Wireless Scheduling Algorithms with $O(1)$ Overhead for M-hop Interference Model

Yung Yi and Mung Chiang

***Abstract*—We develop a family of distributed wireless scheduling algorithms that requires only $O(1)$ complexity for M-hop interference model, for any finite M. The recent technology advances and heterogeneity in wireless networks lead to various interference patterns. Thus, a scheduling algorithm geared into a specific interference model (typically one-hop or two-hop in literature) may be limited in its applicability. In this paper, we tackle this problem, and develop a family of scheduling algorithms (which guarantees throughput and delay performance) for M-hop interference models. To achieve such a goal, we use the concept of *vertex augmentation*, and for a given M, the family of parameterized algorithms are proposed and the tradeoffs among throughput, complexity, and delay are studied.**

I. Introduction

A. Motivation and Related Work

There have been extensively growing interests in distributed MAC scheduling algorithm with provable throughput-guarantees since the seminal work by Tassiulas and Ephremides [1]. Since then, various approaches to distributed algorithms, possibly with low complexity, have been taken based on the following ideas: *(i)* maximal/greedy algorithms (e.g., [2], [3]), *(ii)* pick-and-compare algorithms (e.g., [4], [5]) inspired by [6], and *(iii)* random-access approach (e.g., [7]). In recent studies in [8]–[10], the authors have developed the parameterized scheduling algorithms, which tradeoffs between throughput and complexity. The key idea is to parameterize the algorithms with some integer k, where the achieved throughput and the complexity become a function of k. In [8], the authors used the concept of link-augmentation under one-hop interference model. The authors in [9], [10] used the notion of graph-partitioning.

Most of afore-mentioned research focuses only on one-hop or two-hop interference model. The research under more general interference models has been made in [11], where the authors considered M-hop interference models and have experimentally shown that the optimal value of M can be affected by the physical layer characteristics, allowing larger than 2. Finding *a distributed algorithms with constant overhead that works for M-hop interference model* is an under-explored area, which we study in this paper.

B. Model

We assume that time is slotted, denoted by t. A time-slot duration is suitably chosen to accommodate the transmission of one fixed-size packet. We model the wireless multi-hop network by a graph $G(L,N)$, where L and N denote a set of (bi-directional) links, and a set of nodes, respectively. We abuse the notations L and N to refer to the number of links and nodes, respectively. We denote a bi-directional link from node i to node j by $\overline{ij}$. We let $d(l,l')$ be the hop distance between two links $l,l' \in L$. The wireless system has a *single* channel (e.g., frequency or code) and each node is time-synchronized and has a half-duplex radio.

We consider M-hop interference models: for some positive integer M, if $d(l,l') \leq M$ then the transmissions over both links l,l' are interfering with each other. The increasing values of M represent more stringent interference constraints in a wireless system. This interference model generalizes popular one-hop (appropriate for Bluetooth and FH-CDMA network) or two-hop (appropriate for IEEE 802.11) interference models in literature.

A *link schedule*, or simply a schedule, $\boldsymbol{S} = (S_l \in \{0,1\} : l = 1,\ldots,L)$, is a vector representing the set of scheduled links, where $S_l = 1$ if the link l is scheduled, and 0 otherwise. A link schedule $\boldsymbol{S}$ is said to be *valid* under M-hop interference model, if all transmissions scheduled by S are successful without any collision. We denote by $\mathcal{S}$ the set of the entire valid schedules.

We denote by $D_l[t]$ the number of arrivals over the link l. The arrival process is assumed to be i.i.d. Bernoulli process, and also independent across links, with mean $\mathrm{E}[D_l[t]] = \lambda_l$. Denote by $\boldsymbol{\lambda} = (\lambda_l : l \in L)$ the mean arrival vector. Let $Q_l[t]$ be the queue length of link l at time t. Then, the queue length dynamics of link l is represented by:

$$Q_l[t+1] \quad = \quad Q_l[t] + D_l[t+1] - \mathbf{1}_{\{Q_l[t]>0\}} S_l[t], \qquad (1)$$

where $\mathbf{1}_E$ is the indicator function for the event E.

Definition 1.1 (Stability):

$$\limsup_{t\to\infty} \frac{1}{t} \sum_{s=0}^{t} \mathrm{E}\Big[\sum_{l\in L} Q_l[s]\Big] < \infty.$$

Definition 1.2 (Throughput-region): The throughput-region $\Lambda \in \mathcal{R}_+^L$ of the system is defined as the set of $\boldsymbol{\lambda}$ for which there exists a scheduling stabilizing the system.

Definition 1.3 (α-Throughput-optimality): A scheduling is said to be α-throughput-optimal[1], $0 < \alpha \leq 1$, if it can stabilize the system for any $\boldsymbol{\lambda} \in \alpha\Lambda$.

Finally, we introduce the famous *conflict graph* to understand interference relations among links: For a given graph $G(L,N)$ under M-hop interference model, its conflict graph $G^c(E,V)$ is such that $L = V$ and two vertices $i,j \in V$ are

The authors are with the Department of Electrical Engineering, Princeton University. E-mails: {yyi,chiangm}@princeton.edu.

[1]For simplicity, we use just 'throughput-optimal' when $\alpha = 1$.

connected if $d(i,j) \leq M$ in G. To avoid confusion, we call the elements of L and N of the original graph G *links* and *nodes*, and those of E and V of its conflict graph G^c *edges* and *vertices*.

C. Contributions

The main contributions of this paper are the followings:

1) We develop a family of constant-overhead scheduling algorithms that are parameterized by the given interference number M and other performance parameter k. The parameters k as well as M characterize the tradeoffs among throughput, delay, and complexity, e.g., as k increases, the achieved throughput also increases at the cost of larger complexity and delay.
2) Our work is an extension of [8] in that we also use the idea of augmentation in graph theory. However, we consider augmentation in the conflict graph, thus called *vertex augmentation*, instead of link augmentation [8] in the original graph. Applying the idea of augmentation to the conflict graph (in conjunction with non-trivial extension of the work in [8]) enables us to study scheduling algorithms irrespective of M. Furthermore, our work differs from [8] in terms of the proof technique, in that we prove the throughput and delay performance using the framework of "approximate" algorithms.

Due to space limitation, we present the proofs of all lemmas and theorems in our technical report [12].

II. Preliminaries and Vertex Augmentation

A. Approximate Algorithms

In this subsection, we describe general criteria which, if satisfied, render a scheduling algorithm α-throughput-optimal. We first define the *weight* of a schedule that intuitively represents a measure of quality of a chosen schedule $\boldsymbol{S}[t]$ (at slot t):

$$W[t] \triangleq W(\boldsymbol{S}[t]) \triangleq \sum_{l \in L} (Q_l[t] \times S_l[t]).$$

We denote by $S^\star[t]$ the schedule that maximizes the weight, i.e.,

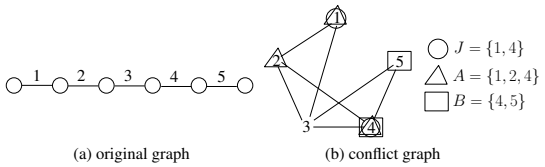
$$\boldsymbol{S}^\star[t] = \arg\max_{S \in \mathcal{S}} \Big(\sum_{l \in L} Q_l[t] \times S_l\Big), \quad W^\star[t] \triangleq W(\boldsymbol{S}^\star[t]).$$

Definition 2.1 ((α,ξ)-approximate algorithms): A scheduling algorithm is said to be (α,ξ)-approximate, $0 < \alpha, \xi < \infty$, if, at any slot t, for two random variables $G[t]$ and $C[t]$,

$$W[t] \geq G[t] W^\star[t] - C[t], \quad G[t] \geq \alpha, \quad \mathrm{E}\Big[C[t] | Q[t]\Big] \leq \xi.$$

Our notion of approximate algorithm generalizes the famous "Max-Weight" scheduling [1], which is throughput-optimal, where $\alpha = 1$ and $C[t] = 0$, a.s. The following algorithm also belongs to (α,ξ)-approximate algorithms, which we use extensively throughout this paper.

Definition 2.2 (Pick-and-Compare [6], [8]): The algorithm first generates a *random* schedule $\boldsymbol{S}'[t]$ satisfying **C1**, and then schedule $\boldsymbol{S}[t]$ defined in **C2**.



(a) original graph (b) conflict graph

Fig. 1. An example of two disjoint augmentations A and B of J under 2-hop interference.

C1. $\exists 0 < \delta \leq 1$, s.t. $\mathrm{P}\Big[\boldsymbol{S}'[t] = \boldsymbol{S} \mid \boldsymbol{Q}[t]\Big] \geq \delta$, for some schedule $\boldsymbol{S}$, where $W(\boldsymbol{S}) \geq \alpha W^\star[t]$.

C2. $\boldsymbol{S}[t] = \arg\max_{S = \{\boldsymbol{S}[t-1], \boldsymbol{S}'[t]\}} W(\boldsymbol{S})$.

The Pick-and-Compare algorithm provides only a *probabilistic* guarantee of finding a "α-optimal" schedule with a suitable comparison operation defined in **C2**. As shown in the next proposition, this is enough to achieve α-throughput-optimality.

Proposition 2.1 ([13]): The (α,ξ)-approximate algorithms are α-throughput-optimal. Further, a Pick-and-Compare algorithm is $(\alpha, L(1+\alpha)/\delta)$-approximate, thus α-throughput-optimal.

B. Vertex Augmentation

In this subsection, we introduce the notion of "augmentation," which is a key concept in the distributed algorithms we develop. Consider an independent set J in the conflict graph $G^c(E,V)$, where recall that J is a valid schedule in G. A path or cycle, which is a sequence of *vertices,* with respect to an independent set J is said to be *alternating,* if the vertices are chosen alternatingly from J and $V \setminus J$. An alternating path or cycle A (of J) is said to be an *augmentation*, $A \oplus J$ is also an independent set, where $A \oplus J \triangleq (A \cup J) \setminus (A \cap J)$. This means that a vertex-set generated by adding $A \setminus J$ to J and removing $A \cap J$ from J is still an independent set. In such a case, J is said to be *augmented* by A.

We define the *size* of an augmentation A, $\theta(A)$, to be $|A \setminus J|$, which is the number of vertices that are *newly added* to J. Two augmentations A and B are said to be *disjoint*, if any two nodes $v \in A \setminus J$ and $v' \in B \setminus J$ are not *connected* in G^c. This implies that 1) $(A \cup B) \oplus J$ is also an independent set, and 2) $(A \cup B) \oplus J$ can be obtained by sequentially augmenting J with A and B, irrespective of their orders. We define a *sub-augmentation* of an augmentation A to be a subset of A. Note that disjointness and augmenting operation are well-defined for sub-augmentations.

Suppose that a vertex $v \in G^c$ is assigned some weight Q_v, which is the queue length of link v in G. Then, efficiency of an (sub)augmentation A of J is defined as the *added* weight by augmenting, i.e.,

$$G(A) \quad \triangleq \quad \sum_{v \in A \setminus J} Q_v - \sum_{v \in A \cap J} Q_v.$$

Example 2.1: Figure 1 illustrates a set of two disjoint augmentations $\mathcal{A} = \{A, B\}$ for G and G^c. Note that the set of augmentations $\mathcal{A}$ is *ambiguous* in the sense that it is derived from the conflict graph. In Figure 1, the hop distances over links '1' and '2' and over links '1' and '3' are different in the original graph, but both are connected by one-hop in the

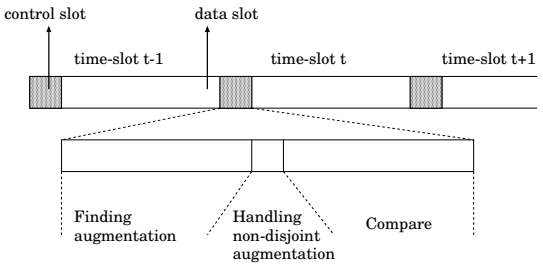


Fig. 2. Slot-structure and algorithm overview

conflict graph. Under M-hop interference model, the real hop distance of connected vertices in the conflict graph can be a value from 1 to M.

We conclude this section by presenting a useful lemma to be used in the algorithm analysis (see Section IV). This lemma intuitively states that for a given augmentation size k, we can find a reasonably "good" schedule, by first finding an appropriate set of disjoint augmentations $\mathcal{A}$ and then by augmenting $\boldsymbol{S}[t-1]$ with $\mathcal{A}$.

Lemma 2.1: For a given parameter k, consider a schedule $\boldsymbol{S}[t-1]$ at time $t-1$, which is an independent set in G^c. Then, there exists a set of disjoint augmentations, $\mathcal{A}$, of $\boldsymbol{S}[t-1]$, such that

$$W(\boldsymbol{S}[t-1] \oplus \mathcal{A}) \geq \frac{k}{k+2} W^\star[t], \quad \forall A \in \mathcal{A}, \theta(A) \leq k \qquad (2)$$

The similar result was shown in [8] for *link augmentation in G under one-hop interference model.* Lemma 2.1 says that a similar property holds for *vertex augmentation in* G^c (implying that it works for any M-hop interference model).

III. Algorithm Description

The slot structure and the corresponding control operations are depicted in Figure 2. One time-slot is composed of control and data slot, where there are multiple mini-slots in a control slot. The number of mini-slots are parameterized by both k and M. The operations in a control slot are divided into the following three phases of sub-operations:

O1. *Finding random augmentations.* We randomly find a set of augmentations $\mathcal{A}$, which satisfies (2) in Lemma 2.1 with some positive probability. This will ensure **C1** in the Pick-and-Compare algorithm, where $\alpha = k/(k+2)$.

O2. *Handling non-disjointness.* The augmentations found in **O1** does not guarantee their disjointness, in which case $\boldsymbol{S}[t-1] \oplus \mathcal{A}$ may not be an independent set in G^c. We slightly modify $\mathcal{A}$ to one of its sub-augmentation $\tilde{\mathcal{A}}$, such that $\tilde{A} \oplus \boldsymbol{S}[t-1]$ is an independent set.

O3. *Comparison.* We determine the final schedule $\boldsymbol{S}[t]$ in a distributed manner, such that $W(\boldsymbol{S}[t])$ is the maximum of $W(\boldsymbol{S}[t-1])$ and $W(\boldsymbol{S}[t-1] \oplus \mathcal{A})$, such that **C2** in the Pick-and-Compare algorithm is satisfied.

A. Finding random augmentations

The procedure to find random augmentations is summarized as the following: *(i)* Each node decides to be a *seed* with some given probability p, where a seed corresponds to a node of the starting vertex (i.e., link) of an augmentation, and *(ii)* the seed randomly chooses its augmentation size out of $\{1, \ldots, k\}$, and adds the links one by one to constitute its augmentation until the intended augmentation size is reached or some collision occurs. This happens for multiple seeds in a parallelized way.

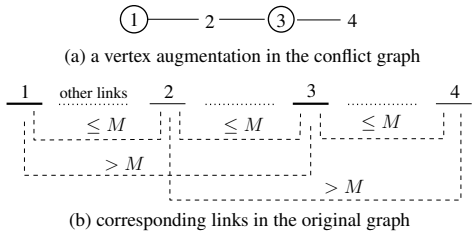


Fig. 3. Hop-requirement for a vertex augmentation under M-hop interference model.

1) Hop-requirement: Consider an vertex augmentation $A = (l_1, \ldots, l_n)$, which is either a path or a cycle. By definition of augmentation on alternation, the links in A should be chosen from $\boldsymbol{S}[t-1]$ and $L \setminus \boldsymbol{S}[t-1]$ alternatingly. Then, it is not hard to see under M-hop interference model, the following *hop-requirement* should be satisfied (see Figure 3 for an illustration):

Requirement 3.1 (Hop requirement):

(i) $d(l_i, l_{i+1}) \leq M$, $i = 1, \ldots, n-1$,
(ii) $d(l_i, l_{i+2}) > M$, $i = 1, \ldots, n-2$.

The (i) comes from the connectivity of two adjacent nodes in G^c, and the (ii) is due to the fact that after augmenting, the resulting set of vertex should be an independent set in G^c (i.e., valid schedule in G).

2) Algorithm for finding augmentation: We now describe the algorithm of finding random augmentations, mainly assuming a single augmentation for simplicity. This may be not complete to explain the full algorithm, but sufficient to include the key features of the algorithm. In Figure 4, let $\boldsymbol{S}[t-1] = \{1, 4, 8\}$ (thick solid lines). The augmentation A is initialized to $A = \emptyset$ to which links will be added. By definition of augmentation, the links out of $\boldsymbol{S}[t-1]$ and $L \setminus \boldsymbol{S}[t-1]$ should be inserted to A alternatingly, which we call <u>old</u> and <u>new</u>, respectively.

Suppose only a decides to be a seed, and a randomly chooses its augmentation size out of $\{1, \ldots, k\}$. Recall that the augmentation size is defined as $|A \setminus J|$. The procedure to add alternate links are performed by exchanging REQ and ACK, where we call the REQ-sender *active*. The active nodes keep changing to a randomly chosen path as time advances.

The REQ message has three attributes, denoted by REQ($h_p, h,$ `what`). `what = 0` means that an old link is requested to be added, and `what = N`, otherwise. We introduce the notation 'toggle' with toggle(`0`)=`N` and toggle(`N`)=`0`. When a node, say v, receives REQ($h_p, h,$ `what`) message, the following operations are performed:

a) If $h \neq 0$, then v just decrements h by one, and relay the received REQ to a random outgoing link.
b) If $h = 0$,

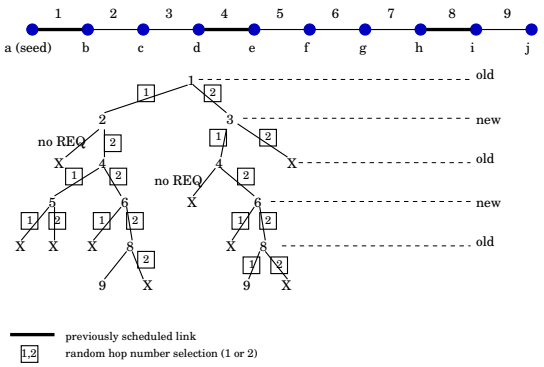


Fig. 4. Example for a single vertex augmentation: two-hop interference model.

b-1) Find an old (resp. new) outgoing link if what = 0 (resp. what = N).
b-2) If found, then send ACK to the REQ-sender.
b-2) Let $h' = \text{RAND}(M-1-h_p, M-1)$ [2]. Then, finally send $\text{REQ}(h', h', \text{toggle}(\texttt{what}))$ across the chosen outgoing link.

The $\text{REQ}(h_p, h, \texttt{O/N})$ from nodes v to v' means that node v tries to add an old/new link which is h hop away from v, where the maximum value of h can be $M-1$ (see (i) in the hop-requirement) Also, we observe that the h_p value in the new REQ is set to the new h (see (ii) in the hop-requirement).

To illustrate, consider an example in Figure 4. The seed node a adds the old link 1 to its augmentation by sending $\text{REQ}(M-1, 0, \texttt{O})$ (the seed node just uses $h = 0$ for immediate addition of the outgoing link and $h_p = M-1$) and receiving ACK from the node b. Node b newly generates and sends $\text{REQ}(h, h, \texttt{N})$ to node c, where $h = \text{RAND}(M-1-(M-1), 1) = \text{RAND}(0, 1)$, where we can consider two cases.

(i) $h = 0$: In this case, link 2 is requested as a new link and added to A. The node c receives REQ/ACK from/to node b, and will generate a new REQ message for the next old link. Since link 1 is just one-hop away from link 2, $h = 0$ should be excluded in the REQ message from node c due to the hop requirement (ii). In order to avoid such an unnecessary situation, node c needs to know the value of h that b originally chooses at the time of REQ generation. To that end, when a node generate REQ with random value of h, it copies $h_p = h$. Note that h is decremented by 1 at each hop, but h_p is invariant until REQ message reaches its final intended destination node. Then, a new node that needs to generate REQ message just generates the value of $h = \text{RAND}(M-1-h_p, M-1)$. In our example, since node b selected $h = 0$, node c, in its new REQ generation, selects $h = \text{RAND}(2-1-0, 1) = 1$.
(ii) $h = 1$: In this case, node b just decrements h and relays REQ to the node c, which will add link 3, and then node d generate a new $\text{REQ}(h, h, \texttt{N})$, where $h = \text{RAND}(2-1-1, 2-1)$ [3]

3) Stopping rule: The procedure for finding augmentation ends until the following events occurs:

- Augmentation size: The intended augmentation size (randomly chosen by the seeds initially out of $\{1, \ldots, k\}$) is satisfied.
- No desired link: A node receives $\text{REQ}(\cdot, h, \texttt{what})$ with $h = 0$, but the outgoing link corresponds to what (i.e., old or new) is not found.
- REQ/ACK collision: The REQ sender cannot receive ACK possibly due to REQ/ACK collision.

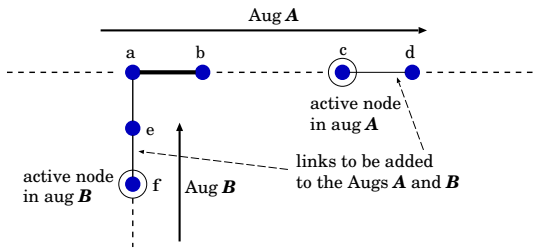


Fig. 5. Example of non-disjoint augmentations under 3-hop interference model.

B. Handling non-disjointness

We note that the sets of augmentations found according to the procedure in the previous sub-section are not necessarily disjoint. Guarantee of disjointness is crucial since it enables the algorithm (i) to run in a parallelized manner (recall that a set of disjoint augmentations allows us to augment $\boldsymbol{S}[t-1]$ in any order), and (ii) to satisfy **C2** in the Pick-and-Compare algorithm (this will be elaborated after Comparison operation in Subsection III-C).

To prevent this non-disjointness, the following operations are performed:

Handling non-disjointness

1. A source node of a link that has been added to an augmentation, but *was not in* $\boldsymbol{S}[t-1]$, sends REQ and waits for ACK.
2. If ACK not received, then the source node deletes the corresponding links from its augmentation.

To illustrate, consider Figure 5, where two augmentations A and B are currently being made. Nodes c and e are the active nodes of the augmentations A and B, respectively. Suppose that both $\overline{ab}$ and $\overline{fe}$ are not in $\boldsymbol{S}[t-1]$, but $\overline{ab}$ are already in the A. Assume that c is sufficiently far from e, such that $\overline{cd}$ and $\overline{fe}$ are not interfering with each other ($d(\overline{cd}, \overline{fe}) > 3$). In this case, $\overline{fe}$ (resp. $\overline{cd}$) will be added to B (resp. A), since their REQs will not collide. Note, however, $d(\overline{ab}, \overline{fe}) \leq 3$, thus, A and B are not disjoint. By handling non-disjointness described in the above, over the links in A and B, but not

[2]The function $\text{RAND}(n, m)$ returns a random number out of $\{n, n+1, \ldots, m\}$.

[3]As a special case, if there is a direct outgoing link which was in $\boldsymbol{S}[t-1]$ to the node d, then the node d can directly add it in its augmentation as an old link (i.e., just letting h=0), since otherwise we cannot find another old link within M-hop of d. However, it does not hurt the proof of throughput property of the algorithm.

in $\boldsymbol{S}[t-1]$, such as $\overline{ab}$ and $\overline{fe}$, REQs/ACKs are exchanged, leading to collision. Thus, $\overline{ab}$ and $\overline{fe}$ are deleted from A and B. By this operation, the final augmentations may become sub-augmentation. We also note that this procedure does not occur for the links in the both augmentation and $\boldsymbol{S}[t-1]$, since they are interference-free by our schedule construction.

C. Comparison

We initially let $\boldsymbol{S}[t] = \boldsymbol{S}[t-1]$. Using the set of disjoint sub-augmentations $\tilde{A}$, for each $A \in \tilde{\mathcal{A}}$, we compute its gain $G(\tilde{A})$ by backtracking from the terminus in A. If $G(\tilde{A}) > 0$, we decide to augment, i.e., subtract $\tilde{A} \cap \boldsymbol{S}[t-1]$ from $\boldsymbol{S}[t]$ and add $\tilde{A} \setminus \boldsymbol{S}[t]$. If $G(\tilde{A}) \leq 0$, we take no action, i.e., we decide to schedule the links in $\boldsymbol{S}[t-1] \cap A$ from the initialization of $\boldsymbol{S}[t] = \boldsymbol{S}[t-1]$. Note that from disjointness of sub-augmentations in $\tilde{\mathcal{A}}$, we are safe to perform the above operations simultaneously in $\forall A \in \tilde{\mathcal{A}}$. Then, for the resulting schedule $\boldsymbol{S}[t]$, it is easy to prove that our algorithm satisfies **C2** in the Pick-and-Compare algorithm.

IV. Analysis: Throughput, Complexity, and Delay

First, we describe Lemma 4.1, which is the key to the proof of throughput-guarantee.

Lemma 4.1: For a given M and k, there is a positive probability lower-bounded by $\delta(k, M)$ that our algorithm finds a set of disjoint augmentations satisfying (2) in Lemma 2.1, where

$$\delta(k, M) = \min\left[1, \left(\frac{p}{1-p}\right)^N\right]\left(\frac{1-p}{k\Delta M}\right)^N, \tag{3}$$

where Δ is the maximum degree of the graph G.

Theorem 4.1: Our algorithm is $k/(k+2)$-throughput-optimal for any finite $M > 0$.

The proof is as follows: Lemma 4.1 guarantees **C1** in Definition 2.2, and it is easy to prove that the operations of handling non-disjointness and comparison guarantees **C2**. Thus, the result follows. In terms of the proof technique, different from those in [6], [8], which use a combination of two Lyapunov functions, we use the framework of (α, ξ)-approximate algorithms (see Proposition 2.1), which we believe is separately interesting.

We analyze the algorithm complexity, measured by the number of control mini-slots. First, for the operation of finding augmentations, at maximum, for an augmentation of $\theta(k)$, there are $(k+1)$ links in $\boldsymbol{S}[t-1]$. Then, there are $2k$ "intervals" alternating between any old and new links. At each interval, there are $(M-1)$ links at maximum. Thus, the number of mini-time-slots are $2k+1+2k(M-1) = 2kM+1$. Second, we need 2 slots for handling non-disjoint augmentations. Finally, we need $2kM+1$ complexity to traverse from the seed to the terminus in order to recompute the gain. Then, we need another $2kM+1$ complexity to notify the links about their schedulability by backtracking from the terminus to the seed. Thus, the total complexity is $2kM + 1 + 2 + 2(2kM + 1) = 6kM + 5$.

Remark 4.1: For one-hop interference model (i.e., $M = 1$), the algorithm does not require the operation of handling non-disjoint augmentations, as well as the operation of recomputing the gain. Thus, the total complexity becomes $4k + 2$, which is same as that in [8].

It is well-known that the exact delay analysis of this constrained queueing system is still open problem. However, it is possible to compute the following bound on the total average delay:

Theorem 4.2:

$$\lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T}\sum_{l\in L} \mathrm{E}\big[Q_l[t]\big] \leq \frac{L^2\left(\frac{(1+\alpha)}{\delta(k,M)}+1\right)}{\alpha d_k(\boldsymbol{\lambda})},\ \alpha = \frac{k}{k+2}, \tag{4}$$

where

$$d_k(\boldsymbol{\lambda}) \triangleq \sup\left\{\epsilon | \boldsymbol{\lambda} \in (1-\epsilon)\left(\frac{k}{k+2}\right)\Lambda\right\},$$

and $\delta(k, M)$ is defined in (3).

We conclude this section by providing interpretations on the analytic results described above. For a fixed interference model (i.e., a fixed M), as k increases, the achieved throughput region also increases, but at the cost of increasing complexity (which is still $O(1)$). The major cause of sustaining $O(1)$ complexity is due to the sacrifice of exponentially increasing delay (see (3) and (4)), with respect to the network size. Similarly, for a given k, increasing values of M do not affect throughput property (i.e., determined only by k) with increase of $O(1)$ complexity. However, again we should pay the cost of exponentially increasing delay.

V. Concluding Remarks

We have proposed a family of distributed MAC scheduling algorithms, parameterized by the performance parameter k under a general M-hop interference model. Our analysis on throughput, delay, complexity provides inherent achievable tradeoffs.

References

[1] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling for maximum throughput in multihop radio networks," *IEEE Transactions on Automatic Control*, vol. 37, no. 12, pp. 1936–1949, December 1992.

[2] P. Chaporkar, K. Kar, and S. Sarkar, "Throughput guarantees through maximal scheduling in wireless networks," in *Proceedings of the Allerton Conference*, 2005.

[3] X. Wu and R. Srikant, "Bounds on the capacity region of multi-hop wireless networks under distributed greedy scheduling," in *Proceedings of INFOCOM*, 2006.

[4] E. Modiano, D. Shah, and G. Zussman, "Maximizing throughput in wireless networks via gossiping," in *Proceedings of ACM SIGMETRICS*, New York, NY, USA, 2006.

[5] A. Eryilmaz, A. Ozdaglar, and E. Modiano, "Polynomial-time complexity algorithms for full utilization of multi-hop wireless networks," in *Proceedings of IEEE INFOCOM*, 2007.

[6] L. Tassiulas, "Linear complexity algorithms for maximum throughput in radionetworks and input queued switches," in *Proceedings of IEEE INFOCOM*, 1998.

[7] C. Joo and N. B. Shroff, "Performance of random access scheduling schemes in multi-hop wireless networks," in *Proceedings of IEEE INFOCOM*, 2007.

[8] S. Sanghavi, L. Bui, and R. Srikant, "Distributed link scheduling with constant overhead," in *Proceedings of ACM SIGMETRICS*, 2007.

[9] S. Ray and S. Sarkar, "Arbitrary throughput versus complexity tradeoffs in wireless networks using graph partitioning," in *Proceedings of Information Theory and Applications Second Workshop*, 2007.

[10] K. Jung and D. Shah, "Low delay scheduling in wireless network," in *Proceedings of ISIT*, 2007.

[11] G. Sharma, R. R. Mazumdar, and N. B. Shroff, "On the complexity of scheduling in wireless networks," in *Proceedings of MOBICOM*, 2006.

[12] Y. Yi and M. Chiang, "Wireless scheduling algorithms with O(1) complexity for M-hop interference model," Princeton University, Tech. Rep., 2007.

[13] Y. Yi, A. Proutiere, and M. Chiang, "Complexity in wireless scheduling: Impact and tradeoffs," in *Proceedings of ACM MOBIHOC*, 2008, to appear.