Tuning Up the Performance of Constant-Time Distributed Scheduling Algorithms via Majorization

Han Cai   and   Do Young Eun

Dept. of ECE, North Carolina State University, Raleigh, NC 27695

Email: {hcai2, dyeun}@ncsu.edu

Abstract— **Scheduling algorithms assign contention probability for each link in Wireless Ad Hoc Networks and plays a key role in deciding the system performance. Recently, many low-cost distributed scheduling algorithms are proposed. In this paper, we propose to improve the performance of a class of** *distributed collision-based scheduling algorithms*, **called constant-time distributed scheduling algorithms, by exploring the advantage brought by the** *unevenness* **of links' contention probabilities. Specifically, we prove that there exists ordering relationship for the success probability of any neighborhood when the contention probability vectors are ordered in the sense of** *majorization*. **We show how to modify the existing algorithms so as to find a new contention probability vector that majorizes the original one** *in a distributed manner*. **Our simulation results indicate that by using our modified algorithms, the average queue-lengths of a stable system can be reduced by** 25% **to** 50%**, while keeping the capacity region the same. Our modification to the existing algorithms is extremely simple and entails essentially zero additional cost.**

I. INTRODUCTION

Scheduling plays an important role in determining the WANET performance. Collision-free scheduling algorithms, such as those based on TDMA or maximal matching [12], [10], [3], [2], [11], [7], [1], [13], generally offer higher throughput (or larger capacity region), but also require either centralized implementation or high computational complexity [5], thus are not scalable. On the other hand, collision-based algorithms are more suitable for distributed implementation with low complexity. In particular, [4], [6], [8] have proposed constant-time distributed scheduling algorithms[1] with provable guaranteed performance.

Capacity region and queue-length are two critical metrics to measure the performance of scheduling algorithms. Suppose there are N links in the system with capacity C_l of link l ($l = 1, 2, \ldots, N$). Let λ_l be data generation rate at link l and $q_l(n)$ denote the queue-length of link l at the nth time slot. The set of $\{\lambda_l\}$ leading to finite $\{q_l\}$ forms the *capacity region* [8]. Existing results [4], [6], [8] focus on the capacity region of the constant-time distributed scheduling algorithm, i.e., how much input traffic can the system absorb while maintaining all queue-lengths to be finite? While this provides the stability region of the system (all queues remain finite), the following question remains unanswered: *how about the system performance inside the capacity region*? In other words, although existing scheduling algorithms ensure stability and focus on enlarging the stability region (capacity region), *can we further improve the system performance (or decrease*

the queue-length) when the system is stable, i.e., when all the queues are only guaranteed to be just finite?
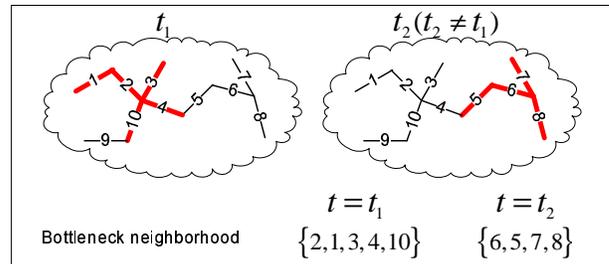

Fig. 1. Time-varying bottleneck neighborhood in multi-neighborhood network: Assign a weight to each link. Bottleneck neighborhood can be defined as the neighborhood with the maximum weight sum. Due to the change in the incoming traffic and user movement, the link weight may change over time. Thus, the bottleneck neighborhood also change over time.

In order to address the question above, we first consider the notion of time-varying *bottleneck neighborhood*. Define link l's neighborhood as a set of all links interfering (depending on the interference model chosen) with l plus itself. For example, in Figure 1, link 1's neighborhood is $\{1, 2\}$. In a *single-neighborhood* system where all links interfere with each other, there is only one neighborhood: $\{1, 2, \cdots, N\}$. Figure 1 shows a *multi-neighborhood* system where a subset of links (belonging to different neighborhoods) can transmit at the same time. If link l has weight $w_l(n)$ at the nth time slot (as function of $q_l(n)$, C_l, etc.), then *Bottleneck neighborhood* is defined as the neighborhood with the maximum weight sum.

In this paper, we show that by slightly modifying existing constant-time distributed scheduling algorithms [4], [6], [8], we can make the contention probability vector[2] of the bottleneck neighborhood in multi-neighborhood networks *more uneven*[3] in the sense of *majorization* [9], *in a distributed way*. Specifically, we prove that the probability of successful channel assignment (no collision) for any given neighborhood (e.g., bottleneck neighborhood) is increased when the contention probability vector is replaced by a 'more uneven' one that majorizes the original contention probability vector. Our modified algorithm is fully distributed and autonomous in the sense that performing our algorithms at each link – without knowing whether it currently belongs to the changing bottleneck neighborhood – automatically leads to higher

---

[1]Time to schedule channel(s) does not grow with the network size.

[2]The contention probability vector comprises of the contention probability of all links in the neighborhood under consideration.

[3]For example, $\{0.1, 0.9\}$ is more uneven than $\{0.5, 0.5\}$. See Section II-B for details.

success probability of the unknown bottleneck neighborhood, which helps reduce queue-lengths at the bottleneck within the stability region. We also provide simulation results showing that the performance of bottleneck neighborhood, as well as that of multi-neighborhood system, is significantly improved in the sense that the average queue-length decreases by 25% to 50% while the capacity region remains the same, with essentially *no* additional cost introduced by our modification to those in [4], [6], [8].

II. PRELIMINARIES

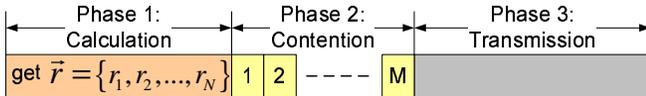### A. Distributed Contention-based Scheduling Algorithms

Fig. 2.   Three phases of contention-based scheduling algorithm: (i) calculate contention probability r_l; (ii) random backoff: users contend for the channel; (iii) transmission (if no collision in phase 2).

Constant-time distributed scheduling algorithms [4], [6], [8] have three phases in one frame, as shown in Figure 2. In the first phase, contention probability r_l is calculated for user (link) l based on its queue size, weight, etc. The second phase includes $M \geq 1$ time slots. User l chooses to contend on some time slot(s) with certain probability (as a function of its own contention probability r_l) if no one has already taken the channel. If only a single user contend for channel at the ith time slot ($i = 1, 2, \cdots, M$), then this user successfully gets the channel and can immediately begin Phase 3 – data transmission. There is no data transmission in this frame if no user contend during all M time slots or collision happens at any time slot.

Specifically, algorithms in [4], [8] works as follows. In Phase 1, contention probability r_l ($l = 1, 2, \cdots, N$) for each user l is decided; in Phase 2, user l chooses a random number $j \in \{1, 2, \cdots, M\}$ with probability $f_j(r_l)$ and contends for the channel at the jth time slot. On the other hand, the algorithm in [6] is slightly different from [4], [8], only in Phase 2. In this case, a user l contends for the channel at the first time slot with probability $g_1(r_l)$ and skips with probability $1 - g_1(r_l)$. If the first time slot is skipped, it then contends for the channel at the second time slot with probability $g_2(r_l)$, and so on.

We note that the algorithm in [6] can be made equivalent to those in [4], [8] by setting

$$f_i(r_l) = g_i(r_l) \prod_{j=1}^{i-1} (1 - g_j(r_l)), \quad (i = 1, 2, \cdots, M) \quad (1)$$

for each user l. Hence, from now on, our analysis will be based on the first system model (algorithms in [6] with $f_j(\cdot)$).

B. Majorization Theory

Majorization captures the notion of 'unevenness' of vectors. For any $\vec{x} = \{x_1, x_2, \ldots, x_n\} \in \mathcal{R}^n$, let $x_{[1]} \geq x_{[2]} \geq \ldots \geq$ $x_{[n]}$ denote the components of $\vec{x}$ in decreasing order (e.g., $x_{[1]} = \max\{x_1, \ldots, x_n\}$ and $x_{[n]} = \min\{x_1, \ldots, x_n\}$).

Definition 1: [9] For $\vec{x}, \vec{y} \in \mathcal{R}^n$, we say that $\vec{x}$ is majorized by $\vec{y}$ (or $\vec{y}$ majorizes $\vec{x}$) and write $\vec{x} \prec \vec{y}$, if

$$\sum_{i=1}^{k} x_{[i]} \leq \sum_{i=1}^{k} y_{[i]}, \quad k = 1, \ldots, n-1,$$

and $\sum_{i=1}^{n} x_{[i]} = \sum_{i=1}^{n} y_{[i]}$.   □

For example, if $\vec{x} = \{0.2, 0.2, 0.2\}$ and $\vec{y} = \{0.1, 0.3, 0.2\}$, we have $\vec{x} \prec \vec{y}$, i.e., $\vec{y}$ is 'more uneven' than $\vec{x}$.

Definition 2: [9] A real-valued function ϕ defined on $\mathcal{R}^n$ is said to be Schur-convex on $\mathcal{R}^n$ if for all $\vec{x}, \vec{y} \in \mathcal{R}^n$ with $\vec{x} \prec \vec{y}$ we have $\phi(\vec{x}) \leq \phi(\vec{y})$.   □

Remark 1: The Schur convex function is order-preserving, i.e, the order relationship between $\vec{x}$ and $\vec{y}$ does not change by taking the function.

III. EFFECT OF MAJORIZATION RELATIONSHIP ON CHANNEL ASSIGNMENT

In this section, we mathematically prove that for any single neighborhood (e.g., bottleneck neighborhood), the probability of successful channel assignment can be improved through using more uneven contention probability vector in the sense of majorization.

Consider a single neighborhood with N users interfering with each other. Given the contention probability vector $\vec{r}$, we define by $R_i(\vec{r})$ the probability that exactly one user contends for the channel at the ith time slot (success probability). Then, from the notations in Section II-A, we have

$$R_1(\vec{r}) = \sum_{j=1}^{N} \left[f_1(r_j) \prod_{k \neq j} (1 - f_1(r_k)) \right],$$

where $\prod_{k \neq j} (1 - f_1(r_k))$ is the probability that no user except j chooses to contend on the first time slot. Similarly, for any time slot $i \in \{1, 2, \cdots, M\}$, note that $\sum_{n=1}^{i} f_n(r_k)$ is the probability that user k chooses to contend for the channel in one of time slots $1, 2, \ldots, i$. Thus, we have

$$R_i(\vec{r}) = \sum_{j=1}^{N} \left[f_i(r_j) \prod_{k \neq j} \left(1 - \sum_{n=1}^{i} f_n(r_k) \right) \right]. \quad (2)$$

Finally, the success probability for channel assignment (over any of the M time slots) is

$$R(\vec{r}) = \sum_{i=1}^{M} R_i(\vec{r}). \quad (3)$$

We are now ready to present our main result here.

Theorem 1: Given a contention probability vector $\vec{r} = \{r_1, r_2, \cdots, r_N\}$, suppose that user j contends for the channel at the ith ($i \in \{1, 2, \cdots, M\}$) time slot with probability $f_i(r_j)$ of the form:

$$f_i(x) = e^{-\alpha_{i-1} x} - e^{-\alpha_i x}, \quad (4)$$

where $\{\alpha_i\}$ is *any* arbitrary nondecreasing sequence with $\alpha_0 = 0$, i.e., $0 = \alpha_0 \le \alpha_1 \le \cdots \le \alpha_{M-1}$. Then, if there exists a vector $\vec{s}$ such that $\vec{r} \prec \vec{s}$, we have

$$R(\vec{r}) \le R(\vec{s}), \tag{5}$$

where the function $R(\cdot)$ is defined in (3). $\qquad\square$

Proof: See Appendix. $\qquad\blacksquare$

Remark 2: Note that $\sum_{i=1}^{M} f_i(\cdot)$, i.e., the probability for a user to contend on some time slot, is always increasing (nondecreasing), while each $f_i(\cdot)$ needs not be so. In addition, by assigning $\alpha_i = i/M$, we get the same algorithm as in [4].

As mentioned in Remark 2, Theorem 1 covers a larger class of algorithms including the one in [4]. For this class of algorithms, Theorem 1 shows that the success probability of channel assignment for any single neighborhood, which could be bottleneck neighborhood in multi-neighborhood networks, is increased by using a new contention probability vector $\vec{s}$ that majorizes the original one $\vec{r}$. In the next section, we focus on how to find the vector $\vec{s}$ in practice in a distributed way.

IV. MODIFIED CONSTANT-TIME DISTRIBUTED SCHEDULING ALGORITHMS

In this section, we explain how to modify existing constant-time distributed scheduling algorithms [4], [6], [8] for performance improvement. Specifically, we show how to find a 'more uneven' vector $\vec{s}$ (that majorizes $\vec{r}$) for the unknown and possibly time-varying bottleneck neighborhood in multi-neighborhood networks in a *distributed way* at *any time slot*.

A. How to make the contention probability vector of any single neighborhood more uneven?

We consider the algorithm in [4]. Let $q_l(n)$ and C_l denote the queue size and capacity of link l, respectively. Let $\mathcal{I}_i$ be link i's neighborhood, i.e., a set including link i and all links that cannot transmit with link i at the same time. At the beginning of nth time slot (channel contention), link l computes:

$$r_l(n) = \alpha \cdot \frac{q_l(n)/C_l}{\max_{i \in \mathcal{I}_l}[\sum_{k \in \mathcal{I}_i}(q_k(n)/C_k)]}, \tag{6}$$

where $\alpha = \log(M)$ is constant. Then, each link chooses a time slot for contention from $i \in \{1, 2, \cdots, M+1\}$ ($M+1$ implies no contention) with probability f_i given by

$$f_i = e^{-\frac{i-1}{M}r_l} - e^{-\frac{i}{M}r_l} \ (i = 1, 2, \ldots, M), \text{ and } f_{M+1} = e^{-r_l}.$$

In single-neighborhood case, all links interfere with each other, i.e., link l's neighborhood is $\mathcal{I}_l = \{1, 2, \cdots, N\}$, regardless of l. Thus, the denominator on the RHS of (6) does not change with l.

How to find a 'more uneven' vector that majorizes $\vec{r}(n) = \{r_1(n), \ldots, r_N(n)\}$? One straightforward and fair way would be to apply the same (universal) function to the contention probability of *each individual user* $r_l(n)$. In particular, we want to find some function ϕ such that

$$\frac{\{x_1, x_2, \cdots, x_N\}}{\sum_{i=1}^{N} x_i} \prec \frac{\{\phi(x_1), \phi(x_2), \cdots, \phi(x_N)\}}{\sum_{i=1}^{N} \phi(x_i)}. \tag{7}$$

From Proposition B.2. in [9], the class of convex function $\phi(x) = x^\gamma$ with $\gamma \ge 1$ satisfies (7). In addition, as γ increases, the vector becomes more uneven, i.e., becomes 'larger' in majorization relationship.
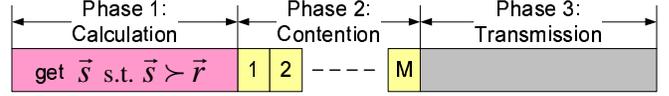

Fig. 3. Our modified algorithm: We only change Phase 1 in the original algorithm (as shown in Figure 2) by calculating new contention probability vector $\vec{s}$ that majorizes the original one.

Figure 3 illustrates our *modified algorithm*, in which we change (6), the contention probability vector calculated in Phase 1, and keep Phase 2 the same. From the discussion right after (7), we set the *new contention probability for link l at time n* as:

$$r_l^{New}(n) = \alpha \cdot \frac{(q_l(n)/C_l)^\beta}{\max_{i \in \mathcal{I}_l}[\sum_{k \in \mathcal{I}_i}(q_k(n)/C_k)^\beta]}, \tag{8}$$

where $\beta \ge 1$ is some constant whose selection will be discussed later in Section IV-B.

Since $\mathcal{I}_l$ doesn't depend on l for a single neighborhood, it follows that $\vec{r}(n) \prec \vec{r}^{New}(n)$, i.e., the new contention probability vector $\{r_l^{New}(n), l = 1, \ldots, N\}$ is more uneven than the original one $\{r_l(n), l = 1, \ldots, N\}$. Note that the only additional cost of calculating the new vector is taking power function $(\cdot)^\beta$ at each link, which is essentially zero.

The algorithms in [8], [6] rely on similar steps as in (6) to calculate $r_l(n)$ in Phase 1, except that different constant α is used. Hence, our modification to those in [8], [6] are again given by the transformation from (6) to (8), with constant $\alpha = 1$ for [8] and $\alpha = (\sqrt{M} - 1)/2$ for [6]. Although algorithms in [8], [6] are slightly different[4] from that in [4] in Phase 2, we can take similar steps via majorization in Phase 1 so as to improve the system performance with almost no additional cost. Our simulation results in Section V also support this.

B. Is the contention probability vector of bottleneck neighborhood more uneven in the modified algorithms?

As briefly mentioned in the introduction, we define the bottleneck neighborhood $\mathcal{N}_0(n)$ at time slot n in multi-neighborhood networks as

$$\mathcal{N}_0(n) \triangleq \{\mathcal{I}_k : \ k = \arg\max_{k \in \mathcal{E}} \sum_{l \in \mathcal{I}_k} q_l(n)/C_l\}, \tag{9}$$

where $\mathcal{E}$ denotes the set of all links in the network. Similarly, we define by

$$\mathcal{N}_0^z(n) \triangleq \{\mathcal{I}_k : \ k = \arg\max_{k \in \mathcal{E}} \sum_{l \in \mathcal{I}_k} (q_l(n)/C_l)^z\},$$

the z–bottleneck neighborhood when each link weight x is transformed into x^z. Further,

$$z_0(n) \triangleq \max_z \{z \mid \mathcal{N}_0^\theta(n) = \mathcal{N}_0(n), \forall \theta \in [1, z]\}. \tag{10}$$

---

[4]In Phase 2, algorithm in [8] uses different function f_i; algorithm in [6] use different system model with $g_i(r_l(n)) = r_l(n)/M$ for all i ($i = 1, 2, \cdots, M$), as explained in Section II-A.

Note that $\mathcal{N}_0^0(n)$ denotes the neighborhood comprised of the maximum number of links at the nth contention time slot, and $\mathcal{N}_0^\infty(n)$ represents the neighborhood of the link with maximal $q_l(n)/c_l$. For $0 < z < \infty$, larger z gives more weight to links with larger $q_l(n)/c_l$ when deciding $\mathcal{N}_0^z(n)$.

Since $\mathcal{N}_0^1(n) = \mathcal{N}_0(n)$ (i.e., $z = 1$), we trivially have $z_0(n) \geq 1$ from (10). Now, consider a constant $\beta \in [1, z_0(n)]$. Then, for any link $l \in \mathcal{N}_0^\beta(n)$, from (8),

$$r_l^{New}(n) = \alpha \cdot (q_l(n)/C_l)^\beta / \max_{k \in \mathcal{E}} \sum_{l \in \mathcal{I}_k} (q_l(n)/C_l)^\beta.$$

Let $\{r_l(n)\}_{\mathcal{N}_0^\beta}$ and $\{r_l^{New}(n)\}_{\mathcal{N}_0^\beta}$ denote the contention probability vector for z–bottleneck neighborhood ($\mathcal{N}_0^\beta(n)$) given by the original algorithm in (6) and the modified one in (8), respectively. Since $\beta \in [1, z_0(n)]$, we have $\mathcal{N}_0^\beta(n) = \mathcal{N}_0(n)$. This immediately leads to $\{r_l(n)\}_{\mathcal{N}_0^\beta} \prec \{r_l^{New}(n)\}_{\mathcal{N}_0^\beta}$, i.e., the new vector $\{r_l^{New}(n)\}_{\mathcal{N}_0^\beta}$ is *more uneven* than the original one $\{r_l(n)\}_{\mathcal{N}_0^\beta}$. Hence, by Theorem 1, the success probability of channel assignment for the bottleneck neighborhood at the nth contention time slot can be increased by using (8) in stead of (6).

Lastly, as illustrated in Figure 1, the bottleneck neighborhood $\mathcal{N}_0(n)$ defined in (9) is time-varying and so is $z_0(n)$ (see (10). While we only know that $z_0(n) \geq 1$ for any n and is generally unavailable to any given single link, we can bypass this problem in real implementation by suitably choosing small β (e.g., $\beta \leq 10$) to ensure that it is less than $z_0(n)$ for *most* of the time slot n.

V. SIMULATION

In this section, we provide simulation results for both single-neighborhood and multi-neighborhood cases. Although Sections III and IV have already shown that the success probability for channel assignment in an unknown and time-varying bottleneck neighborhood can be improved in a distributed way, our focus here is how this increased success probability can be translated into reduction in queue-length, when compared to the original (unmodified) constant-time distributed scheduling algorithms. For the case of multi-neighborhoods, the performance of bottleneck neighborhood is critical to that of the system. However, the performance improvement of bottleneck neighborhood does not automatically guarantee improved system performance overall, since any improvement of a bottleneck neighborhood can possibly be at the cost of performance degradation of its adjacent neighborhood, which will then become a new bottleneck neighborhood in the next run. Thus, while we can always improve the performance of any arbitrary bottleneck neighborhood, it is still interesting to see how our modified algorithm performs in complex multi-neighborhood networks.

A. Single Neighborhood Case

In our single-neighborhood simulation, 10 links interfere with each other. Each link has the same capacity of 10 pkts per frame time, i.e., once a link successfully gets the channel, it can transmit at most 10 packets. We generate traffic as follows. At the beginning of each frame, an input of 1 pkt arrives to each link randomly with probability λ. We gradually increase the input rate (traffic load) λ by a small step, e.g., 0.01, until the average queue-length goes to infinite at $\lambda = \lambda_{max} + 0.01$, i.e., λ_{max} denotes the system's capacity region. Let M be the number of contention slots. We simulate using $M = 1, 10$.
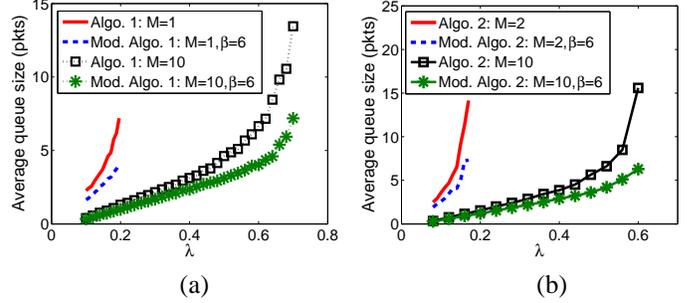

(a)   (b)

Fig. 4. Average queue-length in a single-neighborhood system for algorithm in [4] along with our modified version (a), and for algorithm in [6] with our modified one in (b). We set $\beta = 6$ in (8) for our modified algorithms. For $M = 10$, the capacity region (λ_{max}) for algorithms in [4], [6] as well as our modified versions are 0.73 and 0.61, respectively. While our modified algorithms lead to the same capacity (stability) region, the average queue-lengths for our modified algorithms are always smaller and become half of those in [4], [6] when λ approaches to λ_{max} (heavy-traffic case).

Figure 4 shows simulation results for algorithms in [4], [6], as well as our modified algorithms proposed in Section IV. Note that our modification doesn't change the capacity region λ_{max} (i.e., the average queue-length becomes infinite for $\lambda > \lambda_{max}$ for both cases), but reduces the average queue-length *inside* the capacity region. For heavy-traffic case, i.e., when λ approaches to λ_{max}, the average queue-length of our modified algorithms is *less than half* of those produced by the original algorithms.

B. Multi-Neighborhood Case
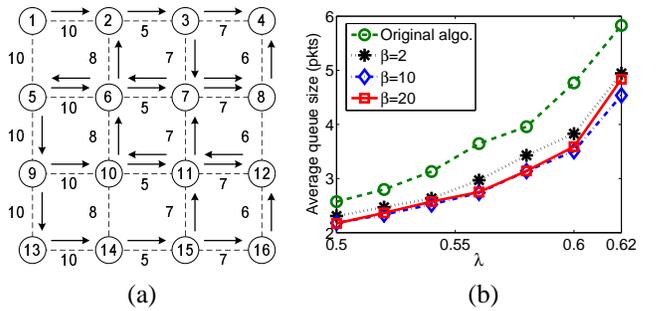

(a)   (b)

Fig. 5. (a) Simulation setting (same as that in [8]): dotted line and solid line with arrowhead represent the link and flow (with direction), respectively. The number on the below/left of dotted line denotes the link capacity. (b) Average queue-length in a multi-neighborhood system: Similar to Fig. 4, the average queue-length decreases, while the capacity region remains the same. In heavy-traffic case, the reduction in the average queue-length is around 25%.

For the multi-neighborhood simulation, we use the algorithm in [4] and our corresponding modified version. For a fair comparison, we use the same simulation setting as that in

[8], which is the preliminary version of [4].[5] Figure 5 shows that for multi-neighborhood networks, we have similar results as in the single-neighborhood case in Figure 4. The average queue-length again becomes smaller (and around 25% or less in heavy-traffic case), while the capacity region remains the same. We have also tested different values of β in (8) to see its effect. In this case, we observe much of the reduction in the average queue-length by choosing $\beta = 2$, while further increasing β to 10 or 20 only gives marginal reduction for any traffic load inside the capacity region.

VI. CONCLUSION

In this paper, we have shown how to tune up the performance – in terms of the success probability of a given neighborhood and the average queue-length – of a class of collision-based, constant-time distributed scheduling algorithms inside their stability regions, with zero additional cost. We explore the benefit of being 'more uneven' for the link contention probability vector and quantify the ordering via the tool of majorization. We expect that our main idea in this paper that leverages the notion of *majorization* of contention probability vectors can also be applied to other distributed collision-based algorithms in wireless ad-hoc networks.

APPENDIX

Proof of Theorem 1: One straightforward way to prove (5) is to show that function $R(\cdot)$ defined in (3) is Schur convex as in Definition 2. By Theorem A.4 in [9], to show that a function $h : \mathcal{R}^N \to \mathcal{R}^N$ is Schur convex, it is necessary and sufficient to show the following:

(a) h is symmetric on $\mathcal{R}^N$.

(b) For all $k \neq j$,

$$(r_k - r_j)[h_{(k)}(\vec{r}) - h_{(j)}(\vec{r})] \geq 0, \quad \forall \vec{r} \in \mathcal{R}^N,$$

where $\vec{r} = \{r_1, \ldots, r_N\}$ and $h_{(k)}(\vec{r}) = \partial h(\vec{r})/\partial r_k$, i.e., $h_{(k)}(\vec{r})$ is the partial derivative of function h w.r.t. the kth element of $\vec{r}$. By the symmetric condition in (a), the above condition is equivalent to

$$(r_1 - r_2)[h_{(1)}(\vec{r}) - h_{(2)}(\vec{r})] \geq 0, \quad \forall \vec{r} \in \mathcal{R}^N. \quad (11)$$

First, note that R_i defined in (2) is symmetric on $\mathcal{R}^N$. Hence, $R(\vec{r}) = \sum_{i=1}^{M} R_i(\vec{r})$ is also symmetric on $\mathcal{R}^N$.

Second, by calculating $R_{i(j)} \triangleq \partial R_i(\vec{r})/\partial r_j$, we have

$$R_{i(1)} = \left[f_i'(r_1)(1 - \sum_{n=1}^{i} f_n(r_2)) - \left(\sum_{n=1}^{i} f_n'(r_1) \right) f_i(r_2) \right] N_{i1}$$

$$- \left(\sum_{n=1}^{i} f_n'(r_1) \right) \left(1 - \sum_{n=1}^{i} f_n(r_2) \right) N_{i2}, \quad (12)$$

where

$$N_{i1} = \prod_{k \neq 1,2} \left(1 - \sum_{n=1}^{i} f_n(r_k) \right) \geq 0,$$

$$N_{i2} = \sum_{l \neq 1,2} \left[f_i(r_l) \prod_{k \neq j,l,2} \left(1 - \sum_{n=1}^{i} f_n(r_k) \right) \right] \geq 0.$$

---

[5][4] did not provide simulation results.

We can obtain $R_{i(2)}$ by exchanging indexes 1 and 2 in (12). From (4),

$$f_i'(r_1)(1 - \sum_{n=1}^{i} f_n(r_2)) - \left(\sum_{n=1}^{i} f_n'(r_1) \right) f_i(r_2)$$

$$= e^{-\alpha_i(r_1 + r_2)} \left[2\alpha_i - \alpha_{i-1} e^{\triangle_{i-1} r_1} - \alpha_i e^{\triangle_{i-1} r_2} \right], \quad (13)$$

where $\triangle_{i-1} = \alpha_i - \alpha_{i-1}$. Also,

$$\left(\sum_{n=1}^{i} f_n'(r_1) \right) \left(1 - \sum_{n=1}^{i} f_n(r_2) \right) = \alpha_i e^{-\alpha_i r_1} e^{-\alpha_i r_2}. \quad (14)$$

From (13) and (14) and by interchanging the indexes 1 and 2 in these two equations to obtain $R_{i(2)}$, we have from (12),

$$R_{i(1)} - R_{i(2)} = e^{-\alpha_i(r_1 + r_2)} \triangle_{i-1} (e^{\triangle_{i-1} r_1} - e^{\triangle_{i-1} r_2}) N_{i1}.$$

Since α_i is non-decreasing, $\triangle_{i-1} \geq 0$. Thus, if follows that

$$(r_1 - r_2)(R_{i(1)} - R_{i(2)})$$
$$= e^{-\alpha_i(r_1 + r_2)} \triangle_{i-1}(r_1 - r_2)(e^{\triangle_{i-1} r_1} - e^{\triangle_{i-1} r_2}) N_{i1} \geq 0.$$

Hence, for any $i \in \{1, \ldots, M\}$, $R_i(\cdot)$ is Schur convex, and so is their sum $R(\cdot)$. This completes the proof.

REFERENCES

[1] P. Chaporkar, K. Kar, and S. Sarkar. Throughput Guarantees in Maximal Scheduling in Wireless Networks. In *Proceedings of 43rd Annual Allerton Conference on Communication, Control and Computing*, Monticello, IL, September 2005.

[2] R. Cruz and A. Santhanam. Optimal routing, link scheduling and power control in multi-hop wireless networks. In *Proceedings of IEEE INFOCOM*, San Francisco, CA, April 2003.

[3] A. Eryilmaz, R. Srikant, and J. R. Perkins. Stable scheduling policies for fading wireless channels. *IEEE/ACM Transactions on Networking*, 13(2):411–424, 2005.

[4] A. Gupta, X. Lin, and R. Srikant. Low-Complexity Distributed Scheduling Algorithms for Wireless Networks. In *Proceedings of IEEE INFOCOM*, Anchorage, Alaska, May 2007.

[5] M. Hanckowiak, M. Karonski, and A. Panconesi. On the distributed complexity of computing maximal matchings. *SIAM Journal of Discrete Mathematics*, 15(1):41–57, 2001.

[6] C. Joo and N. B. Shroff. Performance of random access scheduling schemes in multi-hop wireless networks. In *Proceedings of IEEE INFOCOM*, Anchorage, Alaska, May 2007.

[7] Emilio Leonardi, Marco Mellia, Fabio Neri, and Marco Ajmone Marsan. On the stability of input-queued switches with speed-up. *IEEE/ACM Transactions on Networking*, 9(1):104–118, 2001.

[8] X. Lin and S. Rasool. Constant-time distributed scheduling policies for ad hoc wireless networks. In *Proceedings of IEEE Conference on Decision and Control*, San Diego, CA, December 2006.

[9] A. W. Marshall and I. Olkin. *Inequalities: Theory of Majorization and Its Applications*. Academic Press, 1979.

[10] M. J. Neely, E. Modiano, and C. E. Rohrs. Dynamic power allocation and routing for time varying wireless networks. In *Proceedings of IEEE INFOCOM*, San Francisco, CA, April 2003.

[11] S. Sanghavi, L. Bui, and R. Srikant. Distributed link scheduling with constant overhead. In *Proceedings of ACM SIGMETRICS*, San Diego, CA, June 2007.

[12] L. Tassiulas. Scheduling and performance limits of networks with constantly changing topology. *IEEE Transactions on Information Theory*, 43(3):1067–1073, May 1997.

[13] X. Wu and R. Srikant. Bounds on the capacity region of multihop wireless networks under distributed greedy scheduling. In *Proceedings of IEEE INFOCOM*, Barcelona, Spain, April 2006.