Improving the Delay Performance of CSMA Algorithms: A Virtual Multi-Channel Approach

Po-Kai Huang and Xiaojun Lin

***Abstract*—CSMA algorithms have recently received a significant amount of interest in the literature for designing efficient wireless control algorithms. CSMA algorithms are attractive because they incur low computation complexity and communication overhead, and can be shown to achieve the optimal capacity under certain assumptions. However, it has also been observed that CSMA algorithms suffer the starvation problem and incur large delay that may grow exponentially with the network size. In this paper, we propose a new algorithm, called Virtual-Multi-Channel (VMC-) CSMA, that can dramatically reduce delay without sacrificing the high capacity and low complexity of CSMA. The key idea of VMC-CSMA to avoid the starvation problem is to use multiple virtual channels to emulate a multi-channel system and compute a good set of feasible schedules simultaneously (without constantly switching/re-computing schedules). Under the protocol interference model and a single-hop utility-maximization setting, our proposed VMC-CSMA algorithm can approach arbitrarily close to the optimal total system utility, with both the number of virtual channels and the computation complexity increasing logarithmically with the network size. The VMC-CSMA algorithm inherits the distributed nature of CSMA algorithms. Further, once our algorithm converges to the steady-state, the expected packet delay for each link equals to the inverse of its long-term average rate, and the distribution of its head-of-line (HOL) waiting time can also be asymptotically bounded. Our simulation results confirm that the proposed VMC-CSMA algorithm indeed achieves both high throughput and low delay. Further, it can quickly adapt to network traffic changes.**

I. Introduction

A central problem to the design of wireless control algorithms is how to schedule the link transmissions in the presence of interference. Among the many design goals for wireless scheduling, there are three of them that are perhaps the most important. First, in order to support the increasing amount of the traffic placed on wireless networks, the control algorithm should achieve high capacity. Second, the packet delay should be small to meet the applications' service requirements. Third, for large networks, the control algorithms should have low computational complexity and low communication overhead, and preferably can be implemented in a distributed manner.

Existing algorithms in the literature achieve different tradeoffs among capacity, delay, and complexity. It is well known that max-weight types of algorithms [1] can attain the largest capacity region of the network, based on which a large number of wireless cross-layer control algorithms have been developed to optimize various performance objectives such as the system throughput, utility/fairness, and power efficiency [2,3]. However, for many problem settings the max-weight algorithm incurs exponentially high computational complexity as the network size increases. Further, it is a centralized algorithm that requires global information. There are a number of low-complexity algorithms that can be viewed as approximations to the max-weight schedule (see [2] and the reference within.) However, these algorithms can only guarantee a fraction of the optimal system capacity. Recently, a class of CSMA (Carrier Sense Multiple Access) algorithms were studied in [4,5]. CSMA algorithms are very attractive in the sense that they incur low computational complexity and are fully distributed, and can be shown to achieve the optimal capacity under certain assumptions. However, CSMA algorithms have been observed to have large delay that may grow exponentially with the network size [6,7], which makes the usefulness of the capacity gain questionable because most applications (even as simple as web-browsing) require some level of low delay.

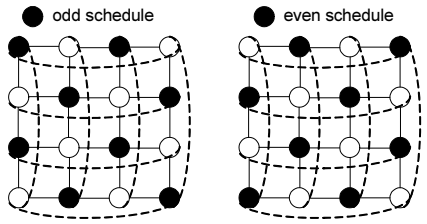


Fig. 1. A torus interference graph with the odd and even schedules.

In this paper, we are interested in developing improved version of the CSMA algorithms that can significantly reduce the delay without sacrificing the high capacity and low complexity. This remains a challenging problem in spite of a number of recent works that attempt to improve the delay of CSMA algorithms [5,7]–[14]. The main difficulty of reducing the delay of CSMA may be explained by the following example. Consider an n-by-n torus interference graph as shown in Fig. 1, where each circle represents a unit-capacity link, and the edge between two circles means that these two links can not transmit simultaneously. Suppose that the target rates of all links are 0.5. In a typical discrete-time version of the standard CSMA algorithms [5], it is possible to change the parameters so that the CSMA algorithms will stay at either the even schedule or the odd schedule with equal probability close to 0.5 to attain the target rates. However, once CSMA finds either the even or the odd schedule, it will be "locked" to this schedule for a long time before it can switch to the other schedule [7], and the corresponding inactive links will be starved of service in this period. This problem is known as the "starvation" problem for CSMA. Hence, it is difficult for CSMA to attain low delay.

Po-Kai Huang and Xiaojun Lin are with the School of ECE, Purdue University (Email: huang113@purdue.edu, linx@purdue.edu).

To resolve this problem, we will propose a new algorithm, called Virtual-Multi-Channel (VMC-) CSMA, that can achieve both high capacity and low delay with low complexity. The main novelty of our proposed algorithm to significantly reduce delay is to take advantage of multiple (physical- or virtual-) channels. This main idea can be explained as follows. Suppose that there are two separate channels for the same topology in Fig. 1, each with half the bandwidth of the original system. We can simply let one channel use the odd schedule and let the other channel use the even schedule. Obviously, each link can then achieve both the target rate of 0.5 as well as low delay (because each link is served at a constant rate at all time-slots). Note that the key idea here is to compute a set of good schedules for multiple channels at once, rather than computing one schedule at a time and constantly switching/recomputing schedules. Thus, the starvation problem is avoided. Note that although this idea seems to be quite natural, there are three main difficulties to generalize it to arbitrary networks. First, in many systems, we may only have one physical channel. How can we still reduce delay by using multiple channels? Second, even if we have more than one channels, how can we design a low-complexity and distributed algorithm to compute the right multi-channel schedules? Third, the number of channels in the above example is two, but it may not be sufficient for general topologies. Will the number of channels required to approach near-optimal performance be exceedingly large, which will then increase the complexity to a prohibitive level?

Our proposed VMC-CSMA algorithm precisely addresses these difficulties. First, for systems with only one physical channel, we introduce the notion of "virtual channels." Specifically, there are C virtual channels, and each virtual channel can have a different schedule. By randomly choosing a virtual channel and using the corresponding schedule at each time slot, we can then emulate the behavior of multiple physical channels. (See Section III-B for the corresponding distributed implementation.) Second, assuming that each link ℓ has a utility function $U_\ell(R_\ell)$ of its rate R_ℓ, our proposed VMC-CSMA algorithm iteratively updates the schedules across all virtual channels to optimize the total system utility. Further, it only requires local information exchange and incurs a low complexity that increases linearly with the number of virtual-channels C. Third, we quantify the throughput and delay performance of the VMC-CSMA algorithm with respect to C. Specifically, for an arbitrary network topology, let L denote the total number of links. We show that when $\epsilon \leq 0.1$ and the number of virtual-channels C is larger than $\frac{2\log L}{3\epsilon^2}$, our algorithm can allocate an expected rate vector $\vec{R} = [R_\ell]$ to each link such that $\sum_{\ell=1}^{L} U_\ell(R_\ell) \geq (1-\epsilon)\sum_{\ell=1}^{L} U_\ell([R_\ell^* - \epsilon]^+)$, where $\vec{R}^* = [R_\ell^*]$ is the rate vector with maximum system utility. Thus, our algorithm can achieve close-to-optimal system utility with the number of channels (and hence the corresponding complexity) increasing very slowly ($\mathcal{O}(\log L)$) with the network size. For delay, we show that, once our algorithm converges to the steady-state, the expected packet delay of link ℓ equals to $1/R_\ell$, and the distribution of its head-of-line (HOL) waiting time can also be asymptotically bounded (see Lemma 5 for details). Our simulation results confirm that the proposed VMC-CSMA algorithm indeed achieves both high throughput and low delay. Further, it can quickly adapt to network traffic changes. In summary, the proposed VMC-CSMA algorithm achieves dramatically low delay without sacrificing the high capacity and low complexity of standard CSMA algorithms.

The standard CSMA algorithm has been extended to multi-channel networks in [15]. However, the delay issue was not discussed. There have been a number of recent studies that try to quantify and improve the delay performance of CSMA algorithms [7]–[14]. However, none of them can attain the same level of throughput/delay performance as we reported here. The work in [13] compares the delay performance as one tunes the parameter of a class of CSMA algorithms. However, it is unclear whether such tuning will fundamentally alter the exponential order of the starvation time. CSMA algorithms with deadline constraints are studied under the setting of a complete graph in [12] (where each link interferes with every other link). In another work [7], the authors propose to periodically reset or "unlock" the schedule to an initial empty schedule. They show that, for a torus interference graph like Fig. 1, the delay can be made independent of the network size. However, it seems difficulty to generalize the results from [7,12] to arbitrary network topologies. [10,11] show that, if the offered load is sufficiently small (as a function of the maximum degree of the interference graph), both the starvation time and the delay of CSMA can be reduced to $\mathcal{O}(\log L)$ [10] or even $\mathcal{O}(1)$ [11]. However, for such results to hold, the offered load must be reduced significantly from the optimal capacity. Similarly, [14] considers using multiple physical channels for CSMA, but under the constraint that each link can occupy at most one channel. Thus, the resulting capacity could also be far from optimal. Our idea of using multiple channels is also inspired by [16]. However, it is difficult to modify the algorithm to optimize global system utility. In [9], the authors propose to effectively partition the network into (random) pieces with finite size and run CSMA in each piece. Hence, the expected packet delay can be bounded by a function of the size of each piece and is independent of the network size. However, in order to approach the optimal capacity, the size of each partition must be large. Thus, the actual delay in each partition may still be large. Finally, [8] shows that, for an open-loop setting (i.e., without congestion control), there exists worst-case topologies such that, in order to even support a small fraction of the optional system capacity, either the computational complexity or the delay must grow exponentially with the network size. Our results do not contradict with that of [8] because both our notion of delay (steady-state delay vs. transient delay) and our system setting (with vs. without congestion control) are different (see the end of Section III-C for detailed discussions).

The rest of the paper is organized as follows. The system model is presented in Section II. In Section III, we present the VMC-CSMA algorithm along with the performance analysis. We discuss the implementation issues in Section IV and the

simulation results in Section V. Then we conclude.

II. System Model

We consider a wireless network with N nodes and L links, where each node represents a communication device, and each link corresponds to a pair of transmitting node and receiving node. We assume the so-called protocol interference model, i.e., two links interfere with each other if they can not transmit data at the same time. We let $\mathcal{E}_\ell$ be the set of links that interfere with link ℓ. We will call two links ℓ_1 and ℓ_2 neighbors if ℓ_1 and ℓ_2 interfere with each other. We consider a time-slotted system, where each slot has unit length. We assume that the wireless network has only one physical channel. The capacity of each link is assumed to be 1, i.e., each link can transmit one unit-sized packet in one time slot if there are no interfering links transmitting at the same time. At each time slot, we define the term *schedule* as the set of links that transmit packets. We say that a schedule is *feasible* if no two links in the set interfere with each other. To represent a schedule, we will use a L-dimension vector such that the ℓ^{th} element is 1 if link ℓ is included in the schedule, and is 0 otherwise.

Associated with each link is a one-hop flow, i.e., packets of the flow will immediately leave the network after it traverses the link. We assume that each flow is infinitely backlogged, i.e., at the application layer it always has packets to send. Further, each flow at link ℓ has a utility function $U_\ell(\cdot)$ associated with it. If the long-term average rate of link ℓ is R_ℓ, then $U_\ell(R_\ell)$ represents the satisfactory level of the corresponding flow [2]. We assume that each utility function is positive, non-decreasing, strictly concave, and twice differentiable [17]. Further, we assume that it is bounded on the domain $D = [0, 1]$. Let $U(\vec{R}) = \sum_{\ell=1}^{L} U_\ell(R_\ell)$. Let Ω denote the capacity region of the wireless network, which is given by the set of all rate vectors $\vec{R} = [R_\ell]$ such that there exists a control policy that can support the long-term average rate R_ℓ at all links ℓ.

As we discussed in Section I, we are interested in both high capacity and low delay. For high capacity, we aim to solve the following optimization problem:

$$\max_{R_\ell \geq 0} \sum_{\ell=1}^{L} U_\ell(R_\ell), \ \vec{R} = [R_\ell] \in \Omega. \tag{1}$$

Let $\vec{R}^* = [R_\ell^*]$ be the optimal solution of the above optimization problem. Note that this problem is known to be a cross-layer control problem [2] because it involves two control mechanisms. First, the application layer of the flow at each link ℓ needs to determine how packets can be injected at the long-term average rate of R_ℓ^*, which can be viewed as a congestion control component. Second, at the MAC layer, the system must determine how to schedule the link transmissions to support the long-term average rate R_ℓ^* at all links. As we discussed in Section I, although this problem has been extensively studied in the literature, existing solutions suffer either from high complexity, low capacity (i.e., low utility), or large delay.

To define the delay in this infinite-backlog setting, we note that, for each packet, there is a time when the congestion control component at the transport layer decides to inject the packet into the buffer of the corresponding link. Then, at a later time, this packet is transmitted by the link. We define the delay of the packet as the difference between these two time instants. In other words, the packet delay is defined as the number of time-slots that a packet needs to stay in the buffer before being served. Although this definition seems to be a natural definition of delay, it unfortunately does not fully capture the effect of the possible starvation problem [7]. For example, consider a queue with a single buffer. A new packet is added to the buffer immediately after the old packet is served. Suppose that the time to serve each packet follows the pattern below. For every 1000 packets, it takes only one time-slot to serve each of the first 999 packets. However, the 1000^{th} packet suffers starvation for 1000 time-slots. In this case, the expected packet delay (average over all packets) is a low value of 1.99. Thus, the effect of the starvation problem is not obvious. Due to this reason, we introduce another notion of delay as follows. At a given time t, we study the time that the head-of-line (HOL) packet has waited in the system. In the above example, the expected HOL waiting time at any given time would be around 250. Hence, the negative impact of the starvation problem is more obvious. In this paper, by low delay, we mean that both the packet delay and the HOL waiting time should be small. Thus, the goal in this paper is to develop low-complexity and distributed control algorithms that can achieve both provably high system utility and provably low delay.

III. VMC-CSMA Algorithm Design and Analysis

In this section, we propose a new low-complexity Virtual-Multi-Channel (VMC-) CSMA algorithm, with provable high capacity and low delay. Since our algorithm is motivated by the standard CSMA algorithm, we will first briefly describe a discrete time version of the standard CSMA algorithm [2,5] for solving problem (1) and discuss its weakness. We will then introduce the new VMC-CSMA algorithm.

A. The Standard CSMA algorithm

Let $Q_\ell(t)$ be the number of packets of link ℓ at the beginning of time slot t. Let $\vec{S}(t) = [S_\ell(t)]$ be the schedule chosen by the scheduling algorithm at time t. Define a set $\mathcal{S}$ of *decision schedules* such that every element in $\mathcal{S}$ is a feasible schedule. Further, each link must be scheduled by at least one schedule in $\mathcal{S}$. Let $w_\ell(t)$ be an suitable increasing function of $Q_\ell(t)$ as in [5]. For example, $w_\ell(t) = \log(\alpha Q_\ell(t))$, where α is a positive constant. The following CSMA algorithm is known to solve problem (1) under certain assumptions.

CSMA Algorithm: At each time t,

- *Decision Phase:* Choose a decision schedule $\vec{S}^D(t) = [S_\ell^D(t)]$ randomly in $\mathcal{S}$.
- *Scheduling Phase:* For each link ℓ, if $S_\ell^D(t) = 1$ and $S_{\ell'}(t) = 0$ for every link $\ell' \in \mathcal{E}_\ell$, $S_\ell(t)$ is determined with random distribution: $P(\{S_\ell(t) = 1\}) = \frac{\exp(w_\ell(t))}{1+\exp(w_\ell(t))}$ and $P(\{S_\ell(t) = 0\}) = \frac{1}{1+\exp(w_\ell(t))}$. If $S_\ell^D(t) = 0$ or $S_{\ell'}(t) = 1$ for any link $\ell' \in \mathcal{E}_\ell$, let $S_\ell(t) = S_\ell(t-1)$.

- *Congestion Control:* Each link ℓ injects a random number of packets according to a Poisson random variable with mean $r_\ell = \arg\max_{r\geq 0}\{U_\ell(r) - \beta Q_\ell(t)r\}$, where β is a positive constant.

It can be shown that, under a time-scale separation assumption [5], if β is sufficiently small, $\vec{r} = [r_\ell]$ will converge to values close to the optimal solution of (1) [2]. In practice, the above time-scale separation assumption must be approximated by a small value of α. However, when α and β are small, $Q_\ell(t)$ and $w_\ell(t)$ tend to grow to large values. In that case, the algorithm will likely to be stuck in one good schedule for a long time before it switches to another good schedule [7] (as we discussed in Section I). In the worst case, one can show that the starvation time and the delay may grow exponentially with the network size [6].

B. Virtual-Multi-Channel CSMA Algorithm

We now introduce our proposed VMC-CSMA algorithm that overcomes the above difficulties of starvation problem and large delay. As we have mentioned in Section I, the key to achieve both high capacity and low delay is to take advantage of C multiple channels and simultaneously compute C feasible schedules over all channels. Once a good set of C feasible schedules are found, we can then avoid constantly switching schedules (and hence the starvation problem). To make this idea feasible in wireless systems with only one physical channel, next we will use the concept of "virtual channels." Specifically, there are C virtual channels for each link ℓ. For each link ℓ, we use $\vec{V}_\ell = [V_{\ell 1}, \cdots, V_{\ell C}]$ to denote its schedule in all virtual channels, where $V_{\ell k} = 1$ if link ℓ is scheduled in the k^{th} virtual channel, and $V_{\ell k} = 0$ otherwise. We use $\vec{V} = [\vec{V}_\ell]$ to denote the global schedules of all virtual channels and all links. When we focus on a specific virtual-channel k, there is a feasible schedule $\vec{S}(\vec{V})_k = [V_{1k}, \cdots, V_{Lk}]$ for the network. Note that given the global schedule $\vec{V}$, the total number of virtual channels used by link ℓ is given by $x_\ell(\vec{V}) = \sum_{k=1}^{C} V_{\ell k}$. To use these schedules, at each time slot, the network randomly chooses a virtual-channel $k(t)$ uniformly from 1 to C, *i.i.d.* across time-slots. All links in the network then use the feasible schedule $\vec{S}(\vec{V})_{k(t)}$ in this time-slot, i.e., each link ℓ transmits a packet if $V_{\ell,k(t)} = 1$.[1] Note that each link only needs to know its own schedules $\vec{V}_\ell$. Further, the randomization of the virtual-channel $k(t)$ can be achieved in a distributed manner if all links are synchronized, and they have the same random-number generator with a common-seed, which only needs to be agreed upon at the beginning of the system operation. With this implementation, each link ℓ will be scheduled for actual transmission with probability equal to $r_\ell(\vec{V}) = x_\ell(\vec{V})/C$, *i.i.d.* across time-slots. Thus, the long-term average rate of link ℓ will be equal to $r_\ell(\vec{V})$, and the inter-service time is $1/r_\ell(\vec{V})$. Hence, the delay will likely be small as we will show below.

[1] We note that the idea of using a random schedule has some similarity to the stationary randomized policy [3] that has been used to analyze the throughput optimality of other algorithms. However, there are no low-complexity methods of computing the right stationary randomized policy.

It remains to develop a low-complexity and distributed algorithm for computing the global schedule $\vec{V}$ that leads to high total system utility. Specifically, we seek solution to the following optimization problem, which can be viewed as an approximation to the original optimization problem (1):

$$\begin{aligned} &\max_{\vec{V}} \sum_{\ell=1}^{L} U_\ell(r_\ell(\vec{V})), \\ &\vec{S}(\vec{V})_k \text{ is a feasible schedule, } k = 1, 2, \cdots, C. \end{aligned} \tag{2}$$

Our hope is that, when C is sufficient large, the optimal solution of (2) will be close to that of (1).

We next describe our proposed low-complexity VMC-CSMA algorithm for solving (2). Later in Section III-C, we will study its throughput and delay as the number of virtual-channels C varies. Because VMC-CSMA algorithm updates the global schedule $\vec{V}$ iteratively over time, we will use the notation $\vec{V}(t)$, $\vec{V}_\ell(t)$, and $V_{\ell k}(t)$ to denote their corresponding values at time slot t. Similar to the standard CSMA algorithm, we define a set $\mathcal{S}$ of decision schedules such that it satisfies the following three conditions: (1) each decision schedule is a feasible schedule. (2) each link ℓ must be scheduled by at least one decision schedule in $\mathcal{S}$. Note that these two conditions are the same as those in the standard CSMA algorithm [5]. (3) each link scheduled by a decision schedule can broadcast C bits to its neighbors in a time slot. We will discuss in Section IV how to implement the decision schedules at each time slot. Now, we describe our algorithm.

Virtual-Multi-Channel CSMA Algorithm: At each time t,

- *Decision Phase:* Choose a decision schedule $\vec{S}^D(t) = [S_\ell^D(t)]$ randomly in $\mathcal{S}$.
- *Update Phase:* For each link ℓ, if $S_\ell^D(t) = 0$, let $\vec{V}_\ell(t) = \vec{V}_\ell(t-1)$. Otherwise, perform Algorithm 1.

Algorithm 1 Update phase for link ℓ if $S_\ell^D(t) = 1$:

Choose a permutation $(n_1^\ell, n_2^\ell, \cdots, n_C^\ell)$ of the set $\{1, 2, \cdots, C\}$ uniformly at random. Let $f_\ell(x) = \exp(\alpha U_\ell(\frac{x}{C}))$ and $x_\ell^1 = \sum_{k=1}^{C} V_{\ell k}(t-1)$.
for $i = 1$ to C **do**
  if $V_{\ell', n_i^\ell}(t-1) = 1$ for any link $\ell' \in \mathcal{E}_\ell$ **then**
    $V_{\ell, n_i^\ell}(t) = V_{\ell, n_i^\ell}(t-1)$.
  else
    $V_{\ell, n_i^\ell}(t)$ is determined with the random distribution:

$$\begin{aligned} &P(\{V_{\ell, n_i^\ell}(t) = y\}) \\ = \ &\frac{f_\ell(x_\ell^i + y - V_{\ell, n_i^\ell}(t-1))}{f_\ell(x_\ell^i - V_{\ell, n_i^\ell}(t-1)) + f_\ell(x_\ell^i + 1 - V_{\ell, n_i^\ell}(t-1))}, \end{aligned} \tag{3}$$

    where $y = 0$ or 1.
  end if
  $x_\ell^{i+1} = x_\ell^i + V_{\ell, n_i^\ell}(t) - V_{\ell, n_i^\ell}(t-1)$.
end for
Link ℓ broadcasts $\vec{V}_\ell(t)$ to all of its neighbors.

- *Scheduling Phase:* A common virtual-channel $k(t)$ is chosen by all links in the network uniformly at random from 1 to C, and each link ℓ transmits a packet if $V_{\ell,k(t)}(t)=1$.
- *Congestion Control:* All links ℓ use the window-based flow-control algorithm with window-size 1, i.e., a new packet is injected to link ℓ only if a packet is served.

We note a key difference between the VMC-CSMA and the standard CSMA. Under the VMC-CSMA (correspondingly standard CSMA) algorithm, the random distribution for updating the schedule of link ℓ is a function of its own utility $U_\ell(x_\ell/C)$ (correspondingly its own queue length $Q_\ell(t)$). The significance of this key difference is as follows. Consider (3) and $V_{\ell,n_i^\ell}(t-1)=0$ for an example, we have

$$P(\{V_{\ell,n_i^\ell}(t)=1\})=\frac{f_\ell(x_\ell^i+1)/f_\ell(x_\ell^i)}{1+f_\ell(x_\ell^i+1)/f_\ell(x_\ell^i)}\approx\frac{\exp(\frac{\alpha}{C}U_\ell'(\frac{x_\ell^i}{C}))}{1+\exp(\frac{\alpha}{C}U_\ell'(\frac{x_\ell^i}{C}))} \tag{4}$$

Recall that $U_\ell(\cdot)$ is a strictly concave function, which implies that $U_\ell'(\cdot)$ is a strictly decreasing function. Hence, if a link has larger x_ℓ^i value, it is less likely to activate itself in a new virtual channel, and vice versa. Thus, the schedules across virtual channels are in fact coordinated to reach a state $\vec{V}$ with total system utility close to the optimal value. Hence, VMC-CSMA do not constantly change the schedule and avoid the starvation problem. An additional benefit is that we can use the window-based flow-control as the congestion control component, which further controls the packets in the system and reduces the delay.

C. Utility Optimality and Delay Performance

In this subsection, we study the capacity/utility and delay performance when our VMC-CSMA algorithm reaches steady state. Note that, under the VMC-CSMA algorithm, the global schedule $\vec{V}(t)$ behaves as a Markov chain. Hence, we will also refer to $\vec{V}(t)$ as the state. Let $\mathcal{V}$ be the state space of the Markov chain. We first introduce a proposition that describes the stationary distribution of the Markov chain.

Proposition 1: The stationary distribution of the Markov chain $\vec{V}(t)$ is given by $P(\vec{V})=\frac{1}{Z}\exp(\alpha\sum_{\ell=1}^L U_\ell(r_\ell(\vec{V})))$, where Z is a normalization constant for all $\vec{V}\in\mathcal{V}$.

Proof: Proposition 1 can be proved by checking that the local balance equation holds. Details are given in [18]. ∎

Proposition 1 implies that the state $\vec{V}$ with larger value of total system utility has a higher chance to be visited. Further, since α appears in the exponent in the stationary distribution, as α increases, the probability of reaching the state with the largest utility, which is the solution to problem (2), will approach 1. However, due to the quantization effect with a finite C, the optimal utility in (2) may be smaller than the optimal utility of the original problem (1). Hence, the key question is how many virtual-channels C we need such that the two optimal-utility values are close. We note that this is an important question because the complexity of our algorithm also increases with C. In practice, we would prefer a smaller value of C. A first thought for solving this question is to use the Carathodory's theorem [19]. Specifically, since the capacity region of the optimization problem in (1) is a convex hull of all the feasible schedules, the Caratheodory's theorem tells us that the optimal solution to (1) can be written as the convex combination of $L+1$ feasible schedules. As a result, we may guess that we need C to be at least $\Theta(L)$ so that the optimal solution to (1) can be approximated by a valid global schedule $\vec{V}$. Somewhat surprisingly, by allowing a small margin ϵ for error and using a novel probabilistic argument, we can reduce the order of C to $\mathcal{O}(\log L)$. This nice result is presented in the following proposition. Recall that R_ℓ^* is the optimal rate of link ℓ.

Proposition 2: If $\epsilon\le 0.1$ and $C>\frac{2\log L}{3\epsilon^2}$, then there exists a state $\vec{V}^s$ in the state space $\mathcal{V}$ of the Markov chain such that $r_\ell(\vec{V}^s)\ge R_\ell^*-\epsilon$, for all link ℓ.

Proof: The proof is given in Appendix A ∎

Note that since the rate is always bigger than zero, it then follows from $r_\ell(\vec{V}^s)\ge R_\ell^*-\epsilon$ that $r_\ell(\vec{V}^s)\ge\max(R_\ell^*-\epsilon,0)\triangleq[R_\ell^*-\epsilon]^+$. With the help of Proposition 2, we can prove the first main theorem in this paper. Recall that $U(\vec{r})=\sum_{\ell=1}^L U_\ell(r_\ell)$.

Theorem 3: Under our VMC-CSMA algorithm, for any $\epsilon\le 0.1$, we can choose $C>\frac{2\log L}{3\epsilon^2}$ and α large enough such that $P\{U(\vec{r}(\vec{V}(t)))\ge\sum_{\ell=1}^L U_\ell([R_\ell^*-\epsilon]^+)\}\ge 1-\epsilon$, where $\vec{R}^*$ is the optimal solution of problem (1).

Proof: Consider a fixed C. Let $\vec{V}^{\max}$ be the solution to (2). Then $U(\vec{r}(\vec{V}^{\max}))\ge U(\vec{r}(\vec{V}))$, for all $\vec{V}\in\mathcal{V}$. Define the set $A=\{\vec{V}\in\mathcal{V}|U(\vec{r}(\vec{V}))=U(\vec{r}(\vec{V}^{\max}))\}$. Denote $f(\vec{V})=\exp(\alpha U(\vec{r}(\vec{V})))$. Let $f^{\max}=f(\vec{V}^{\max})$. Note that $f(\vec{V})=f^{\max}$ for all $\vec{V}\in A$. Since $\mathcal{V}$ is finite for a fixed C, there must exist $\epsilon'>0$ such that $f(\vec{V})\le f^{\max}e^{-\alpha\epsilon'}$ for all $\vec{V}\notin A$. Using Proposition 1 and this inequality, we can then show that $P(\{\vec{V}\in A\})=\frac{\sum_{\{\vec{V}\in A\}}f(\vec{V})}{\sum_{\{\vec{V}\in A\}}f(\vec{V})+\sum_{\{\vec{V}\notin A\}}f(\vec{V})}\ge\frac{f^{\max}|A|}{f^{\max}|A|+f^{\max}e^{-\alpha\epsilon'}(|\mathcal{V}|-|A|)}$. This implies that for any ϵ, there exists α such that $P(\{\vec{V}\in A\})>1-\epsilon$. Further, from Proposition 2, for any $\epsilon\le 0.1$, we can choose a fixed $C>\frac{2\log L}{3\epsilon^2}$, such that for $\vec{V}\in A$, $U(\vec{r}(\vec{V}))\ge U(\vec{r}(\vec{V}^s))\ge\sum_\ell U_\ell([R_\ell^*-\epsilon]^+)$. The results then follows. ∎

Theorem 3 leads immediately to the following corollary on the average throughput and expected packet delay for each link ℓ.

Corollary 4: Let $P(\vec{V})$ be steady state probability when the state of the Markov chain $\vec{V}(t)$ is $\vec{V}$. The average throughput R_ℓ of link ℓ is $R_\ell=\sum_{\{\vec{V}\in\mathcal{V}\}}P(\vec{V})r_\ell(\vec{V})$, and the expected packet delay is $1/R_\ell$. Further, $U(\vec{R})\ge(1-\epsilon)U([\vec{R}^*-\epsilon]^+)$.

Proof: The claim on the utility follows from Theorem 3 and the fact that the utility function is concave. Further, since we use window-based flow control with window-size 1, the expected packet delay for each link ℓ follows from the Little's law to be $1/R_\ell$. Details can be found in [18]. ∎

As we have mentioned in Section II, we use the HOL waiting time as a measure to capture the effect of starvation. The following lemma shows that under our algorithm, the tail distribution of the HOL waiting time will decay quickly.

Lemma 5: Let $r_\ell^{\min}=\min_{\{\vec{V}\in\mathcal{V},U(\vec{r}(\vec{V}))=U(\vec{r}(\vec{V}^{\max}))\}}r_\ell(\vec{V})$, where $\vec{V}^{\max}$ is the solution to (2). Under our VMC-CSMA algorithm, for a fixed integer $d>0$ and any $\epsilon>0$, there exists sufficiently large α so that, for each link ℓ,

$P\{\text{HOL waiting time} \geq d\} \leq (1 - r_\ell^{\min})^d + \epsilon$.

Proof: Define the set $A = \{\vec{V} \in \mathcal{V} | U(\vec{r}(\vec{V}^{\max})) = U(\vec{r}(\vec{V}))\}$. From the proof of Theorem 3, we can find α such that $P\{\vec{V}(t) \notin A\} \leq \epsilon/d$. Let E be the event that $\vec{V}(t_1) \in A, t_1 = t - d + 1, \cdots, t$. Hence, by the union bound, $P(E^c) \leq d\frac{\epsilon}{d} = \epsilon$. Let $w_\ell(t)$ be the HOL waiting time for link ℓ at time t. We have that $P\{w_\ell(t) \geq d|E\} = P\{\text{link } \ell \text{ is not served for } t-d+1 \text{ to } t|E\} \leq (1 - r_\ell^{\min})^d$. Use this inequality, we can then show that $P\{w_\ell(t) \geq d\} = P\{w_\ell(t) \geq d|E\}P\{E\} + P\{w_\ell(t) \geq d|E^c\}P\{E^c\} \leq (1 - r_\ell^{\min})^d + \epsilon$. This concludes the proof. ∎

Note that, for each link ℓ, both the expected packet delay and the distribution of the HOL waiting time only depend on R_ℓ and $r_\ell^{\min}$, and are otherwise independent of the network size. It may still be possible for $r_\ell^{\min}$ to be small. For example, if the utility function of a link ℓ has a much smaller derivative than others, then this link may be assigned a very small rate in the optimal solution of (1). However, we believe that his effect has more to do with the system setting than the control algorithms. For reasonable topologies and utility functions, we would expect that $r_\ell^{\min}$ will not grow arbitrarily small as the network size grows. For instance, consider the torus interfere graph in Fig. 1. It is easy to see that $R_\ell^* = 0.5$ for all links if all links have the same utility function. Thus, as long as the number of virtual channels are even, we must have $r_\ell(\vec{V}^{\max}) = 0.5$. Hence, both the expected delay and the distribution of the HOL waiting time do not deteriorate with the network size.

We note that, in [8], the authors show that, for open-loop systems, even at an offered load that is a small fraction of the optimal capacity, there exists network topologies such that either the complexity or the delay must grow exponentially with the network size. We note that our result differs from [8] in two important aspects. First, the delay definition is different. We are interested in the steady state delay, i.e., after the Markov chain converges to the stationary distribution. The convergence time (which we did not capture in this paper) may still be exponential in the network size. In contrast, the delay in [8] is the worst-case delay across time and hence also captures the transient phase. Second, we use a closed-loop system (with congestion control) in contrast to an open-loop system in [8]. Hence, our results do not contradict with that of [8]. On the other hand, we believe that a low value of delay as we defined under the setting in this paper is useful in practice. Thus, the results in this paper suggests that the impossibility results in [8] may not prevent us to develop low-complexity, low-delay and high-capacity algorithms that are useful in practice.

D. Computational Complexity and Communication Overhead

In this subsection, we discuss the computational complexity and communication overhead of the VMC-CSMA algorithm. It is easy to see that, for each link, the computational complexity of the VMC-CSMA algorithm is $\mathcal{O}(C)$. This is because, for each link, it only needs to go through the C virtual channels. Further, finding the random permutation in the "update phase" is of complexity $\mathcal{O}(C)$ through the use of the Fisher-Yates shuffle [20]. Note that all of the computations can be carried out in parallel at all links. Regarding the communication overhead, note that each link ℓ can use a C-bit vector to denote its schedule $\vec{V}_\ell$ across all virtual channels. In each time-slot, a link scheduled by $\vec{S}^D(t)$ may need to broadcast this C-bit vector to all of its neighbors. Hence, the communication overhead is also $\mathcal{O}(C)$. From Theorem 3 and Corollary 4, we can see that $C = \mathcal{O}(\log L)$, which grows very slowly with the network size. Hence, our algorithm is scalable to large networks. In practice, all C bits may be put into a single control packet in one time-slot. For example, if the size of a control packet is 250 bytes, it can accommodate $C = 2000$ virtual channels. Even for a large network with $L = 1000$ links, $C = 2000$ virtual channels are sufficient to reduce ϵ to be as small as 0.05 (see Theorem 3). Hence, the corresponding capacity/utility reduction will be very small. If a data packet of length 2000 bytes (like in 802.11) is transmitted in each time-slot, such a control packet corresponds to a low overhead of 12.5%. In practice, this overhead can be further reduced by performing the VMC-CSMA updates once several time-slots. For example, if we perform the operation in the update phase every 5 time-slots, the communication overhead is further reduced to 2.5%. Note that, at each time-slot, we can still randomly choose a virtual channel and use its schedule for transmission. Thus, reducing the frequency of the VMC-CSMA updates will only affect the convergence of the algorithm, but it will not affect the capacity and delay once the Markov chain converges to its stationary distribution.

IV. Implementation

In this section, we discuss two implementation issues. We start with an improved scheduling algorithm. It is well known that, for CSMA algorithms, there is a trade-off between optimality and convergence speed, depending on the value of α. In practice, we want to choose a suitable α that is not too large to shorten the convergence time. However, when α is not very large, we also observe a common source of performance degradation, which can be explained as follows. Suppose that the Markov chain has found the optimal state $\vec{V}^{\max}$, and a link ℓ is active in virtual channel k. When α is not very large, there is a substantial probability that link ℓ will turn itself off in virtual channel k. If all other links in its neighborhood have interfering links that are active in virtual channel k, no neighboring links can be turned on in this virtual channel. Hence, link ℓ may turn itself on again in virtual channel k later, and, for link ℓ, the transmission opportunities during this period are lost unnecessarily. Such performance degradation can be easily avoided with the following improved algorithm. We call the schedule computed by VMC-CSMA as the soft schedule. In addition, we now introduce hard schedule as follows. Whenever a link ℓ is turned on in virtual channel k by the soft schedule (i.e. $V_{\ell k} = 1$), we also turn on link ℓ in virtual channel k in the hard schedule. However, even if $V_{\ell k} = 0$, we will only turn off link ℓ in the virtual channel k in the hard schedule when a neighboring link of link ℓ decides to use virtual channel k (i.e., it turns itself on in virtual channel k by the soft schedule). As a

result, the hard schedule will give up transmission opportunities until the last minute. Not only that our earlier throughput/delay results still hold, we can avoid the throughput loss due to the reason described above. Due to the space constraint, a complete description of the improved algorithm can be found in [18].

Second, we briefly discuss how to choose the decision schedule at each time slot. Recall in Section III-B that, at each time slot, we choose a decision schedule $\vec{S}^D$ with a fixed probability distribution from a set $\mathcal{S}$ of decision schedules. Further, the set $\mathcal{S}$ of decision schedules should meet the following three conditions: (1) A decision schedule is a feasible schedule. (2) Each link ℓ must be scheduled by at least one decision schedule. (3) Each link scheduled by the decision schedule can broadcast C bits to its neighbors. Note that, for the standard CSMA algorithms [5], the authors provide a random backoff algorithm for computing decision schedules that satisfy the first two conditions. There are a number of ways to satisfy the remaining third condition. One possibility is to assume that there is a separate control channel (e.g. using CDMA). On the other hand, if such a separate control channel is not available, the other possibility is to make the decision schedule more sparse. Specifically, not only that two active links in the decision schedule do not interfere with each other (and hence the decision schedule is a feasible schedule), but the two active links do not share any common neighbors. As a result, if a link is not scheduled by the decision schedule, it will not receive more than one broadcast transmission. Such a decision schedule can still be computed in each time-slot via a random-backoff-based algorithm similar to that of [5], with additional signaling message to resolve conflicts at common neighbors. For details, please refer to [18].

V. Simulation

We evaluate the VMC-CSMA algorithm in two topologies with our C++ simulator. Specifically, we simulate the improved scheduling algorithm and a random backoff-based decision-schedule algorithm discussed in Section IV. For both topologies, there is a one-hop flow associated with each link, and its utility function is given by $\log(10^{-5}+\cdot)-\log(10^{-5})$. Further, the simulation time is 15,000 slots.

We first study a 8-by-8 torus interference graph as described in Section I and compare our algorithm with the standard CMSA algorithm.[2] Note that, because of symmetry, the optimal rate R^*_ℓ for each link under this setting will be 0.5. For the standard CSMA algorithm, the weight is chosen as $w_\ell(t) = 0.5Q_\ell(t)$, and the parameter β for congestion control is 0.1. For our VMC-CSMA algorithm, we let $C = 30$ and $\alpha = 29$. The corresponding $\epsilon = 0.3$.[3] We denote the node in the top left corner of the torus as node 0 and the node right next to it as node 1. Note that node 0 is active in the odd schedule, and node 1 is active in the even schedule (See Fig. 1). The average throughput, average delay across packets, and average HOL waiting time across the time slots for node 0 are presented in the following table.

	throughput	delay	HOL waiting time
CSMA	0.427	159	372.8
VMC-CSMA	0.479	2.09	2.10

The result shows that our algorithm can indeed achieve throughput close to the optimal rate. Further, the delay performance is exactly equal to the inverse of the average throughput. In contrast, both the packet delay and the HOL waiting time of the standard CSMA algorithm are 80 and 170 times larger, respectively.[4] In Figure 2(a), we plot the tail distribution of the HOL waiting time under both algorithms. The HOL waiting time of our algorithm decays quickly, which confirms Lemma 5 and explains why the average HOL waiting time is small. In contract, the HOL waiting time of CSMA decays slowly (see Fig. 2(b)) due to the starvation problem.

To give a sense of the convergence time of our algorithm, in Fig. 2(c) we plot the time evolution of the instantaneous system utility $\sum_{\ell=1}^{L} U_\ell(r_\ell(\vec{V}(t)))$ under different α. The simulation results show that the utility approaches very close to the optimal utility after 100 time slot. Further, the utility is larger than the lower bound given by $(1-\epsilon)U([\vec{R}^*-\epsilon]^+)$ (Corollary 4) even for a small value of C. Note that when α is larger, the utility value after convergence is also closer to the optimal value of (1). However, larger α also incurs longer convergence time.

Before we proceed to a larger topology, we comment on the choice of α for different C. Through our simulation, we observe that a rule of thumb is to choose α to be proportional to C. The reason can be explained by equation (4). For a fixed value of $r_\ell = \frac{x^i_\ell}{C}$, as C increases, in order to maintain the same probability of adding another virtual channel, we should increase α proportional to C. Our simulation studies indicate that, as long as the ratio α/C is fixed, the tradeoff between convergence and optimality is roughly the same for different values of C. As this ratio increases, the optimality is improved at the cost of a longer convergence time. Thus, we will use this rule of thumb also in the following simulation results.

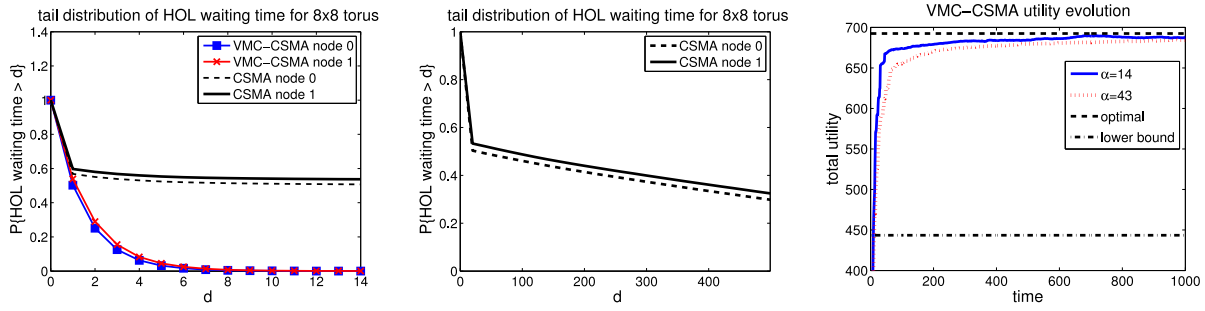
Next, we simulate our algorithm under a larger random topology with 100 nodes and 100 links. We set the maximum node degree to be 4, and under this constraint, each link is generated by randomly choosing two nodes.[5] We assume that each link will interfere with the links that are two-hop away (i.e., the two-hop interference model). In addition to the standard CSMA algorithm, we will also compare our algorithm with two other algorithms: the constant-time (CT) distributed algorithm [22] and the well-known maximum weighted matching algorithm (MWM) [2]. We caution however that, since these algorithms incur very different computational complex-

[2]Note that we use this simple and symmetric topology first because it allows us to easily compare with the optimal solution.

[3]Note that Theorem 3 requires $\epsilon = 0.1$ and a corresponding $C = 277$. In this simulation setting, we intentionally choose a small C to show that in reality we can also use small C to achieve good performance.

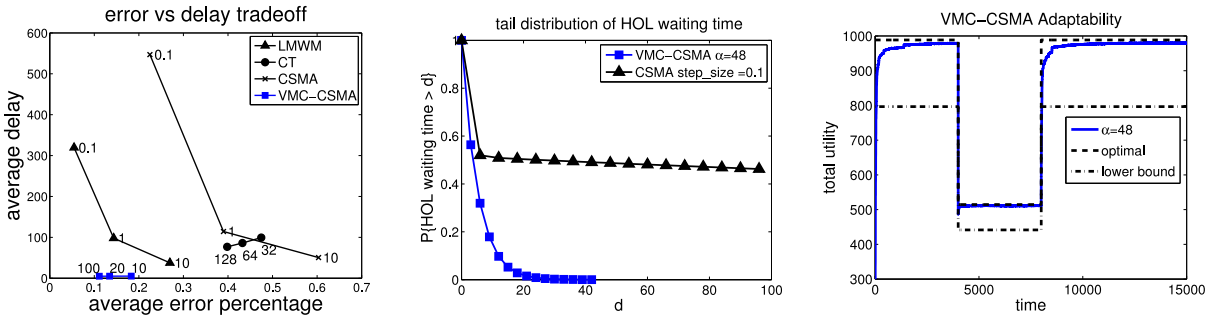
[4]Note that the expected packet delay is an average over packets, and the expected HOL waiting time is an average across time. As in the classical Inspector's Paradox [21], these two ways of taking expectation will lead to different values when the inter-service time is not memoryless, which is the case for the standard CSMA algorithm.

[5]We set a maximum on the node degree simply to prevent the case that the optimal rate of a node is very small.



(a) (b) (c)

Fig. 2. The simulation results for the 8-by-8 torus graph.



(a) (b) (c)

Fig. 3. The simulation results for a random network with 100 nodes and 100 links.

ity and communication overhead, it is difficult to conduct a completely fair comparison. Because our proposed VMC-CSMA algorithm, the standard CSMA algorithm, and the CT algorithm require only one round of local control-message exchange (including channel sensing) in each iteration, we will compare their performance directly. On the other hand, the MWM algorithm is a centralized algorithm that requires the queue length information of all links. Further, its complexity may be exponential under the two-hop interference model. In order to make the comparison slightly more fair, we simulate a version of the MWM algorithm that exchanges queue length information and computes the schedule once every 100 time-slots. We call this algorithm LMWM (Low-frequency MWM). Note that even at the reduced frequency, the LMWM algorithm is still very costly to implement due to its high complexity and the requirement of collecting global queue length information.

In Fig. 3(a), we report the throughput and delay tradeoff of each algorithm. Specifically, the y-axis is the average packet delay, and the x-axis is the average error percentage of the rate vectors computed by these algorithms under various parameter setting. The error percentage for each link is defined by $\max[\text{optimal\_rate} - \text{allocated\_rate}, 0]/\text{optimal\_rate}$. Thus, the most desirable algorithm will correspond to a point close to the origin, where it attains both high capacity and low delay. As we can see in Fig. 3(a), for the VMC-CSMA algorithms, when we vary C (which is labeled next to each point) and set $\alpha = C * 0.48$, the VMC-CSMA consistently achieves low error percentage and low delay. On the other hand, the performance curves for CSMA and LMWM (each point is labeled with the value of β, the step size used in the congestion control) are significantly worse. For CT algorithm, the curve (each point is labeled with the value of the backoff window size) exhibits large error percentage because CT can only achieve a fraction of the capacity region. Similar to the torus interference graph, we also report the tail distribution of the HOL waiting time. Specifically, we let $C = 100$ and choose a node where the optimal allocated rate is 0.2177. The result is shown in Fig. 3(b). It shows that the tail distribution under our algorithm again decays quickly. In contrast, the tail distribution of the HOL waiting time under CSMA algorithm does not decay even after 100. Finally, we simulate our algorithm under the possibility of traffic changes and observe how our algorithm adapts with the change. Specifically, we let $C = 100$ and $\alpha = 48$. Further, we will turn off the traffic on half of the links after 4000 time-slots and turn on the traffic after 8000 time-slots. Then, we record the evolution of the total utility compared with the optimal utility computed offline. The result is shown in Fig. 3(c). We can see our algorithm adapts well to the traffic changes and the instantaneous utility quickly approaches to the optimal utility.

VI. Conclusion

In this paper, we propose a virtual-multi-channel (VMC-) CSMA algorithm to improve the delay performance of the standard CSMA algorithms for wireless networks under the protocol interference model and a single-hop utility-maximization

setting. The key idea of VMC-CSMA is to resolve the starvation problem of standard CSMA algorithms by emulating a multi-channel system with C virtual channels and computing multiple feasible schedules simultaneously. VMC-CSMA inherits the distributed nature of CSMA, and the complexity of each link grows linearly with C. We use a novel probabilistic argument to show that VMC-CSMA can approach arbitrarily close to the optimal total system utility with C (and hence the complexity) increasing logarithmically with the network size. Further, after our algorithm converges to the steady-state, the expected packet delay for each link equals to the inverse of its long-term average rate, and the distribution of its head-of-line (HOL) waiting time can also be asymptotically bounded. Our simulation results show that VMC-CSMA significantly improve the delay performance of CSMA algorithms. In the future work, we will study how to extend this novel idea to the wireless networks with arrivals and multi-hop traffic.

Acknowledgement: The work is partially supported by the National Science Foundation through grant CNS-0643145, CNS-0721484, and CNS-0831999.

APPENDIX A

Proof: Recall that $\vec{R}^* = [R_\ell^*]$ is the optimal rate allocation of the system. By Caratheodory's Theorem [19], $\vec{R}^*$ can be represented as a convex combination of $L+1$ feasible schedules $\vec{S}_1, \vec{S}_2, \cdots, \vec{S}_{L+1}$, i.e., $\vec{R}^* = \sum_{i=1}^{L+1} \alpha_i \vec{S}_i$, where $\alpha_i \geq 0$ and $\sum_{i=1}^{L+1} \alpha_i = 1$. Now, consider the following random system. There are C trials, for each $k = 1, \cdots, C$ choose a $\vec{S}$ from $\vec{S}_1, \cdots, \vec{S}_{L+1}$ based on the probability distribution $P(\{\vec{S} = \vec{S}_i\}) = \alpha_i$, for all i, independently across k. Let r_ℓ^s be the number of times link ℓ is scheduled by this random set of C schedules divided by C. Then, in order to show that there exists $\vec{V}^s$ such that $r_\ell(\vec{V}^s) \geq R_\ell^* - \epsilon$, for all link ℓ, it is sufficient to show that the following event occurs with a positive probability: $R_\ell^* - r_\ell^s \leq \epsilon$ for all links ℓ. Note that by union bound we have $P(\bigcap_\ell \{R_\ell^* - r_\ell^s \leq \epsilon\}) \geq 1 - \sum_{\ell=1}^{L} P(R_\ell^* - r_\ell^s > \epsilon)$. Hence, it is sufficient to show that

$$P(R_\ell^* - r_\ell^s > \epsilon) < \frac{1}{L}, \text{ for all links } \ell. \tag{5}$$

Consider a fixed link ℓ. Intuitively, since the probability that link ℓ is active in each trial is equal to R_ℓ^*, *i.i.d.* across trials, (5) should hold when C is sufficiently large. Precisely, note that (5) holds trivially if $R_\ell^* \leq \epsilon$. Further, if $R_\ell^* = 1$, then link ℓ will be activated by all schedules $\vec{S}_i$, and $P(R_\ell^* - r_\ell^s > \epsilon) = 0$. Hence, we only need to consider the possibility that $\epsilon < R_\ell^* < 1$. Let h be a random variable that represents the number of times that link ℓ is activated by the series of C random schedules, and $r_\ell^s = \frac{h}{C}$. Note that h is a Binomial random variable, and its moment generating function is $E[e^{-\frac{ht}{C}}] = (R_\ell^* e^{-\frac{t}{C}} + 1 - R_\ell^*)^C$. Using the Chernoff bound, we then have, for all $t > 0$,

$$\begin{aligned} P(R_\ell^* - r_\ell^s > \epsilon) &= P(-\tfrac{h}{C} > \epsilon - R_\ell^*) \\ &\leq e^{t(R_\ell^* - \epsilon)} E[e^{-\frac{ht}{C}}] = e^{f(t)}, \end{aligned} \tag{6}$$

where $f(t) = t(R_\ell^* - \epsilon) + C\log(R_\ell^* e^{-\frac{t}{C}} + 1 - R_\ell^*), \forall t > 0$. Minimizing $f(t)$ over t, we obtain (see [18] for details)

$$P(R_\ell^* - r_\ell^s > \epsilon) \leq \left[\left(\frac{R_\ell^* - \epsilon}{R_\ell^*}\right)^{-(R_\ell^* - \epsilon)} \left(\frac{1 - R_\ell^*}{1 - R_\ell^* + \epsilon}\right)^{1 - R_\ell^* + \epsilon} \right]^C.$$

Hence, a sufficient condition of (5) to hold is

$$C > \frac{\log L}{(1 - R_\ell^* + \epsilon)\log(\frac{1 - R_\ell^* + \epsilon}{1 - R_\ell^*}) + (R_\ell^* - \epsilon)\log(\frac{R_\ell^* - \epsilon}{R_\ell^*})}. \tag{7}$$

We can verify that the denominator of the right hand side of (7) is no smaller than $\frac{3}{2}\epsilon^2$ for all $\epsilon \leq 0.1$ and $\epsilon < R_\ell^* < 1$ (see details in [18]). Thus, $C > \frac{2\log L}{3\epsilon^2}$ is sufficient for the results to hold. ■

REFERENCES

[1] L. Tassiulas and A. Ephremides, "Stability Properties of Constrained Queueing Systems and Scheduling Policies for Maximum Throughput in Multihop Radio Networks," *IEEE Trans. on Automatic Control*, vol. 37, no. 12, pp. 1936–1948, December 1992.

[2] X. Lin, N. B. Shroff, and R. Srikant, "A Tutorial on Cross-Layer Optimization in Wireless Networks," *IEEE JSAC*, Aug. 2006.

[3] 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.

[4] L. Jiang and J. Walrand, "A Distributed CSMA Algorithm for Throughput and Utility Maximization in Wireless Networks," *IEEE/ACM Trans. on Networking*, vol. 18, no. 3, pp. 960 – 972, June 2010.

[5] 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," in *IEEE INFOCOM*, 2010.

[6] C. Borgs, J. Chayes, A. Frieze, J. H. Kim, P. Tetali, E. Vigoda, and V. H. Vu, "Torpid mixing of some Monte Carlo Markov chain algorithms in statistical physics," in *IEEE FOCS*, 1999.

[7] M. Lotfinezhad and P. Marbach, "Throughput-Optimal Random Access with Order-Optimal Delay," in *IEEE INFOCOM*, 2011.

[8] D. Shah, D. Tse, and J. N. Tsitsiklis, "Hardness of low delay network scheduling," in *Information Theory Workshop*, 2010.

[9] D. Shah and J. Shin, "Delay optimal queue-based CSMA," in *ACM SIGMETRICS*, 2010.

[10] L. Jiang, M. Leconte, J. Ni, R. Srikant, and J. Walrand, "Fast mixing of parallel Glauber dynamics and low-delay CSMA scheduling," in *IEEE INFOCOM*, 2011.

[11] V. Subramanian and M. Alanyali, "Delay performance of CSMA in networks with bounded degree conflict graphs," in *IEEE ISIT*, 2011.

[12] B. Li and A. Eryilmaz, "A Fast-CSMA Algorithm for Deadline-Constrained Scheduling over Wireless Fading Channels," *Technical Report, http://arxiv.org/abs/1203.2834*, 2012.

[13] C.-H. Lee, D. Y. Eun, S.-Y. Yun, and Y. Yi, "From Glauber Dynamics To Metropolis Algorithm: Smaller Delay in Optimal CSMA," in *IEEE ISIT*, 2012.

[14] K. Lam, C. Chau, M. Chen, and S. Liew, "Mixing Time and Temporal Starvation of General CSMA Networks with Multiple Frequency Agility," in *IEEE ISIT*, 2012.

[15] A. Proutiere, Y. Yi, T. Lan, and M. Chiang, "Resource Allocation over Network Dynamics without Timescale Separation," in *IEEE INFOCOM*, 2010.

[16] Y. Yi, G. de Veciana, and S. Shakkottai, "Learning contention patterns and adapting to load/topology changesin a mac scheduling algorithm," in *IEEE WiMesh*, 2006.

[17] S. Boyd, *Convex Optimization*. Cambridge University Press, 2004.

[18] P.-K. Huang and X. Lin, "Improving the Delay Performance of CSMA Algorithms: A Virtual Multi-Channel Approach," *Technical Report, Purdue University*, 2013. [Online]. Available: http://docs.lib.purdue.edu/ecetr/441/

[19] D. P. Bertsekas, A. Nedic, and A. E. Ozdaglar, *Convex Analysis and Optimization*. Belmont, MA: Athena Scientific, 2003.

[20] D. E. Knuth, *The Art of Computer Programming, volume 2, (3rd ed.): Seminumerical Algorithms*. Addison-Wesley, 1997.

[21] S. M. Ross, *Stochastic Processes*, 2nd ed. New York: John Wiley & Son, 1996.

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