, pp. 1103–1111.

[9] U. Akyol, M. Andrews, P. Gupta, J. Hobby, I. Saniee, and A. L. Stolyar, "Joint scheduling and congestion control in mobile ad-hoc networks," in *Proceedings of IEEE INFOCOM*, Phoenix, AZ, April 2008, pp. 619–627.

[10] L. Bui, R. Srikant, and A. Stolyar, "Optimal resource allocation for multicast sessions in multihop wireless networks," *Philosophical Transactions of the Royal Society, Series A*, vol. 366, no. 1872, pp. 2059–2074, June 2008.

[11] L. Ying, R. Srikant, and D. Towsley, "Cluster-based back-pressure routing algorithm," in *Proceedings of the IEEE INFOCOM*, Phoenix, AZ, April 2008, pp. 484–492.

[12] L. Ying, S. Shakkottai, and A. Reddy, "On combining shortest-path and back-pressure routing over multihop wireless networks," in *Proceedings of the IEEE INFOCOM*, 2009.

[13] L. Bui, R. Srikant, and A. Stolyar, "Novel architectures and algorithms for delay reduction in back-pressure scheduling and routing," arXiv preprint 0901.1312, available at http://arxiv.org/abs/0901.1312.

[14] R. J. Gibbens and F. P. Kelly, "Resource pricing and the evolution of congestion control," *Automatica*, vol. 35, pp. 1969–1985, 1999.

[15] M. Bramson, "Convergence to equilibria for fluid models of FIFO queueing networks," *Queueing Systems*, vol. 22, no. 1-2, pp. 5–45, March 1996.

[16] J. Mo and J. Walrand, "Fair end-to-end window-based congestion control," *IEEE/ACM Transactions on Networking*, vol. 8, no. 5, pp. 556–567, October 2000.