Scheduling with Per-link Queues and No Per-flow Information in Multi-hop Wireless Networks

Bo Ji, Changhee Joo and Ness B. Shroff

***Abstract*—This paper focuses on designing and analyzing throughput-optimal scheduling policies that avoid using per-flow or per-destination information, maintain a single data queue for each link, exploit only local information, and potentially improve the delay performance, for multi-hop wireless networks under general interference constraints. Although the celebrated back-pressure algorithm maximizes throughput, it requires per-flow or per-destination information (which may be difficult to obtain and maintain), maintains per-flow or per-destination queues at each node, relies on constant exchange of queue length information among neighboring nodes to calculate link weights, and may result in poor delay performance. In contrast, the proposed schemes can circumvent these drawbacks while guaranteeing throughput optimality. We rigorously analyze the throughput performance of the proposed schemes and show that they are throughput-optimal using fluid limit techniques via an inductive argument. We also conduct simulations to show that the proposed schemes can substantially improve the delay performance.**

I. Introduction

Link scheduling is a critical resource allocation functionality in multi-hop wireless networks, and also perhaps the most challenging. The seminal work of [1] introduces a joint adaptive routing and scheduling algorithm, called back-pressure, that has been shown to be throughput-optimal, i.e., it can stabilize the network under any feasible load. This paper focuses on the settings with fixed routes, where the back-pressure algorithm degenerates to a scheduling algorithm consisting of two components: flow scheduling and link scheduling. The back-pressure algorithm calculates the weight of a link as the product of the link capacity and the maximum "back-pressure" (i.e., the queue length difference between the queues at this link and the next hop link for each flow) among all the flows passing through the link, and solves a MaxWeight problem to activate a set of non-interfering links that have the largest weight sum. The flow with the maximum queue length difference at a link is chosen to transmit packets when the link is activated.

Since the development of the back-pressure algorithm, there have been numerous variations that have integrated them into an overall optimal cross-layer solution. However, the MaxWeight problem that the back-pressure-type of algorithms need to solve requires centralized operations, and is NP-hard under general interference constraints [2].

B. Ji is with Department of ECE at the Ohio State University. C. Joo is with Department of EECE at Korea University of Technology and Education, Korea. N. B. Shroff is with Departments of ECE and CSE at the Ohio State University. Emails: ji@ece.osu.edu, cjoo@kut.ac.kr, shroff@ece.osu.edu.

This work was supported in part by ARO MURI Award W911NF-08-1-0238, and NSF Awards 1012700-CNS, 0721236-CNS, and 0721434-CNS.

Recently, exciting advances have been made in developing simple, distributed and throughput-optimal algorithms that are based on Carrier Sensing Multiple Access (CSMA) [3]–[5]. The key idea of these CSMA-based algorithms is that the links adaptively adjust their channel attempting probabilities based on local queue lengths (or locally measured arrival and service rates). The CSMA-based algorithms of [3]–[5] are typically designed for single-hop traffic. The work in [3] extends the idea to the case of multi-hop traffic. *However, such extensions still rely on the idea of back-pressure that requires per-flow information at each node and constant exchange of queue length information among neighboring nodes, which become major obstacles to their distributed implementation.*

While the back-pressure-type of scheduling algorithms (including CSMA for multi-hop traffic) maximize throughput, they typically have the following shortcomings: 1) require per-flow or per-destination information (which may be difficult to obtain and maintain, especially in large networks where there are numerous flows, e.g., the Internet), 2) need to maintain separate queues for each flow or destination at each node, 3) rely on constant exchange of queue length information among neighboring nodes to calculate link weights, and 4) may result in poor overall delay performance, as the queue length needs to increase as one moves back from a flow destination to its source, which leads to a large queue length build-up along the route a flow takes [6], [7].

An important question is whether one can circumvent the above drawbacks of the back-pressure-type of algorithms and design throughput-optimal scheduling algorithms that do not require per-flow or per-destination information, maintain a small number of data queues (ideally, a single data queue for each link), exploit only local information when making scheduling decisions, and potentially have good delay performance. Recently, there have been studies (e.g., [6], [8]–[10]) towards this direction. A cluster-based back-pressure algorithm that can reduce the number of queues is proposed in [9], where nodes (routers) are grouped into clusters and each node needs only to maintain separate queues for destinations within its cluster. In [6], the authors propose a back-pressure policy making scheduling decisions in a shadow layer (where counters are used as per-flow shadow queues). Their scheme needs only to maintain a single *First-In First-Out (FIFO)* queue instead of per-flow queues for each link and shows dramatically improvement in the delay performance. However, their shadow algorithm still requires per-flow information and constant exchange of shadow queue length information among neighboring nodes. The work in [8] proposes exploiting

978-1-61284-823-5/11/$26.00 ©2011 IEEE

local queue length information to design throughput-optimal scheduling algorithms. Their approach combined with CSMA algorithms of [3]–[5] can achieve fully distributed scheduling without any information exchange. Their scheme is based on a two-stage queue structure, where each node maintains two types of data queues: per-flow queues and per-link queues. The two-stage queue structure imposes additional complexity, and is similar to queues with regulators [11], which have been empirically noted to have very large delays. In [10], the authors propose a back-pressure algorithm that integrates the shortest path routing to minimize average number of hops between each source and destination pair. Their scheme increases the number of queues and maintains a separate queue $\{i, d, k\}$ at node i for the packets that will be delivered to destination node d within k hops.

Although these algorithms partly alleviate the effect of the aforementioned disadvantages of the traditional back-pressure algorithms, to the best of our knowledge, no work has addressed all the aforementioned four issues. In particular, a critical drawback of the earlier mentioned works is that they require per-flow or per-destination information to guarantee throughput performance. In this paper, we propose a class of throughput-optimal schemes that can remove this per-flow or per-destination information requirement, maintain a single data queue for each link, and exploit only local information. A by-product is that these proposed schemes also improve the delay performance.

The main contributions of our paper are as follows.

First, we propose a scheduling scheme with *per-hop* queues to address the four key issues mentioned earlier, where a single FIFO queue $Q_{l,k}$ is maintained for all packets whose k-th hop is link l. This hop-distance information is much easier to obtain and maintain compared to per-flow or per-destination information. A shadow algorithm similar to [6] is adopted in our framework, where a shadow queue is associated with each data queue. We consider the MaxWeight algorithm based on shadow queue lengths, and show that this per-Hop-Queue-based MaxWeight Scheduler (HQ-MWS) is throughput-optimal using fluid limit techniques via a hop-by-hop inductive argument. For illustration, in this paper, we focus on the centralized MaxWeight-type of policies. However, one can readily extend our approach to a large class of scheduling policies (where fluid limit techniques can be used). For example, combining our approach with the CSMA-based algorithms of [3]–[5], we can completely remove the requirement of information exchange, and develop throughput-optimal scheduling schemes that are fully distributed (i.e., no information exchange is required). To the best of our knowledge, this is the first work that develops throughput-optimal scheduling schemes without per-flow or per-destination information in wireless networks with multi-hop traffic. In addition, we believe that using this type of per-hop queue structure to study the problem of link scheduling is of independent interest.

Second, we have also developed schemes with per-link queues (i.e., a single data queue for each link) instead of per-hop queues, extending the idea to per-Link-Queue-based MaxWeight Scheduler (LQ-MWS). We propose two schemes based on LQ-MWS using different queueing disciplines. We first combine it with the *priority* queueing discipline (PLQ-MWS), where a priority is given to the packet that traverses a smaller number of hops, and show throughput optimality of PLQ-MWS. This, however, requires that nodes sort packets according to their hop-distance information. We remove this restriction by combining LQ-MWS with the FIFO queueing discipline (FLQ-MWS). We show throughput optimality of FLQ-MWS in networks where flows do not form loops.

Finally, we show through simulations that PLQ-MWS and FLQ-MWS can significantly improve the delay performance in certain scenarios, which implies that maintaining per-link queues not only simplifies the data structure, but also can contribute to scheduling efficiency and delay performance.

The remainder of the paper is organized as follows. In Section II, we present a detailed description of our system model. In Section III, we prove throughput optimality of HQ-MWS using fluid limit techniques via a hop-by-hop inductive argument. We extend our ideas to show that PLQ-MWS is throughput-optimal in Section IV, and that FLQ-MWS is throughput-optimal in special networks in Section V. We evaluate different scheduling schemes through simulations in Section VI. Finally, we conclude our paper in Section VII.

Due to space limitations, the proofs are omitted and provided in our online technical report [12].

II. System Model

We consider a multi-hop wireless network described by a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ denotes the set of nodes and $\mathcal{E}$ denotes the set of links. Nodes are wireless transmitters/receivers and links are wireless channels between two nodes if they can directly communicate with each other. Let $b(l)$ and $e(l)$ denote the transmitting node and receiving node of link $l = (b(l), e(l)) \in \mathcal{E}$, respectively. Note that we distinguish links (i, j) and (j, i). We assume a time-slotted system with a single frequency channel. Let c_l denote the link capacity of link l, i.e., link l can transmit at most c_l packets during the time slot if none of the links that interfere with l is transmitting. We assume unit capacity, i.e., $c_l = 1$ for all $l \in \mathcal{E}$. A flow is a stream of packets from a source node to a destination node. Packets are injected at the source, and traverse multiple links to the destination via multi-hop communications. Let $\mathcal{S}$ denote the set of flows in the network. We assume that each flow s has a single, fixed, and loop-free route that is denoted by $\mathcal{L}(s) = \{l_1^s, \cdots, l_{|\mathcal{L}(s)|}^s\}$, where the route of flow s has $|\mathcal{L}(s)|$ hop-length from the source to the destination, l_k^s denotes the k-th hop link on the route of flow s, and $|\cdot|$ denotes the cardinality of a set. Let $L^{\max} \triangleq \max_s |\mathcal{L}(s)| < \infty$ denote the length of the longest route over all flows. Let $H_{l,k}^s$ be 1, if link l is the k-th hop link on the route of flow s, and 0, otherwise. Note that the assumption of single route and unit capacity is only for ease of exposition, and one can easily extend the results to more general scenarios with multiple fixed routes and heterogeneous

link capacities. We also restrict our attention to those links that have flows passing through them. Hence, without loss of generality, we assume that $\sum_s \sum_{k=1}^{|\mathcal{L}(s)|} H_{l,k}^s \geq 1$, for all $l \in \mathcal{E}$.

The interference set of link l is defined as $I(l) \triangleq \{j \in \mathcal{E} \mid \text{link } j \text{ interferes with link } l\}$. We consider a general interference model, where the interference is symmetric, i.e., for any $l, j \in \mathcal{E}$, if $l \in I(j)$, then $j \in I(l)$. A *schedule* is a set of (active or inactive) links, and can be represented by a vector $M \in \{0,1\}^{|\mathcal{E}|}$, where component M_l is set to be 1 if link l is active, and 0 if it is inactive. A schedule M is said to be *feasible* if no two links of M interfere with each other, i.e., $l \notin I(j)$ for all l, j with $M_l = 1$ and $M_j = 1$. Let $\mathcal{M}$ denote the set of all feasible schedules over $\mathcal{E}$, and let $Co(\mathcal{M})$ denote its convex hull.

Let $F_s(t)$ denote the cumulative number of packet arrivals at the source node of flow s up to time slot t. We assume that the packet arrival processes satisfy the Strong Law of Large Numbers (SLLN). Let λ_s denote the mean arrival rate of flow s, and let $\lambda \triangleq [\lambda_s]$ denote its vector.

As in [13], a discrete-time queueing system is said to be *stable*, if the underlying Markov chain is positive Harris recurrent. When the state space is countable and all states communicate, this is equivalent to the Markov chain being positive recurrent. We define the *throughput region* of a scheduling policy as the set of arrival rate vectors for which the network is stable under this policy. Further, we define the *optimal throughput region* (or *stability region*) as the union of the throughput regions of all possible scheduling policies, including the offline policies. We denote by Λ^* the optimal throughput region, and Λ^* can be presented as

$$\Lambda^* \triangleq \{\lambda \mid \text{for some } \phi \in Co(\mathcal{M}), \sum_s \sum_k H_{l,k}^s \lambda_s \leq \phi_l \text{ for all links } l \in \mathcal{E}\}. \quad (1)$$

An arrival rate vector is strictly inside Λ^*, if the inequalities above are all strict.

III. Scheduling with Per-hop Queues

In this section, we propose scheduling policies with per-hop queues and shadow algorithm to design throughput-optimal scheduling policies. We later extend the ideas to developing schemes with per-link queues. Note that we make use of the MaxWeight algorithm, a centralized policy, for ease of presentation. Our approach can be combined with the CSMA-based algorithms, leading to fully distributed throughput-optimal scheduling schemes [12].

A. Algorithm Description

We start with description of queue structure, and then specify our scheduling scheme based on per-hop queues and a shadow algorithm. We assume that, at each link l, a single FIFO data queue $Q_{l,k}$ is maintained for packets whose k-th hop is link l, where $1 \leq k \leq L^{max}$. Such queues are called *per-hop* queues. Slightly abusing the notation, we also use $Q_{l,k}(t)$ to denote the queue length of $Q_{l,k}$ at time slot t. Let $\Pi_{l,k}(t)$ denote the service of $Q_{l,k}$ at time slot t, which takes a value of c_l (i.e., 1 in our setting), if queue $Q_{l,k}$ is active, or 0, otherwise. Let $D_{l,k}(t)$ denote the cumulative number of packet departures from queue $Q_{l,k}$ up to time slot t. Let $A_{l,k}(t)$ be the cumulative number of aggregate packet arrivals (including both exogenous arrivals and arrivals from the previous hops) at queue $Q_{l,k}$ up to time slot t. We adopt the convention that, $A_{l,k}(0) = 0$ and $D_{l,k}(0) = 0$ for all $l \in \mathcal{E}$ and $1 \leq k \leq L^{\max}$. The queue length evolves according to the following equations: $Q_{l,k}(t) = Q_{l,k}(0) + A_{l,k}(t) - D_{l,k}(t)$.

Let $\hat{Q}_{l,k}$ denote shadow queue associated with the corresponding data queue $Q_{l,k}$, and let $\hat{Q}_{l,k}(t)$ denote its queue length at time slot t. Below we will describe the arrival and service of shadow queues. We denote by $\hat{A}_{l,k}(t)$ and $\hat{D}_{l,k}(t)$ its cumulative amount of arrivals and departures up to time slot t, respectively. Also, let $\hat{\Pi}_{l,k}(t)$ and $\hat{P}_{l,k}(t) \triangleq \hat{A}_{l,k}(t) - \hat{A}_{l,k}(t-1)$ denote the amount of service and arrivals for queue $\hat{Q}_{l,k}$ at time slot t, respectively. We set $\hat{A}_{l,k}(0) = 0$ and $\hat{D}_{l,k}(0) = 0$ for all queues $\hat{Q}_{l,k}$. We specify the arrivals for shadow queue $\hat{Q}_{l,k}$ as

$$\hat{P}_{l,k}(t) = (1+\epsilon)\frac{A_{l,k}(t)}{t}, \quad (2)$$

i.e., $(1+\epsilon)$ times the average amount of packet arrivals at the corresponding data queue $Q_{l,k}$ up to time slot t, where $\epsilon > 0$ is a sufficiently small positive number such that $(1+\epsilon)\lambda$ is also strictly inside Λ^* given that λ is strictly inside Λ^*. Then, the shadow queue lengths evolve according to the following equations: $\hat{Q}_{l,k}(t) = \hat{Q}_{l,k}(0) + \hat{A}_{l,k}(t) - \hat{D}_{l,k}(t)$.

Next, we specify the scheduling scheme based on shadow queues as follows.

Per-Hop-Queues-based MaxWeight Scheduler (HQ-MWS): At each time slot t, the scheduler serves k^*-hop queues of links in M^* (i.e., $\Pi_{l,k^*}(t) = 1$ for $l \in M^*$, and $\Pi_{l,k}(t) = 0$ otherwise), where

$$k^* \in \text{argmax}_k \, \hat{Q}_{l,k}(t), \text{ for each link } l \in \mathcal{E}, \quad (3)$$

$$M^* \in \text{argmax}_{M \in \mathcal{M}} \sum_{l \in \mathcal{E}} \hat{Q}_{l,k^*}(t) \cdot M_l. \quad (4)$$

Also, we set the service of shadow queues as $\hat{\Pi}_{l,k}(t) = \Pi_{l,k}(t)$ for all l and k.

Remarks: The algorithm needs to solve a MaxWeight problem based on the shadow queue lengths, and ties can be broken arbitrarily if there is more than one queue having the largest shadow queue length at a link or there is more than one schedule having the largest weight sum. Note that we have $\Pi_{l,k}(t) = \hat{\Pi}_{l,k}(t)$ under HQ-MWS, for all links $l \in \mathcal{E}$ and $1 \leq k \leq L^{max}$, and for all time slots $t \geq 0$. Once a schedule M^* is selected, data queues Q_{l,k^*} for links l with $M_l^* = 1$ are activated to transmit packets if they are non-empty, and shadow queues $\hat{Q}_{l,k^*}$ "transmit" shadow packets as well. Note that shadow queues are just counters, and the arrival and departure process of a shadow queue is simply an operation of addition and subtraction, respectively.

B. Performance Analysis

We present the main result of this section as follows.

Proposition 1: HQ-MWS is throughput-optimal, i.e., the network is stable under HQ-MWS for any arrival rate vector strictly inside Λ^*.

We prove the stability of the network in the sense that the underlying Markov chain (see [12] for its state description) is positive Harris recurrent under HQ-MWS, using fluid limit techniques [14]. The results of [14] suggest that it is sufficient to show the stability of the fluid limit model. We provide the outline of the proof and refer to our online technical report [12] for details.

We first note that multi-hop traffic is decomposed into single-hop traffic at each shadow queue, i.e., all the shadow packets leave the system once being completely served at the shadow queues. Then we show that, the single-hop shadow traffic gets smoothed under the arrival process of (2), and is strictly inside the optimal throughput region Λ^* with small enough $\epsilon > 0$. Therefore, using a standard Lyapunov approach, we can show the stability for the sub-system consisting of shadow queues.

Now, we consider the data queues starting with the first hop data queue for each link $l \in \mathcal{E}$. Since the arrival process of data queue $Q_{l,1}$ satisfies the SLLN, the instantaneous arrival of shadow queue $\hat{Q}_{l,1}$ is equal to $(1+\epsilon)\sum_s H^s_{l,1}\lambda_s$ for all time $t > 0$. Hence, the service rate of shadow queue $\hat{Q}_{l,1}$ is no smaller than $(1+\epsilon)\sum_s H^s_{l,1}\lambda_s$ for all time $t > 0$, due to the stability of shadow queues. Then, the service rate of data queue $Q_{l,1}$ is also no smaller than $(1+\epsilon)\sum_s H^s_{l,1}\lambda_s$ for all time $t > 0$, due to the fact that $\Pi_{l,k}(t) = \hat{\Pi}_{l,k}(t)$ under HQ-MWS. On the other hand, the arrival rate of data queue $Q_{l,1}$ is equal to $\sum_s H^s_{l,1}\lambda_s$ for all time $t > 0$. Hence, for the first-hop data queue $Q_{l,1}$, the service rate is strictly greater than the arrival rate, and thus its stability is established. Then, using this as an induction base, we can show the stability of data queues via a hop-by-hop inductive argument. This immediately implies that the fluid limit model of the joint system is stable under HQ-MWS.

Although our proposed scheme is motivated by [6], [8], it has important differences. First, in [6], per-flow information is still required by their shadow algorithm. The shadow packets are injected into the network at the sources, and are then "transmitted" to the destinations via multi-hop communications. Their scheme strongly relies on the information exchange of shadow queue lengths to calculate link weights. In contrast, we take a different approach of constructing the instantaneous arrivals at each shadow queue according to (2) that is based on the average amount of packet arrivals at the corresponding data queue. This novel way of injecting shadow packets allows us to decompose multi-hop traffic into single-hop traffic for shadow queues and exploit only local information when making scheduling decisions. Our scheme needs only per-hop (and not per-flow) information, i.e., the number of hops each packet has traversed. Second, although the basic idea behind the shadow arrival process of (2) is similar to the service process of the per-flow queues in [8], the scheme in [8] requires per-flow information and relies on a two-stage queue architecture that consists of both per-flow and per-link data queues, while per-flow information and per-flow queues are completely removed in our scheme. This simplification of required information and data structure is critical, due to the fact that the maximum number of hops in a network is usually much smaller than the number of flows in a large network. For example, in the Internet, a flow typically traverses tens of hops, while there are billions of nodes and thus the number of flows could be extremely large. Moreover, per-flow queues in their scheme play a similar role as regulators [11], which could result in very large delays and are not amenable in practice. The substantial improvement over the scheme in [8] is illustrated using simulations in Section VI.

Note that the hop-distance in our approach is counted from the source. Such per-hop information is easy to obtain (e.g., from *Time-to-Live (TTL)* information in the Internet). Moreover, at each link, packets with the same hop-distance (from the source of each packet to the link) are kept at the same queue, regardless of sources, destinations, and flows, which significantly reduces the number of queues. We next extend our approach to the schemes with per-link queues, and further remove per-hop information requirement.

IV. Scheduling with Priority Per-link Queues

In this section, we extend the ideas to developing schemes with per-link queues, and show that the proposed scheme with *priority* queueing discipline, called **PLQ-MWS**, is throughput-optimal.

A. MaxWeight Algorithm with Per-link Queues

We consider a network where each link l has a single data queue Q_l. Also, let $Q_l(t)$ denote the queue length of Q_l at time slot t, let $A_l(t)$ denote the cumulative number of packet arrivals at queue Q_l up to time slot t, and let $\Pi_l(t)$ denote the service at Q_l at time slot t. We denote by $\hat{Q}_l$ the associated shadow queue with data queue Q_l. Also, let $\hat{Q}_l(t)$, $\hat{\Pi}_l(t)$ and $\hat{P}_l(t)$ denote the queue length, service and amount of arrivals for shadow queue $\hat{Q}_l$ at time slot t, respectively. Similarly, we specify the arrivals for shadow queue $\hat{Q}_l$ as

$$\hat{P}_l(t) = (1+\epsilon)\frac{A_l(t)}{t}, \tag{5}$$

i.e., $(1+\epsilon)$ times the average amount of packet arrivals at the corresponding data queue Q_l up to time slot t, where $\epsilon > 0$ is a sufficiently small positive number such that $(1+\epsilon)\lambda$ is also strictly inside Λ^* given that λ is strictly inside Λ^*.

Next, we specify the MaxWeight algorithm with per-link queues as follows.

Per-Link-Queues-based MaxWeight Scheduler (LQ-MWS): At each time slot t, the scheduler serves links in M^* (i.e., $\Pi_l(t) = 1$ for $l \in M^*$, and $\Pi_l(t) = 0$ otherwise), where

$$M^* \in \operatorname{argmax}_{M\in\mathcal{M}} \textstyle\sum_{l\in\mathcal{E}} \hat{Q}_l(t) \cdot M_l. \tag{6}$$

Also, we set the service of shadow queues as $\hat{\Pi}_l(t) = \Pi_l(t)$ for all l.

Similar as in HQ-MWS, the shadow traffic under LQ-MWS gets smoothed due to the shadow arrival assignment of (5), and the instantaneous arrival rate of shadow queues can be shown to be strictly inside the optimal throughput region Λ^*,

with any queueing discipline (being applied to data queues). Hence, we can show that the fluid limit model for the subsystem consisting of shadow queues is stable under LQ-MWS, using a standard Lyapunov approach and following the same line of analysis for HQ-MWS.

B. Throughput Optimality of PLQ-MWS

We develop a scheduling scheme called **PLQ-MWS** by combining LQ-MWS with priority queueing discipline. To determine the priority of packets at each per-link queue, we define *hop-class* as follows: A packet has a hop-class-k, if the link where the packet is located is the k-th hop from the source of the packet. When a link is activated to transmit packets, packets with a small hop-class will be transmitted first; and packets with the same hop-class will be transmitted in a FIFO fashion under PLQ-MWS.

Proposition 2: PLQ-MWS is throughput-optimal.

Remarks: The proof follows the same line of analysis for HQ-MWS using fluid limit techniques and induction method. Since a link transmits packets according to their hop-classes, we can view packets with hop-class-k at link l as in a sub-queue $q_{l,k}$. We can sequentially show: i) stability of shadow queues, ii) stability of the first-hop-class sub-queues $q_{l,1}$, and iii) stability of all other sub-queues by induction.

Note that PLQ-MWS is different from HQ-MWS, although they appear to be similar. HQ-MWS makes scheduling decisions based on the queue length of each per-hop queue. This may result in a waste of service if a per-hop queue is activated but does not have enough packets to transmit, even though the other per-hop queues for the same link have packets. In contrast, PLQ-MWS makes decisions based on the queue length at each link and this type of inefficiency does not happen. The performance difference due to this phenomenon will be illustrated through simulations in Section VI.

V. Scheduling with FIFO Per-Link Queues

In this section, we develop a scheduling scheme called **FLQ-MWS** by combining the LQ-MWS algorithm developed in the previous section with FIFO queueing discipline (instead of priority queueing discipline), and show that FLQ-MWS is throughput-optimal in networks where flows do not form loops. FLQ-MWS requires neither per-flow information nor per-hop information. We start with some useful definitions.

Recall that $\mathcal{L}(s)$ denotes the single, fixed, and loop-free route of flow s.

Definition 1: Two flows $s_1, s_2 \in \mathcal{S}$ are *connected*, if they have common links on their routes, i.e., $\mathcal{L}(s_1) \bigcap \mathcal{L}(s_2) \neq \emptyset$, and *disconnected*, otherwise. A sequence of flows $\{\tau_1, \cdots, \tau_n\}$ is a *communicating sequence*, if every two adjacent flows τ_i and τ_{i+1} are connected with each other. Two flows s_1 and s_2 *communicate*, if there exists a communicating sequence between s_1 and s_2.

Definition 2: Let $\mathcal{S}(l) \subseteq \mathcal{S}$ denote the set of flows passing through link l, and let $\mathcal{S}(\mathcal{Z}) \triangleq \bigcup_{l \in \mathcal{Z}} \mathcal{S}(l)$ denote the set of flows passing through a set of links $\mathcal{Z} \subseteq \mathcal{E}$. A non-empty set of links $\mathcal{Z}$ is called a *component*, if the following conditions are all satisfied:

1) $\mathcal{Z} = \bigcup_{s \in \mathcal{S}(\mathcal{Z})} \mathcal{L}(s)$.
2) Either $|\mathcal{S}(\mathcal{Z})| = 1$, or any two flows $s_1, s_2 \in \mathcal{S}(\mathcal{Z})$ communicate.
3) If $s_1' \in \mathcal{S}(\mathcal{Z})$ and $s_2' \notin \mathcal{S}(\mathcal{Z})$, then s_1' and s_2' do not communicate.
4) There is no proper subset $\mathcal{Z}' \subset \mathcal{Z}$ that satisfies conditions 1)-3).

Definition 3: Consider a component $\mathcal{Z}$, a sequence of flows $\{s_1, s_2, \cdots, s_N\} \subseteq \mathcal{S}(\mathcal{Z})$, where $N \geq 2$, is said to form a *flow-loop*, if one can find two links $l_{i_n}^{s_n}$ and $l_{j_n}^{s_n}$ for each $1 \leq n \leq N$, satisfying

1) $i_n < j_n$ for each $1 \leq n \leq N$,
2) $l_{j_n}^{s_n} = l_{i_{n+1}}^{s_{n+1}}$ for each $n < N$,
3) $l_{j_N}^{s_N} = l_{i_1}^{s_1}$.

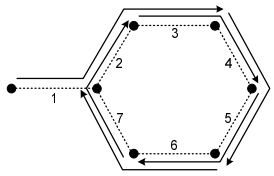
Definition 4: A component $\mathcal{Z}$ is called a *flow-tree*, if $\mathcal{Z}$ does not contain any flow-loops.

Definition 5: Consider a component $\mathcal{Z}$, a link $l \in \mathcal{Z}$ is called a *starting link*, if there exists a flow $s' \in \mathcal{S}(\mathcal{Z})$ such that $H_{l,1}^{s'} = 1$ and $H_{l,k}^{s} = 0$ for all other $s \in \mathcal{S}(\mathcal{Z})$ and all $k \geq 2$, i.e., a starting link has only exogenous arrivals. Similarly, a link $l \in \mathcal{Z}$ is called an *ending link*, if there exists a flow $s'' \in \mathcal{S}(\mathcal{Z})$ such that, $H_{l,\mathcal{L}(s'')}^{s''} = 1$, and $H_{l,k}^{s} = 0$ for all other $s \in \mathcal{S}(\mathcal{Z})$ and all $k < \mathcal{L}(s)$, i.e., an ending link transmits only packets that will leave the system immediately. A path $P = \{l_{P,1}, l_{P,2}, \cdots, l_{P,len(P)}\}$, where $len(P)$ denotes the length of path P and $l_{P,i}$ denotes the i-th hop link of P, is called a *flow-path*, if the following conditions are satisfied:

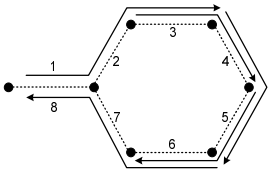
1) Links $l_{P,1}$ and $l_{P,len(P)}$ are the only starting and ending link on the path P, respectively.
2) Either $len(P) = 1$, or for each $1 \leq i < len(P)$, there exists a flow s such that, $l_{P,i} \in \mathcal{L}(s)$ and $l_{P,i+1} \in \mathcal{L}(s)$, i.e., two adjacent links $l_{P,i}$ and $l_{P,i+1}$ are on the route of some flow.

In general, a flow-tree consists of multiple (possibly overlapped) flow-paths. An illustration of flow-loop, flow-path, and flow-tree is presented in Fig. 1. It is clear from Definition 3 that, if there exists a flow-loop in a component, this component must contain a cycle of links, while the opposite is not necessarily true. For example, the components in Figs. 1(b) and 1(c) both contain a cycle, while neither of them contains a flow-loop.

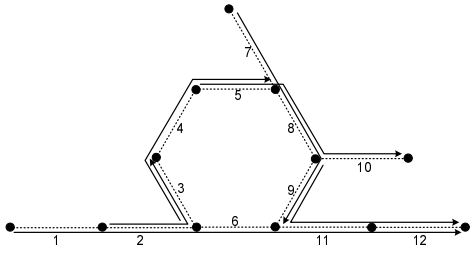
Since we assume $\sum_s \sum_{k=1}^{|\mathcal{L}(s)|} H_{l,k}^{s} \geq 1$ for all $l \in \mathcal{E}$, a network graph $\mathcal{G}$ can be decomposed into multiple disjoint components. We want to show that FLQ-MWS is throughput-optimal in networks, where flows do not form loops, or equivalently, where all the components are flow-trees. Before presenting the main result of this section, we describe Algorithm 1, which is used to assign a rank to each link of a flow-tree such that packet arrivals of a link of the flow-tree are either exogenous or from the links with a smaller rank. Algorithm 1 is important to proving the stability of a feasible system consisting of flow-trees under FLQ-MWS.



(a) A component containing a flow-loop



(b) A flow-tree with one flow-path



(c) A flow-tree with four flow-paths

Fig. 1. Examples of different types of components. Links and flows are denoted by dashed lines with numbers and solid lines with arrows, respectively. Note that two numbers labeled beside a dashed line stand for two links with opposite directions, e.g., links 1 and 8 in Fig. 1(b). In Fig. 1(a), all flows together forms a flow-loop $\{2, 3, 4, 5, 6, 7\}$, and the component is not a flow-tree. In Fig. 1(b), the component is a flow-tree and consists of one single flow-path: $\{1, 2, 3, 4, 5, 6, 7, 8\}$. In Fig. 1(c), the component is a flow-tree and consists of five flow-paths: $P_1 = \{1, 2, 3, 4, 5, 8, 10\}, P_2 = \{1, 2, 6, 11, 12\}, P_3 = \{7, 8, 10\}, P_4 = \{7, 8, 9, 11, 12\}$ and $P_5 = \{1, 2, 3, 4, 5, 8, 9, 11, 12\}$.

Let $\mathcal{E}(P)$ denote the set of links belonging to flow-path P. Let $\mathcal{T}$ denote a flow-tree, and let $\mathcal{P}(\mathcal{T})$ denote the set of all flow-paths in $\mathcal{T}$, i.e., $\mathcal{P}(\mathcal{T}) \triangleq \{\mathcal{E}(P) \subseteq \mathcal{T} \mid P \text{ is a flow-path}\}$. Let $P_k(\mathcal{T})$ denote the flow-path chosen in the k-th while-loop when running Algorithm 1 for $\mathcal{T}$, and let $\mathcal{P}_k(\mathcal{T}) \triangleq \bigcup_{j<k} P_k(\mathcal{T})$. Let $r(l)$ denote the rank of link $l \in \mathcal{T}$, and let $\mathcal{P}(l)$ denote the set of flow-paths passing through link l, i.e., $\mathcal{P}(l) \triangleq \{P \in \mathcal{P}(\mathcal{T}) \mid l \in \mathcal{E}(P)\}$. Let $\Gamma_k(l) \triangleq \{l' \in \bigcup_{P \in \mathcal{P}(l) \bigcap \mathcal{P}_k(\mathcal{T})} \mathcal{E}(P) \mid r(l') > r(l)\}$ denote the set of links that belong to the flow-paths of $\mathcal{P}(l) \bigcap \mathcal{P}_k(\mathcal{T})$ (i.e., flow-paths that pass through link l and are chosen in the j-th while-loop for $j < k$) and have a rank greater than $r(l)$.

The details of ranking are provided in Algorithm 1. In line 2, we do initialization by setting the rank of all links of $\mathcal{T}$ to -1. In lines 4-21, we pick a flow-path $P \in \mathcal{P}'$, and assign a rank to each link of P starting from link $l_{P,1}$. We may update a link's rank if we already assigned a rank to that link. The set of flow-paths $\mathcal{P}'$ is updated in line 20. The while-loop continues until $\mathcal{P}'$ becomes empty. We set $count = 1$ in line 6, and assign a rank to links $l_{P,i}$ for each $1 \leq i \leq len(P)$. For each link $l_{P,i}$, we consider the following three cases: 1) $r(l_{P,i}) = -1$; 2) $r(l_{P,i}) \geq count$; 3) $0 < r(l_{P,i}) < count$.

Case 1): link $l_{P,i}$ has not been assigned a rank yet. We set $r(l_{P,i}) = count$ in line 9.

Case 2): link $l_{P,i}$ already has a rank that is no smaller than the current $count$. In this case, the rank does not need an update, and we set $count = r(l_{P,i})$ in line 11.

Case 3): link $l_{P,i}$ already has a rank that is smaller than the current $count$. In this case, we update the rank of link $l_{P,i}$ (in line 13) and ranks of some other links. Specifically, for all the links $l \in \Gamma_k(l_{P,i})$, i.e., links that belong to the flow-paths in $\mathcal{P}(l) \bigcap \mathcal{P}_k(\mathcal{T})$ and have a rank greater than $r(l_{P,i})$, we increase their ranks by $count - r(l_{P,i})$ in lines 14-16. After considering all three cases, we increase the value of $count$ by 1 in line 18.

Algorithm 1 Rank Assignment

```
1:  procedure AssignRank(T)
2:      r(l) ← −1 for all l ∈ T
3:      P' ← P(T)
4:      while P' ≠ ∅ do
5:          pick a flow-path P ∈ P'
6:          count ← 1
7:          for 1 ≤ i ≤ len(P) do
8:              if r(l_{P,i}) = −1 then
9:                  r(l_{P,i}) ← count
10:             else if r(l_{P,i}) ≥ count then
11:                 count ← r(l_{P,i})
12:             else
13:                 r(l_{P,i}) ← count
14:                 for all l ∈ Γ_k(l_{P,i}) do
15:                     r(l) ← r(l) + (count − r(l_{P,i}))
16:                 end for
17:             end if
18:             count ← count + 1
19:         end for
20:         P' ← P'\{P}
21:     end while
22: end procedure
```

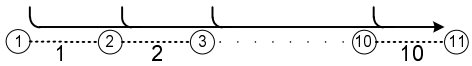
The intention of this ranking is to assign a rank to each link such that the ranks are monotonically increasing when one traverses any flow-path from its starting link. Algorithm 1 may give different ranking to a given flow-tree depending on the order of choosing flow-paths. We give two examples for illustration in our online technical report [12].

Note that Algorithm 1 is only for analysis purpose and it is not used in actual link scheduling. We claim the following lemma.

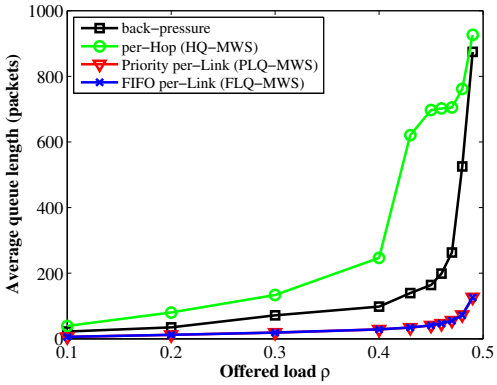
Lemma 3: Algorithm 1 assigns a rank to each link of flow-tree $\mathcal{T}$ such that for any flow-path $P \in \mathcal{P}(\mathcal{T})$, the ranks are monotonically increasing when one traverses the links of P from $l_{P,1}$ to $l_{P,len(P)}$, i.e., $r(l_{P,i}) < r(l_{P,i+1})$ for all $1 \leq i < len(P)$ and for any $P \in \mathcal{P}(\mathcal{T})$.

Corollary 4: Algorithm 1 assigns a rank to each link of flow-tree $\mathcal{T}$ such that packet arrivals at a link are either exogenous or from the links with a smaller rank.

Remarks: Corollary 4 follows immediately from Lemma 3. Motivated by Corollary 4, we extend our analysis for HQ-MWS and show in Proposition 5 that FLQ-MWS is throughput-optimal in networks without flow-loops. Note that different queueing disciplines are applied only to data queues, and shadow queues exhibit similar behaviors in terms of stability under LQ-MWS. Hence, the fluid limit model for the sub-system of shadow queues is stable under FLQ-MWS.



(a) Linear network topology with ten links



(b) Average queue length

Fig. 2. Performance of back-pressure, HQ-MWS, PLQ-MWS and FLQ-MWS in a linear network topology ($\epsilon = 0.01$).

Corollary 4 implies that packet arrivals of links with rank 1 are all exogenous, then we can prove the stability of these data queues by showing that the instantaneous arrival rate is less than the instantaneous service rate. Since Corollary 4 also implies that packet arrivals of links with rank 2 are either exogenous or from links with rank 1, we can similarly show the stability of links with rank 2. Repeating the above argument, we can prove the stability of all data queues by induction, which completes the proof of Proposition 5. Corollary 6 then follows immediately from Proposition 5, because a tree network itself does not contain a cycle of links.

Proposition 5: FLQ-MWS is throughput-optimal in networks where flows do not form loops.

Corollary 6: FLQ-MWS is throughput-optimal in tree networks.

VI. Numerical Results

In this section, we evaluate different scheduling schemes through simulations. We compare scheduling performance of HQ-MWS, PLQ-MWS, FLQ-MWS with the original back-pressure algorithm and the Liu algorithm provided in [8] under the *node-exclusive*[1] interference model. Note that we focus on the node-exclusive interference model for illustration purpose. Our scheduling schemes can be applied to a variety of interference constraints as specified in Section II.

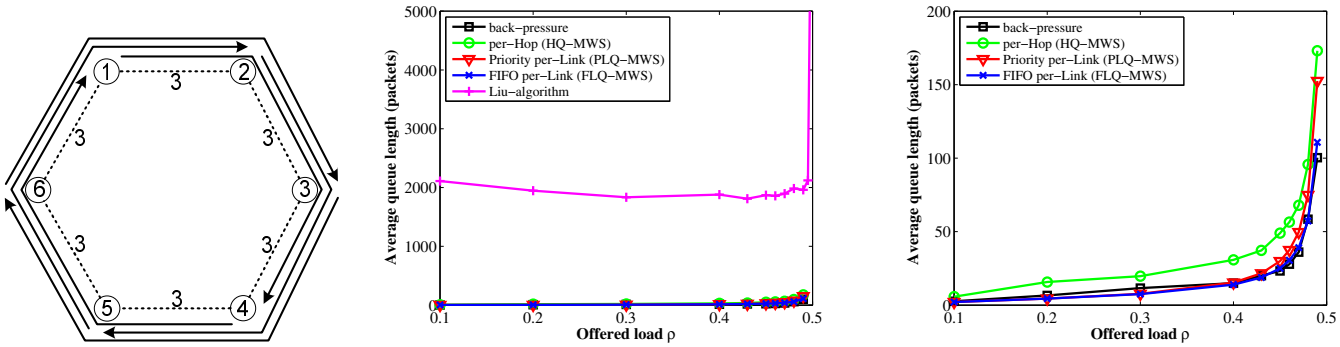
First, we evaluate and compare the scheduling performance of different schemes in a linear network that consists of 11 nodes and 10 links as shown in Fig. 2(a), where nodes are represented by circles and links are represented by dashed lines with link capacity, respectively. We establish 10 flows that are represented by arrows, where each flow i is from node i to node 11 via all the nodes in-between. We consider uniform traffic where all flows have packet arrivals at each time slot following Poisson distribution with the same mean rate $\rho > 0$. We run our simulations with changing traffic load ρ. Clearly, in this scenario, any traffic load with $\rho < 0.5$ is feasible. We use $\epsilon = 0.01$ for HQ-MWS, PLQ-MWS and FLQ-MWS. We evaluate the scheduling performance by measuring average queue lengths in the network over time.

Fig. 2(b) illustrates average queue lengths under different offered loads to examine the performance limits of different scheduling schemes. Each result represents an average of 10 independent simulation runs, where each run lasts for 10^7 time slots. Since the optimal throughput region is defined as the set of arrival rates under which queue lengths remain finite, we can consider the traffic load, under which the queue length increases rapidly, as the boundary of the optimal throughput region Λ^*. Fig. 2(b) shows that all schemes achieve the same boundary (i.e., $\rho < 0.5$), which supports our theoretical results on throughput optimality. However, we observe that HQ-MWS has worse delay performance than back-pressure, while PLQ-MWS and FLQ-MWS achieve substantially better performance. Note that under the back-pressure algorithm, the queue lengths have to build up along the route of a flow from the destination to the source, and in general, earlier hop link has a larger queue length. This leads to poor delay performance especially when the route of a flow is lengthy. Further, packet transmissions are more efficient under PLQ-MWS and FLQ-MWS, since they do not waste service as long as there are enough packets at the activated link, while the back-pressure algorithm and HQ-MWS maintain multiple queues for each link, and may waste service if the activated queue has less packets than the link capacity. HQ-MWS has larger delays than the back-pressure algorithm because the scheduling decisions of HQ-MWS are based on the shadow queue lengths rather than the actual queue lengths: a queue with very small (or even zero) queue length could be activated. This introduces another type of inefficiency in HQ-MWS. Note that PLQ-MWS and FLQ-MWS also make scheduling decisions based on the shadow queue lengths. However, their performance improvement from a single queue per link dominates delay increases from the inefficiency. These imply that maintaining per-link queues not only simplifies the data structure, but also improves scheduling efficiency and reduces delays.

Next, we evaluate the performance of the schemes, including the Liu algorithm[2] of [8], in a size-6 ring network as shown in Fig. 3(a). We establish 6 flows that are represented by arrows, where each flow i is from node i to node $(i+3) \mod 6$ via intermediate nodes $(i+1) \mod 6$ and $(i+2) \mod 6$, where we slightly abuse the notations by setting 6

[1]It is also called the *primary* or *1-hop* interference model, where two links sharing a common node cannot be activated simultaneously. It has been known as a good representation for Bluetooth or FH-CDMA networks [2].

[2]In the Liu algorithm, the service of a per-flow queue at node n for flow f at time slot t is (5) of [8]: $\tilde{X}_f^n(t) = (1+\gamma)\frac{\sum_{\tau=1}^{t} a_f^n(\tau)}{t}$, where $\tilde{X}_f^n(t)$ is not necessarily an integer number. Letting $\tilde{X}_f^n(t)$ be the expected service rate, we generate a Poisson random number with rate $\tilde{X}_f^n(t)$, as the service of the per-flow queue, instead of using $\tilde{X}_f^n(t)$ itself as the instantaneous service. Also, we let $\gamma = \epsilon$.



(a) Size-6 ring network topology (b) Average queue length (c) Average queue length (zoom in)

Fig. 3. Performance of the Liu algorithm, back-pressure, HQ-MWS, PLQ-MWS and FLQ-MWS in a size-6 ring network topology ($\epsilon = 0.01$).

$\mod 6 = 6$. Clearly, in this scenario, any traffic load with $\rho < 0.5$ is feasible. We observe that the Liu algorithm performs substantially worse than the other algorithms. This is due to the reason that per-flow queues of the Liu algorithm play a similar role as regulators, and keep packets from moving to the per-link queues. In addition, in our experiments, per-link queues often remain empty, while per-flow queues of the intermediate nodes hold a large number of packets, which implies that a packet needs to wait for a long time before it is being served at each intermediate link.

In Fig. 3(c), we zoom in to compare the performance of the other schemes depicted in Fig. 3(b), and hence do not compare the Liu algorithm here. Note that FLQ-MWS is not guaranteed to be throughput-optimal in this scenario since flows form a loop. However, the results indicate that all the schemes including FLQ-MWS empirically achieve the optimal throughput performance. This opens an interesting question about throughput performance of FLQ-MWS in general settings. The delay performance of the back-pressure algorithm is comparable to that of PLQ-MWS and FLQ-MWS in this scenario, which comes from the fact that the routes of flows are short and thus the queue length along the routes does not build up significantly. Also, the short routes of flows lead to a small number of per-hop queues at each link for HQ-MWS. This contributes to reducing the performance differences between HQ-MWS and the other schemes due to less chance of wasting service.

VII. Conclusion

In this paper, we develop scheduling policies with per-hop/per-link queues and a shadow algorithm to achieve the overall goal of removing per-flow or per-destination information requirement, simplifying queue structure, exploiting only local information, and potentially reducing delay. We show throughput optimality of the proposed schemes that use only the readily available per-hop information, using fluid limit techniques via an inductive argument. We further simplify the solution using FIFO queueing discipline with per-link queues and show that this is also throughput-optimal in in-tree types of networks (networks without flow-loops). The problem of proving throughput optimality in general networks with algorithms like FLQ-MWS that use only per-link information remains an important open and challenging problem.

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, no. 12, pp. 1936–1948, 1992.

[2] G. Sharma, R. R. Mazumdar, and N. B. Shroff, "On the complexity of scheduling in wireless networks," in *Proceedings of the annual international conference on Mobile computing and networking (MobiCom)*. ACM New York, NY, USA, 2006, pp. 227–238.

[3] L. Jiang and J. Walrand, "A distributed CSMA algorithm for throughput and utility maximization in wireless networks," *IEEE/ACM Transactions on Networking*, vol. 18, no. 3, pp. 960–972, 2010.

[4] J. Ni, B. Tan, and R. Srikant, "Q-CSMA: Queue-length based csma/ca algorithms for achieving maximum throughput and low delay in wireless networks," *Arxiv preprint arXiv:0901.2333*, 2009. [Online]. Available: http://arxiv.org/PS_cache/arxiv/pdf/0901/0901.2333v4.pdf

[5] S. Rajagopalan, D. Shah, and J. Shin, "Network adiabatic theorem: an efficient randomized protocol for contention resolution," in *The ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2009, pp. 133–144.

[6] L. Bui, R. Srikant, and A. Stolyar, "Novel architectures and algorithms for delay reduction in back-pressure scheduling and routing," *Arxiv preprint arXiv:0901.1312v3*, 2009. [Online]. Available: http://arxiv.org/PS_cache/arxiv/pdf/0901/0901.1312v3.pdf

[7] A. Stolyar, "Large number of queues in tandem: Scaling properties under back-pressure algorithm," *Arxiv preprint arXiv:1002.3940*, 2010.

[8] S. Liu, E. Ekici, and L. Ying, "Scheduling in Multihop Wireless Networks without Back-pressure," in *Proceedings of the Annual Conference on Communication, Control and Computing (Allerton)*, 2010.

[9] L. Ying, R. Srikant, and D. Towsley, "Cluster-based back-pressure routing algorithm," in *The IEEE International Conference on Computer Communications (INFOCOM)*, 2008, pp. 484–492.

[10] L. Ying, S. Shakkottai, and A. Reddy, "On combining shortest-path and back-pressure routing over multihop wireless networks," in *The IEEE International Conference on Computer Communications (INFOCOM)*, 2009, pp. 1674–1682.

[11] X. Wu, R. Srikant, and J. Perkins, "Scheduling Efficiency of Distributed Greedy Scheduling Algorithms in Wireless Networks," *IEEE Transactions on Mobile Computing*, pp. 595–605, 2007.

[12] B. Ji, C. Joo, and N. B. Shroff, "Scheduling with Per-link Queues and No Flow Information in Multi-hop Wireless Networks," *Arxiv preprint arXiv:1101.4211*, January 2011. [Online]. Available: http://arxiv.org/abs/1101.4211

[13] M. Bramson, "Stability of queueing networks," *Probability Surveys*, vol. 5, no. 169-345, p. 1, 2008.

[14] J. Dai, "On positive Harris recurrence of multiclass queueing networks: a unified approach via fluid limit models," *The Annals of Applied Probability*, pp. 49–77, 1995.