

Fast and Distributed Computation of Schedules in Wireless Networks

Supratim Deb[†], Karan Mangla[‡], K. V. M. Naidu[†]

[†]Bell Labs Research India, E-mail: {supratim, naidukvm}@alcatel-lucent.com

[‡]Stanford University, CA, USA, E-mail: kmangla@cs.stanford.edu

Abstract—In a wireless network with *node exclusive spectrum sharing*, two popular schedules are maximum weight matching (MWM) schedule and maximum size matching (MSM) schedule. The former has been proved to be throughput optimal and has superior delay properties, and the latter schedules as many links, with packets to transmit, as possible. However, it is challenging to design algorithms for computing these schedules that (i) are distributed, (*i.e.*, only local message exchanges between neighboring nodes are permitted) (ii) have low running times (iii) exchanges a small number of messages. In this paper, we develop algorithms that satisfy these properties and also provide good approximations to MWM and MSM schedules. We also note that constant approximation to MWM leads to improved delay properties.

We refer to a *round* as a length of time over which every node in the network can make at most one message-transmission attempt. We propose distributed algorithms for computing (i) $1/2 - \epsilon$ approximation to MWM schedule in $O(\log(1/\epsilon) \log^2 n)$ rounds, and (ii) $2/3 - \epsilon$ approximation to MSM schedule in $O((1/\epsilon) \log^2 n)$ rounds, where n is the network size. Simulation results with a popular model for wireless ad-hoc networks demonstrate that (i) our algorithms perform within 85% – 95% of the optimal in many scenarios, and (ii) the time-complexity of the algorithms can be reduced considerably in practice. The number of message transmissions for both our algorithms scale as $O(n \log^2 n)$. In summary, ours is the first work to (i) provide half (two-third) approximate distributed algorithms for computing MWM (MSM) schedule with logarithmic time-complexity and quasi-linear message exchanges (ii) demonstrate that the algorithms are close to optimal for realistic topologies.

I. INTRODUCTION

The problem of scheduling in wireless networks refers to selecting transmitter-receiver node pairs so as to utilize the radio resources in an efficient manner. This paper deals with distributed computation of schedules in wireless networks that can be modeled by *node exclusive spectrum sharing*. Under such a spectrum sharing, schedules should satisfy the following: (i) a node can either be a transmitter or a receiver of data in a schedule, and (ii) a node can receive data from, or transmit data to one of its neighbors. Such a spectrum sharing model is motivated by *Frequency Hopping CDMA* [8] and *Bluetooth* [16] networks, and has also been considered in [8], [14], [15], [17], [19]. Another recent area where such a model could be applicable is OFDMA based WiMAX mesh [1]. In OFDMA based WiMAX mesh networks, the frequency band is split into a number of sub-channels. Thus, two nodes that can talk to each other, can still simultaneously transmit to different receivers so long as they are transmitting on different sub-channels. [23] provides a detailed discussion on general interference patterns and the computational complexities of obtaining schedules, and [15] provides an extensive survey on scheduling and rate-control in wireless networks.

In a wireless network with the *node-exclusive* interference constraint, schedules correspond to *matchings* in the corresponding communication graph. We consider a model where each link could have packets to transmit. Two popular schedules in this context are *maximum weight matching* (MWM) schedule and *maximum size matching* (MSM) schedule. An MWM schedule assigns appropriate weights to the active links (*i.e.*, links with packets to transmit) and schedules the set of links maximizing the total weight. The weights could be chosen suitably to maximize the network throughput [25] or other system objectives such as workload [24]. The MSM schedule can be used in various ways, *eg.*, (i) schedule as many links, with backlogged packets, as possible (ii) schedule as many links as possible with more than, say, 100 packets waiting in the buffer. We also say α -approximate MWM (MSM) schedule to mean that the total weight (size) of the schedule is more than α fraction of the weight (size) of MWM (MSM).

There is a wide body of literature that provide valuable insights on MSM and MWM schedules [2], [14], [24], [24], [25], [27], but do not focus much on efficient distributed implementation of the schedules.

Our goal in this work is to compute MWM and MSM schedules using distributed algorithms that satisfy the following desirable properties:

- (i) **Local communication:** The distributed algorithm can only allow communication between a node and its communication neighbors.
- (ii) **Local topological information:** The nodes only know their respective neighbors and do not have global topological information.
- (iii) **Minimal time-complexity and minimal message transmissions:** An efficient distributed algorithm should compute the schedule very fast with minimal number of message transmissions.

In a distributed setting, computing exact schedules using algorithms that satisfy the above properties is often very hard to achieve. Thus, a worthy goal is to devise algorithms that satisfy the above properties and (i) provide provably good/constant approximations to the schedules (ii) performs close to optimal for realistic topologies. Developing such algorithms is the goal of our paper.

A. Prior Work

Distributed algorithms for computing MWM was first considered in [19]. However, the algorithm could take $n/2$ (n = number of nodes) iteration rounds to converge, and thus the time-complexity of the algorithms can be large in general.

In [17], [21], the authors devised distributed scheduling algorithms that provide 100% throughput. Distributed random access based scheduling has been considered in [10]. However, none of these algorithms guarantee a schedule that is within a constant factor (*i.e.*, independent of the network size) of MWM. Thus, while some of these algorithms provide 100% throughput, they could potentially suffer from large delay. As noted in [12] (the discussion is in [12] is in the context of a two-hop interference model, but the same logic applies here too), a constant approximation to MWM can provide superior delay for some sacrifice in throughput.

In [14], [19], the authors provide algorithms for computing maximum weight schedules, but the time-complexity of the algorithm can be polynomial in the number of nodes. In this paper, we seek algorithms with time-complexity logarithmic in the number of nodes. Closest to our work on computing MWM schedules is [26], where the authors have provided distributed algorithms for computing $1/5$ -approximate maximum-weighted matching in $O(\log^2 n)$ time-steps. We improve this to $1/2$ and also demonstrate the performance to be much better for realistic topologies. In [12], the authors consider distributed algorithms for approximating *maximum weight set* for two-hop interference models in a class of graphs called non-expanding graphs (for e.g., grid topologies are non-expanding, but trees are not).

Close to our work on computing approximate MSM schedule is the work in [11], where the authors have provided a distributed algorithm for computing $1/2$ -approximate MSM schedules with a time-complexity $O(\log^2 n)$. In one of our algorithms, we improve upon this approximation factor to $2/3$ for a similar time-complexity. In [22], the authors have devised distributed maximal matching algorithm for tree networks that achieves $2/3$ fraction of the throughput. More recently, [5] has developed low-complexity distributed algorithms for computing maximal matching (which is a $1/2$ approximate MSM). Computing approximate MSM has also been considered in [3], [9]. While these attain an approximation guarantee similar to our algorithm, the message exchange model assumes that, in every round, a node can exchange messages with *all* its neighbors which leads to $O(n^2 \log^4 n)$ message transmissions. We only allow at most one message transmission per node per round which leads to $O(n \log^2 n)$ message exchanges for our algorithms.

The issue of approximate schedules has been studied in [14]. Maximal matching schedule, which is a half approximation of MSM, has been shown to attain 50% of the maximum attainable throughput [2], [14]. In fact, the results of [12] can be used to show that, any algorithm that provides a constant factor (independent of the network size) approximation to MWM, is optimal in an order-wise delay sense.

We model the network as a graph where there is an edge between two nodes that can talk to each other. While recent work [18] points out the pitfalls of graph based models, much of these apply to a two-hop interference model.

B. Main Contributions

We define time-complexity of an algorithm as number of rounds required by the algorithm, where each round refers to a time-interval over which every node in the network can

make at most one message transmission attempt. The main contributions of our work are as follows:

Results for *maximum weight matching (MWM) schedule*: For computing MWM schedule, we have obtained the following:

- We present a distributed $(1/2 - \epsilon)$ -approximate algorithm with a time-complexity of $O(\log(1/\epsilon) \log^2 n)$.
- Simulation experiments with a popular model for wireless adhoc networks to demonstrate that the performance of the to be 85% – 95% of the optimal for typical scenarios.

Results for *maximum size matching (MSM) schedule*: We have obtained the following results for computing MSM (a schedule with maximum number of transmitting-receiving node-pairs).

- We present a $(2/3 - \epsilon)$ -approximate algorithm with a time-complexity of $O(1/\epsilon \log^2 n)$.
- Simulation experiments with a popular model for wireless adhoc networks to demonstrate that the performance of the to be 95% of the optimal for typical scenarios. Furthermore, the time-complexity can be reduced considerably in practice.

We remark that, for our distributed computation model (see Section II), to the best of our knowledge, the previously best known approximation guarantee is $1/2$ for computing MSM schedules [11], [19], and $1/5$ for computing MWM schedules [26]. The key novelty of our approach lies in distributed construction of certain auxiliary graphs that assist in intelligent message passing among neighbors to obtain the desired schedules.

The rest of the paper is structured as follows. In Section II, we describe the model and the precise problem statement. In Section IV, we provide algorithms and results for computing MSM schedules. Section III provides results for computing MWM schedules. Experimental validation of our algorithms are provided in Section V. Finally, we conclude in Section VI.

II. MODEL, ASSUMPTIONS, AND NOTATIONS

We consider a wireless network modeled by a graph $G = (V, E)$, where $V = \{1, \dots, n\}$ is the set of nodes and $E = \{(i, j) : i, j \in V\}$ is the set of node-pairs that can communicate with each other. We assume that nodes are symmetric so that if node i can talk to j , then node j can talk to node i . Let $N_u(G)$ be the neighbors of node- u in G . We will also use the notation $G(V, E, W)$ to mean a weighted graph with set of weights W associated with each edge.

A *matching* in G is a subset of the edges (link) so that no two edges in the subset share a node. A matching corresponds to a valid schedule under node exclusive spectrum sharing. We also use the notation $\text{pair}(u)$ to mean the node that node- u is matched to in a matching. We say matching M to mean, M is the set of matched edges. We also denote the operation of augmenting a matching M along path P by the operation $M \Delta P = (M \setminus P) \cup (P \setminus M)$.

The operation of the network has two phases, *schedule computation* phase, followed by *data transmission* phase. In the *schedule computation* phase, a schedule or a set of transmitter-receiver pairs is computed, and in the *data transmission* phase nodes transmit data using the computed schedule. Our focus

is on the first phase, *i.e.*, to obtain efficient schedules for data transmission.

We remark that the networks considered are multihop networks. Thus, at every scheduling instant, every packet in the node buffer has a next-hop tag. The role of the scheduling algorithm is to decide which of the links (with packets waiting to be transferred along) to activate simultaneously.

Distributed Computation Model: We assume that time is slotted and synchronized in the entire network. Synchronization can be achieved using the protocol in [4]. In each slot/round every node can transmit at most one “small” size message. The message sent by a node in the *schedule computation* phase can either be a broadcast message intended for all its neighbors, or it can be a unicast message intended for only one of its neighbors. We assume that each node is aware of its neighbors in the communication graph.

As described in [19], communications required for the realization of the computation model can be implemented using *control frames* of at most n control mini-slots in which each node is assigned a unique control mini-slot. This computation model has been also assumed in [11], [19], [26] implicitly or explicitly. Note that, the communications in the *schedule computation* phase (as described in [19]) do not have to satisfy the interference constraints imposed in schedule used in the *data transmission* phase. Basically, every data packet transmitted in the *data transmission* phase is tagged with a next-hop destination, whereas, a message transmitted in the *schedule computation* phase could be useful for all the neighbors of the transmitting node.

We point out that the transmitted messages in the *schedule computation* phase are of small sizes. As it will be clear from our algorithms, this is indeed the case. The transmitted messages simply require $O(\log \Delta)$ bits for a graph of maximum degree Δ , and typically Δ is not very large in practice.

We define the *time-complexity* of an algorithm as the number of rounds required by the algorithm for computing a schedule. We also use *with high probability* (w.h.p.) to mean “with probability at least $1 - 1/n^c$ for some $c > 0$.”

III. DISTRIBUTED COMPUTATION OF MWM SCHEDULE

We now present a distributed algorithm for computing MWM schedules. The choice of weight is an interesting question [24], [25]. For e.g., the weight could be the number of packets waiting to be transmitted along the edge. We assume that (i) every node has the knowledge of the weights of all its adjacent edges, and (ii) the weights are poly(n) quantities. For e.g., if the weights are the number of packets waiting to be transferred along an edge and this quantity is clearly constrained by the buffer sizes at the nodes; in fact, the weights are more likely to be $O(1)$ quantities in this case.

The main novelty in the algorithm is to generate certain auxiliary graphs using local message exchanges, and using the auxiliary graphs to improve the weight of an existing matching. Our algorithm involves finding matching in these auxiliary graphs. While any distributed algorithm can be used for this, we adapt an algorithm in [26] for this purpose.

We define the notion of *I-augmentation* paths in a weighted graph $G(V, E, W)$ with matching M . A *I-augmentation* path P is a path of length at most three, that has (a) only one

Algorithm 1 $\frac{1}{2} - \epsilon$ Approximate Maximum Weight Matching (Given a communication graph $G(V, E, W)$, it computes a MWM in a distributed manner)

- 1: Start with a maximal matching on G . Let M be the set of matched edges.
 - 2: **for** k iterations **do**
 - 3: Each node broadcasts the weight of its associated matched edge.
 - 4: Each node calculates the gain from *I-augmentation* paths along its adjacent unmatched edges. Every node marks the neighbors that have a positive gain *I-augmentation* path. Let M' be the matched nodes that have positive gain *I-augmentation* paths through them, and let U' be such unmatched nodes.
 - 5: The nodes M' and U' run *Modified_Approximate_MWM* as given by Procedure 2. Let P be the set of disjoint *I-augmentation* paths returned by the procedure *Modified_Approximate_MWM*. {this is obtained by running approximate maximum weight matching on the auxiliary graph $A_{M', U'}$ }
 - 6: $M \leftarrow M \Delta P$.
 - 7: **end for**
-

unmatched edge (b) matched and unmatched edges alternate in the path, and (c) if the unmatched and the matched edges are swapped (using the operation $M \leftarrow M \Delta P$), the total weight of the matching increases. A *I-augmentation* path can either have two, or one, or zero matched edges. Thus, *I-augmentation* paths can be of one of the following form: (i) either (v_1, v_2, v_3, v_4) for $v_i \in V$, where (v_1, v_2) and (v_3, v_4) are matched edges, (v_2, v_3) is an unmatched edge, or (ii) (v_1, v_2, v_3) , where (v_1, v_2) is a matched edge, v_3 is an unmatched node, or (iii) (v_1, v_2) , where (v_1, v_2) is an unmatched edge and both v_1 and v_2 are unmatched nodes. Let M_P be the set of matched edges in a *I-augmentation* path P , and let f_P be the only unmatched edge in P . We also define the gain of a *I-augmentation* path as follows:

$$\text{gain}(I\text{-augmentation path } P) = \text{weight}(f_P) - \sum_{e \in M_P} \text{weight}(e)$$

We now describe the algorithm and also give the insight behind it. We will introduce some notations that help in describing the algorithms. Let M' (resp. U') be the set of matched (resp. unmatched) nodes with positive gain *I-augmentation* paths going through them, and let $\text{pair}(m)$ be the node that $m \in V$ is matched to. Clearly, $M', U', \text{pair}(m)$ may vary over the course of the algorithm. The algorithm has two stages that are repeated $C \log(1/\epsilon)$ times for some constant $C > 0$: the first stage consists of forming a certain *auxiliary* graph, and the second stage consists of adapting a known max-weight matching algorithm to the auxiliary graph. The two stages are as follows:

1. Forming Auxiliary Subgraph: The first stage (step 3 to step 4 of Algorithm 1) involves forming a weighted auxiliary graph in a distributed manner. The auxiliary graph, $A_{M', U'}$, has two kinds of nodes: nodes formed by collapsing every matched pair $(m, \text{pair}(m))$, $m \in M'$ (denoted by $\text{coll}(m)$) and every node in U' . There is an edge between two nodes in $A_{M', U'}$ if the nodes share a *I-augmentation* path, and the weight of the edge is the gain of the corresponding *I-augmentation* path. Distributed construction of $A_{M', U'}$ goes as follows. First,

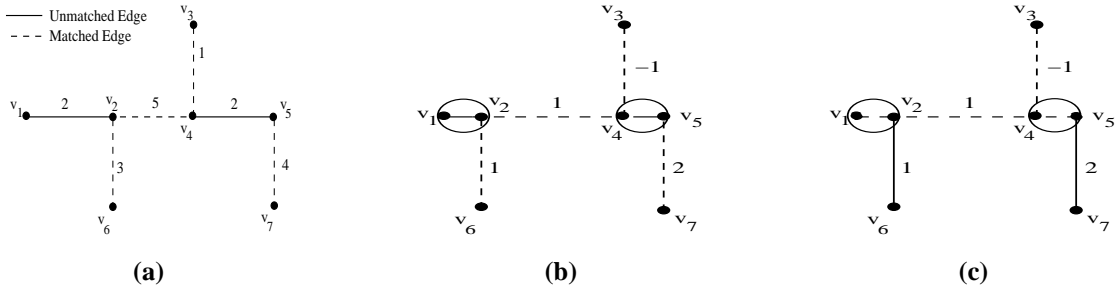


Fig. 1. (a) A graph G with certain matched edges and the edge-weights. (b) the corresponding auxiliary graph and the gains of the edges in the auxiliary graph. (c) augmentation along a disjoint set of 1 -augmentation paths.

Procedure 2 *Modified_Approximate_MWM* : Modified $\frac{1}{5}$ -Approximate Maximum Weight Matching (Given $A_{M',U'}$, this outputs a set of disjoint 1 -augmentation paths with total gain from the 1 -augmentation paths within $1/5$ of the maximum (over all disjoint sets of 1 -augmentation paths) gain.)

- 1: **for** ($\log n$) iterations **do**
- 2: Each node $m \in M'$ transmits to $\text{pair}(m)$ the maximum gain of its 1 -augmentation paths.
- 3: Each node $m \in M'$ broadcasts the maximum gain of the 1 -augmentation paths of m and $\text{pair}(m)$.
- 4: Each node $u \in U'$ broadcasts the maximum of its 1 -augmentation paths. Every $u \in M' \cup U'$ computes $g_{\max}(u)$ as the maximum gain of all 1 -augmentation paths of its neighbors.
- 5: A new graph G'' is formed by the nodes in $M' \cup U'$ such the neighborhood set of any node v in G'' is given by

$$N_v(G'') = \{w \in M' \cup U' : \exists 1\text{-augmentation path through } w \text{ and } v \text{ with gain at least } \frac{g_{\max}(v)}{2}\}.$$

- 6: Run *Maximal_Matching* (Appendix VII-C) on the auxiliary graph A' obtained by collapsing the matched edges in G'' . Let E'' be the set of matched edges returned by the matching A' ; each such edge corresponds to a 1 -augmentation path. {this procedure can be performed in a distributed manner as outlined in Lemma 3.3; also note that E'' consists of unmatched edges in the 1 -augmentation paths}
- 7: Do the following updates:

$$M' \leftarrow M' \setminus \{v : \exists w \text{ such that } (v, w) \in E'' \text{ or } (w, v) \in E''\}$$

$$U' \leftarrow U' \setminus \{v : \exists w \text{ such that } (v, w) \in E'' \text{ or } (w, v) \in E''\}.$$

{this amounts to next iteration being performed only by the remaining nodes in M' and U' }

- 8: **end for**
- 9: **return**(the set of all 1 -augmentation paths corresponding to matchings in A')

each node computes the gains of all the 1 -augmentation paths going through it. To do this gain computation in a distributed manner, note that it is sufficient for every matched node to broadcast the weight of its matched edge (this is easy to see by considering the different structures of a 1 -augmentation path). Next, the negative gain 1 -augmentation paths are removed from consideration by the nodes. Since we have a distributed algorithm, we mimic the functioning of the auxiliary graph by making the matched node-pairs function as a single node (see Procedure 2 for details). An illustration of this is shown in Figure 1 (b). The important point to note is that, a disjoint set

of 1 -augmentation paths in the original graph corresponds to a matching in the auxiliary graph as illustrated in Figure 1 (c). We capture this fact in the form of the following lemma.

Lemma 3.1: Any matching in $A_{M',U'}$ corresponds to a disjoint set of 1 -augmentation paths in the graph G and vice-versa.

2. Approximate matching in the auxiliary graph: The second stage (step 5 to step 6 of Algorithm 1) involves applying any approximate maximum weight matching algorithms to the auxiliary graph $A_{M',U'}$. We adapt the $\frac{1}{5}$ -Approximate Maximum Weight Matching algorithm in [26] to the auxiliary graph $A_{M',U'}$. The matching is obtained (step 5 of Algorithm 1) by Procedure 2 which essentially mimics the effect of finding matching in $A_{M',U'}$ by treating the node-pairs in M' as a single-node. Since this obtains an approximate maximum weight (where weights are the gains of the corresponding 1 -augmentation paths) matching of the auxiliary graph $A_{M',U'}$, we have the following after combining with Lemma 3.1.

Lemma 3.2: Procedure 2 results in finding a disjoint set of 1 -augmentation paths in the original graph such that total gain in weight of the matching due to these 1 -augmentation paths is no less than $1/5$ of the maximum gain possible.

The current matching is then augmented along these disjoint 1 -augmentation paths (step 6 of Algorithm 1).

Clearly, by repeating the first two stages, we keep augmenting high gain 1 -augmentation paths, and hence we move closer to a $1/2$ -approximate maximum weight matching.

A. Main Result

Theorem 1: If we set $k = C \log(1/\epsilon)$ for some constant $C > 0$, then, Algorithm 1 produces a $(\frac{1}{2} - \epsilon)$ -approximate Maximum Weight Matching in $O(\log(1/\epsilon) \log^2 n)$ rounds *w.h.p.*

Before we prove the result, we will first argue the correctness of Step-6 of Procedure 2.

Lemma 3.3: In Procedure 2, step-6 can be performed in a distributed manner in $O(\log n)$ rounds using our computational model described in Section II.

Proof: See Appendix VII-B.

To prove the correctness of our algorithm, we will use the following lemma which follows from a result in [20]. Let $w(M)$ denote the total weight of matching M .

Lemma 3.4: [20] For any matching M , there exists a collection of vertex-disjoint 1 -augmentation paths such that

$$w(M \triangle A) \geq w(M) + \frac{2}{3} \left(\frac{1}{2} w(\text{MWM}) - w(M) \right).$$

Proof of Theorem 1:

Let P_i be the collection of augmentations chosen in the i^{th} iteration and let M_i be the corresponding matching after augmenting along the paths in P_i . Let P^* be the disjoint set of 1-augmentation paths that give maximum possible gain. By Lemma 3.4,

$$\text{gain}(P^*) \geq \frac{2}{3} \left(\frac{1}{2} w(M^*) - w(M_i) \right).$$

Also, by Lemma 3.2, we have $\text{gain}(P_i) \geq (1/5)\text{gain}(P^*)$. We thus have

$$\text{gain}(P_i) \geq \frac{2}{15} \left(\frac{1}{2} w(M^*) - w(M_i) \right)$$

Letting Y_i be the weight of the matching after i iterations the preceding can be written as $Y_{i+1} - Y_i \geq \frac{2}{15}(\bar{w} - Y_i)$. Now let $X_i = \bar{w} - Y_i$, where $\bar{w} = (1/2)w(M^*)$. We now have the following.

$$X_0 \leq \bar{w}, X_{k+1} \leq \frac{13}{15} X_k \leq \left(\frac{13}{15} \right)^k \bar{w}$$

It follows that, there exists $C > 0$ such that $X_k = \bar{w}\epsilon$ for $k \geq C \log(1/\epsilon)$ and hence $Y_k = \bar{w}(1 - \epsilon)$ for $k \geq C \log(1/\epsilon)$.

The results follow with high probability since the maximal matching produced by step-6 of Procedure 2 happens with high probability. Note that, $O(\log n)$ occurrences of high probability events translate to all the events occurring *w.h.p.*

IV. DISTRIBUTED COMPUTATION OF MSM SCHEDULE

We now describe algorithms for computing MSM. MSM schedule could be used to schedule as many links, that satisfy a certain desired property, as possible. The property could be (i) links with packets to transmit (ii) links with more than a certain number of packets in the buffer etc.. Throughout this section, we say communication graph to imply the sub-graph with only those links/edges that satisfy the desired property.

A. Intuition and Overview of the Algorithm

We first define the notion of 2-augmentation paths in a graph G with some matching M . A path (v_1, v_2, v_3, v_4) (clearly, $\{(v_1, v_2), (v_2, v_3), (v_3, v_4)\} \subset E$) is a 2-augmentation path if it satisfies the following property:

$$\text{(P)} \quad (v_1, v_2) \notin M, (v_2, v_3) \in M, (v_3, v_4) \notin M, \\ v_1 \text{ and } v_4 \text{ are unmatched nodes.}$$

Essentially, if we replace the matched edge in an 2-augmentation path by the two unmatched edges in the path, we still have a valid matching. This operation is called augmenting the matching M along an augmentation path P and can be formally stated as $M \leftarrow (M \setminus P) \cup (P \setminus M)$. The algorithm we develop is based on the following standard fact from graph theory.

Lemma 4.1: [20] If M is a matching in graph $G(V, E)$ such that there are no 2-augmentation paths, then,

$$|M| \geq \frac{2}{3} |\text{MSM in } G(V, E)|. \quad \square$$

We will also use the following result based on an algorithm in [11], [26]. The algorithm uses the computation model in Section II.

Lemma 4.2: [26] In any graph, the Algorithm provided in Appendix VII-C [11], [26] computes a maximal matching in $O(\log n)$ rounds *w.h.p.* \square

Remark 1: At the end of the maximal matching algorithm (see Appendix VII-C), every matched pair has a left-right orientation.

We will give an overview of our algorithm for computing 2/3 approximation to MSM schedule. The algorithm runs in two phases. The first phase involves computing a maximal matching in a distributed manner using the algorithm in Appendix VII-C [11], [26]. In the second phase, the algorithm essentially tries to get rid of as many 2-augmentation paths as possible in a distributed manner and with only one transmission per node per round.

The first phase ensures that there are no unmatched nodes with unmatched neighbors. Thus, if U is the set of all unmatched nodes, then the 2-augmentation paths can be discovered by obtaining all unmatched node pairs having a common matched edge as a neighbor. In other words, if $v_1, v_4 \in U$, and there is a matched edge (v_2, v_3) such that $v_2 \in N_{v_1}$ and $v_3 \in N_{v_4}$, then augmenting the matching along the path (v_1, v_2, v_3, v_4) gets rid of the 2-augmentation path. Now, if the 2-augmentation paths at the end of the first phase are all disjoint (*i.e.*, no two 2-augmentation paths have common nodes), then, removing the 2-augmentation paths in a distributed manner is straight-forward:

- 1) Every unmatched node transmits a message to its matched neighbor declaring that it is unmatched (this can be achieved by a broadcast message).
- 2) Every matched node m sends a message to its matched pair (from the matching formed in the first phase) declaring its unmatched neighbor u_m , if any (this information is obtained from step-1).
- 3) For every matched node m that has heard a signal in step-1 and step-2, clearly there is a 2-augmentation path going through it, and hence the matched partner can be simply substituted by the unmatched node learned from step-1.

However, note that there can be multiple 2-augmentation paths v_1 and v_4 can be part of, and all of these cannot be removed simultaneously. A simple example of this is given in Figure 2(a).

In such a scenario, the preceding simple algorithm fails in the last step where a matched node has to choose between multiple 2-augmentation paths. In the algorithm we develop, the goal of the second phase is to remove as many 2-augmentation paths as possible. To do this we will form the following auxiliary tripartite graph in a distributed manner. The auxiliary tripartite graph has three sets of vertices L , C and R . The L and R vertices are mutually disjoint subsets of the unmatched nodes U , and the C vertices are obtained by collapsing the existing matched edges of the graph. We will call this new graph A (see Figure 2(b) for example). Now note that, any 3D matching in A correspond to vertex disjoint 2-augmentation paths in G . In Figure 2(c), we show a 3D-matching in the auxiliary tripartite graph. As it can be seen, the 3D-matching in the tripartite graph corresponds to disjoint 2-augmentation paths in G .

With this intuition, the second phase of the our algorithm

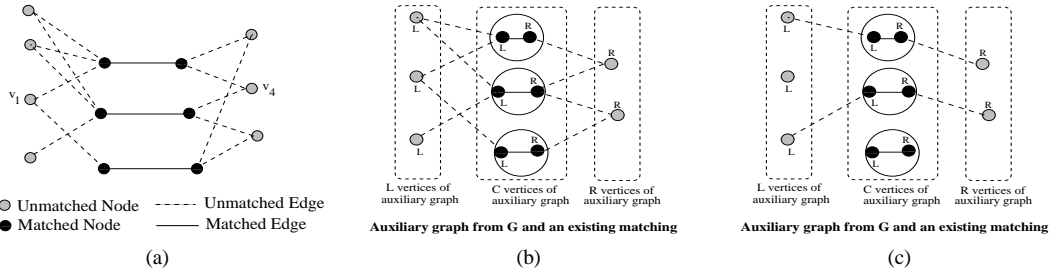


Fig. 2. (a) A graph with multiple overlapping 2-augmentation paths. (b) An auxiliary graph G_T based on the matching in G . Note that, if the matched edges are collapsed into a single node, then we have a tripartite graph. (c) A 3D-matching in the corresponding tripartite graph.

works as follows.

- 1) Form the auxiliary tripartite graph A in a distributed manner. This can be done by every unmatched node randomly deciding to be an L -vertex or an R -vertex in a manner described in the next subsection. The edges are the edges going from L to C and C to R .
- 2) In the tripartite graph thus formed obtain 3D matching in a distributed manner. The details of this step is again given in the detailed description of the algorithm.
- 3) Repeat step-1 and step-2 $O(\log n)$ times.

Essentially, the first step ensures that the tripartite graph captures a constant fraction of the 2-augmentation paths in G that are removed in step-2. Thus, repeating the steps $O(\log n)$ times removes “most” (this will be quantified later) of the 2-augmentation paths.

B. Detailed Description of the Algorithm

We first introduce some notations that will help in describing the algorithms. We use U for set of unmatched nodes U , M for set of matched nodes M and E_M for matched edges. Every node in U and M is marked L (left) or R (right) as the algorithm progresses. Let L_U, R_U, L_M, R_M respectively be the set of unmatched L-nodes, set of unmatched R-nodes, matched L-nodes, set of matched R-nodes. Note that, all these sets change dynamically over the course of the algorithm.

The algorithm runs in two Parts.

In Part 1 (step 1 of Algorithm 3) we run a maximal matching algorithm on the input graph using the algorithm provided in [11]. This algorithm ensures that all nodes in the graph are either matched or have all matched neighbors.

Part 2 (step 2-14 of Algorithm 3) consists of k_1 repetitions of the following procedure, where the appropriate value of k_1 is derived in a later subsection. The procedure is further divided into two subparts, Part 2(a) and Part 2(b).

- In Part 2(a) (step 5-6 of Algorithm 3) we form an auxiliary graph by letting each unmatched node choose to be left (L-node) or right (R-node). Further, each matched pair is already oriented left-right or right-left (see Remark 1). The neighbor set of the auxiliary subgraph is chosen as follows: each unmatched (matched) L-node chooses the matched (unmatched) L-nodes from its neighbor set in the original graph, and similarly for the R-nodes. This step is illustrated in Figure 2(b). Essentially, if we now collapse the matched pairs to be single nodes we have a tripartite graph A as shown in the figure. Now we see that any 3-D matching in the graph A corresponds

to disjoint 2-augmentation paths in the original graph. The next step finds these 2-augmentation paths in a distributed manner.

- In Part 2(b) (step 7-11 of Algorithm 3) the algorithm attempts to discover a maximal set of 3-D matchings in the auxiliary graph A (formed by collapsing the matched node-pairs in the auxiliary graph). This requires k_2 repetitions of the following procedure. First remove any edge along which a 2-augmentation path is not possible. We then run a maximal matching in the bipartite graph formed by the unmatched L-nodes and the matched L-nodes. Call this matching M_L . This is followed by a maximal matching on the bipartite graph formed by a subset of the matched R-nodes (this subset consists of those nodes whose partners are in M_L) and the unmatched R-nodes. Call this matching M_R . It is not hard to see that M_L and M_R combine to give a set of disjoint 2-augmentation paths (this is illustrated in Figure 2(c)). The existing matching is then augmented along these 2-augmentation paths.

C. Main Results

We will now argue that, by setting the parameters k_1 and k_2 suitably, our Algorithm gives a $2/3 - \epsilon$ approximation to maximum size matching. We have the following main result.

Theorem 2: If we set $k_1 = C_1 \log n$ and $k_2 = C_2 \frac{1}{\epsilon}$ for positive constants C_1 and C_2 , then, Algorithm 3 gives us a $(\frac{2}{3} - \epsilon)$ -approximation to Maximum Size Matching in $O(1/\epsilon \log^2 n)$ w.h.p.

We shall now prove the above result through a sequence of lemmas. In the following, we will call Step 3 to Step 13 one phase of the algorithm, and we will call Step 8 to Step 10 of the algorithm as one iteration in a phase.

Lemma 4.3: Let S_{\maximal} (S_{\maximal}) denote a maximal (maximum) set of 2-augmentation paths that is found in the auxiliary graph. Then,

$$|S_{\maximal}| \geq \frac{1}{3} |S_{\maximal}|$$

Proof: Follows by noting that that each path in S_{\maximal} can be mapped to at most three paths in S_{\maximal} . We skip the details. ■

Lemma 4.4: In the auxiliary graph, if after t iterations the size of the maximum disjoint set of 2-augmentation paths in the auxiliary graph is r , then the matching contains at least $\frac{tr}{2}$ matched pairs.

Proof: First note that, any matched pair that is in the maximal matching in the left subgraph of the auxiliary graph

Algorithm 3 $\frac{2}{3}$ -Approximate Maximum Size Matching (Given a communication graph G , it computes a MSM in a distributed manner)

- 1: Run Maximal Matching on the input graph G using the distributed maximal matching algorithm in Appendix VII-C. {Each matched edge has one left marked node and one right marked node. We call them matched L-node and matched R-node (see Remark 1)}
 - 2: **for** k_1 phases **do**
 - 3: Let U = set of unmatched nodes, M = set of matched nodes.
 - 4: Each $u \in U$ chooses to mark itself as L with probability 0.5 and R with probability 0.5.
 - 5: Let P be the set of 2-augmentation paths. Initialize $P := \emptyset$
 - 6: Run *Reduce_Graph* (Procedure 4) on G . This outputs an auxiliary tripartite graph $A = (V_A, E_A)$.
 - 7: **for** k_2 iterations **do**
 - 8: The nodes that are part of A run *3D_Matching* (Procedure 5) to get a disjoint set of 2-augmentation paths T .
 - 9: $P \leftarrow P \cup T$ {essentially, for all $t \in T$, the nodes that are part of t store the neighbors in the path t .}
 - 10: The nodes that are part of paths in T remove themselves from V_A . {the nodes that are removed from V_A can broadcast a *leave* message so that the other nodes in V_A update their neighborhood in graph A appropriately}
 - 11: **end for**
 - 12: Restore all edges to graph G
 - 13: $M \leftarrow M \Delta P$ {since the 2-augmentation paths are disjoint, this step can be easily carried out in a distributed manner.}
 - 14: **end for**
-

Procedure 4 *Reduce_Graph* : Generate Auxiliary Graph

- 1: Initialize $N_u(A) := \emptyset$ for all $u \in U$ and $N_m(A) := \emptyset$ for all $m \in M$.
 - 2: Each node broadcasts its marking, which is either L or R .
 - 3: Each node $m \in R_M$ such that $N_m(G) \cap R_U \neq \emptyset$ (i.e., M has at least one right marked unmatched neighbor) sends an *exist* message to its matched pair. Similarly, each node $m \in L_M$ such that $N_m(G) \cap L_U \neq \emptyset$ sends an *exist* message to its matched pair.
 - 4: Every node $m \in R_M$ (L_M) that participated in the previous step and that gets an *exist* signal from $\text{pair}(m)$ includes in $N_m(A)$ all the nodes in R_U (L_U) it has heard from.
 - 5: Every $m \in M$ such that $N_m(A) \neq \emptyset$ sends a broadcast message *include*. Every node $u \in L_U$ ($u \in R_U$) includes in $N_u(A)$ each node $m \in L_M$ ($m \in R_M$) that has sent a broadcast.
-

will be removed from the auxiliary graph (i.e., will not be part of any 2-augmentation path) after the iteration.

So let us now consider the total number of matched pairs removed after t iterations. Let r_i be the size of the maximum disjoint set of 2-augmentation paths left in the reduced graph after the i^{th} iteration. Clearly, r_i is decreasing in i and hence $r_i \geq r_t = r$. Now, if the size of the maximum disjoint set of 2-augmentation paths is r_i after any iteration, then the size of the maximal matching in the left subgraph is at least $r_i/2$. Furthermore, the matched edges removed in two different iterations are disjoint. This is because, a matched node in a 2-augmentation path that is included in the maximal matching of the left subgraph (of the auxiliary graph) in an iteration, is no more a part of any 2-augmentation path. Thus the total number of matchings in these t iterations should be at least $\sum_{i=1}^t r_i/2 \geq rt/2$. Hence the result. ■

Lemma 4.5: If $k_2 = 24/\epsilon$, then, after $C_1 \log n$ phases of

Procedure 5 *3D_Matching* : Find 3D Matching in the Auxiliary Graph

- 1: Run *Maximal_Matching* on the subgraph of A formed by the nodes $L_U \cup L_M$ using the distributed maximal matching algorithm in [11]. Denote by M'_L the nodes that are part of this matching.
 - 2: Every $m \in L_M \cap M'_L$ sends a *matched* message to $\text{pair}(m)$ (this is the matched partner in the matching M in the original graph) in R_M . Let $S_R \subseteq R_M$ be the set of nodes that has heard.
 - 3: Run *Maximal_Matching* on the subgraph of A formed by nodes in $S_R \cup R_U$ using the maximal matching algorithm in [11]. Let M'_R be the nodes that are part of this matching.
 - 4: Now each node that is contained in $M'_L \cup M'_R$ is obtained in a 2-augmentation path and each one is disjoint. Augment the matching M using these 2-augmentation paths.
 - 5: The above procedure is repeated starting from the right subgraph.
 - 6: The set of all 2-augmentation paths discovered is returned.
-

the Algorithm 3 we have a $\frac{2}{3} - \epsilon$ approximation to maximum size matching in the graph *w.h.p.*

Proof: Let X_k, r_k respectively be the size of the maximum disjoint set of 2-augmentation paths in the beginning of phase- k and remaining after phase- k . Clearly $X_{k+1} - X_k$ is the number of 2-augmentation paths removed from the auxiliary graph in phase- k . Now, each 2-augmentation path can be a part of the auxiliary graph with probability 1/4; because this happens if unmatched neighbor of the matched L-node in the path is an L-node (occurs with probability 1/2) and also the unmatched neighbor of the matched R-node in the path is an R-node (occurs with probability 1/2). Thus, the number of 2-augmentation path in the auxiliary graph in phase- k distributed as Binomial($X_k, 1/4$). Since, the $X_{k+1} - X_k$ 2-augmentation paths removed from the auxiliary graph along with the r_k 2-augmentation paths remaining in the auxiliary graph form a maximal set of 2-augmentation paths in the auxiliary graph, it follows from Lemma 4.3 that

$$E[X_{k+1} - X_k] + r_k \geq \frac{1}{3} \frac{E[X_k]}{4} \Rightarrow E[X_{k+1}] \leq \frac{11}{12} E[X_k] + r_k.$$

It can be shown using Lemma 7.1 in Appendix VII-A that $X_k \leq 12r_{\max}$ for $k = \Omega(\log n)$ *w.h.p.*, where r_{\max} is the maximum value of r_k over $C_1 \log n$ phases.

Consider the phase after which there were r_{\max} 2-augmentation paths left in the auxiliary graph after k_2 iterations. Since $k_2 = 24/\epsilon$, we also know from Lemma 4.4 $|M| \geq 12r_{\max}/\epsilon$. Also, since we have left out at most $12r_{\max}$ 2-augmentation paths after $C_1 \log n$ phases, and augmentation along each of these paths can add one extra edge to the matching, we have from Lemma 4.1

$$|M| + 12r_{\max} \geq \frac{2}{3} |M^*|, \quad (1)$$

where M^* is any maximum size matching. This along with $|M| \geq 12r_{\max}/\epsilon$, can be used to argue that $|M| \geq (2/3 - \epsilon) |M^*|$. ■

Proof of Theorem 2: The theorem follows from Lemma 4.5. We simply note that the number of rounds required is $k_1 k_2 \log n = O(1/\epsilon \log^2 n)$ since computing the maximal matching takes $O(\log n)$ rounds.

V. EXPERIMENTS

In this section we show results of running our algorithms on realistic graphs. The purpose of this sections is two-fold. First, to demonstrate that the performance of our algorithms for certain realistic graphs are much better than guaranteed by our theoretical analysis for arbitrary graphs. Second, to provide insight on the time-complexity of our algorithm for these graphs. The time-complexity in our theoretical results simply underline the *orders of magnitude*, we demonstrate the actual time-complexities of our algorithms in this section.

A. Setup

The communication graphs on which we ran our algorithms were generated by throwing the nodes uniformly at random in a square of unit area. Furthermore, each node was assumed to have an identical communication radius of r . Such graphs are also called random geometric graphs and have been widely used as a model for wireless ad-hoc networks [7], [13]. It was proved in [6] that the critical radius required for connectivity in such a graph scales with the number of nodes n as $\frac{1}{\pi}\sqrt{\log n/n}$. For our experiments, we took the communication radius of the nodes $\frac{c}{\pi}\sqrt{\log n/n}$ for different values of the constant $c > 0$.

The experiments were performed using a simulator written in C to test the protocol and the results were averaged over nine different random graphs and 15 simulation runs on each graph-instant.

B. Results for Maximum Weight Matching

We now demonstrate the performance of our maximum weight matching algorithm. For these set of experiments on random geometric graphs, the weight for each edge was chosen uniformly at random in the range $[1, 1000]$.

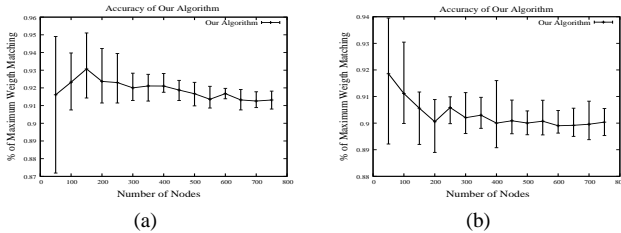


Fig. 3. Performance of Algorithm 1 in terms of %-of from optimal on random geometric graphs with communication radius equal to (a) $\frac{2.1}{\pi}\sqrt{\log n/n}$ (b) $\frac{3}{\pi}\sqrt{\log n/n}$.

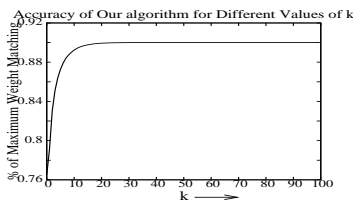


Fig. 4. Plots showing the performance of Algorithm 1 for different values of k . The time-complexity of the algorithm is roughly $k \log^2 n$.

Performance: In Figure 3(a) and 3(b), we have shown the performance of Algorithm 1 for varying number of nodes and for different values of the communication radius. Clearly, for

random geometric graphs, the performance of Algorithm 1 is close to 90% of the optimal value. For instance the least average result observed has 89% accuracy. This is a substantial improvement over our theoretical guarantee of around 50%. Also, the worst case (over several random instances) performance is close to 87%.

Time-complexity: Recall that the time-complexity of Algorithm 1 depends on the number of phases k . The time-complexity of the algorithm is $k \log^2 n$. In Figure 4, we have shown the performance for different values of k for $n = 500$ nodes. Clearly, the additional gain beyond $k = 20$ is minimal. This can be observed as for $k = 20$ we achieve accuracy of up to 88% and the best possible result is within 89% of the solution. Thus, in practice, a small value of k may suffice.

C. Results for Maximum Size Matching

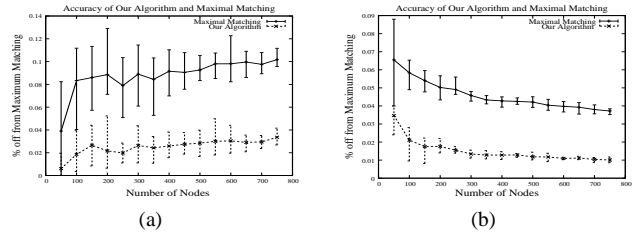


Fig. 5. Performance of Algorithm 3 in terms of %-off from optimal on random geometric graphs with communication radius equal to (a) $\frac{1.12}{\pi}\sqrt{\log n/n}$ (b) $\frac{3}{\pi}\sqrt{\log n/n}$.

Performance: In Figure 5(a) and 5(b), we compare the performance of Algorithm 3 with a simple maximal matching algorithm. On y -axis we have shown the gap between (and a maximal matching algorithm) algorithm and MSM matching (in the figure, we use “Maximum Matching” to mean MSM). Clearly, the gap between maximum size matching and the size of the matching produced by Algorithm 3 is less than 5%, *i.e.*, typically the performance is within 95% of the optimal for random geometric graphs. In most of these cases a simple maximal matching algorithm also performs quite well, but the average gap from the optimal matching is almost twice that of Algorithm 3. Also, the performance of the simple maximal matching algorithm shows much more variability as can be seen from the error-bars in our plots in Figure 5(a) and 5(b).

We now comment on the time-complexity of our algorithm. Recall that, Algorithm 3 has two parameters k_1 and k_2 where k_1 is the number of *phases* and k_2 is the number of iterations within each phase. The time-complexity is clearly $k_1 k_2 \log n$. While Theorem 2 provides bounds for k_1 and k_2 for general graphs, our simulation results with random geometric graphs suggest that, $k_1 = 10$ and $k_2 = 2$ suffice for maximum performance from our algorithm.

VI. CONCLUDING REMARKS

In this paper we have developed approximate scheduling algorithms for wireless networks under *node exclusive spectrum* sharing. We have focused on two popular schedules, namely *maximum size matching* and *maximum weight matching* schedules, and provided improved approximate algorithms for computing these schedules in a distributed manner. Our algorithms have logarithmic time-complexity, and simulations results with a popular model for wireless adhoc networks

suggest that the algorithms perform up to 95% of the optimal in practice.

REFERENCES

- [1] IEEE 802.16e standard, <http://standards.ieee.org/getieee802/802.16.html>.
- [2] P. Chaporkar, K. Kar, and S. Sarkar. Throughput guarantees through maximal scheduling in wireless networks. In *43rd Annual Allerton Conf on Communications, Control, and Computing*, Sep 2005.
- [3] A. Czygrinow, M. Hanczowski, and E. Szymanska. A fast distributed algorithm for approximating the maximum matching. In *Proceedings of European Symposium on Algorithms (ESA)*, 2004.
- [4] A. Giridhar and P. R. Kumar. Distributed clock synchronization over wireless networks: Algorithms and analysis. In *45th IEEE Conference on Decision and Control*, pages 4915–4920, Dec 2006.
- [5] A. Gupta, X. Lin, and R. Srikant. Low-complexity distributed scheduling algorithms for wireless networks. In *INFOCOM*, 2007.
- [6] P. Gupta and P. Kumar. Critical power for asymptotic connectivity in wireless networks. *Stochastic Analysis, Control, Optimization and Applications: A Volume in Honor of W.H. Fleming, W.M. McEneaney, G. Yin, and Q. Zhang (Eds.)*, Birkhauser, Boston, 1998.
- [7] P. Gupta and P. Kumar. Capacity of wireless networks. *IEEE Transactions on Information Theory*, 46(2), 2000.
- [8] B. Hajek and G. Sasaki. Link scheduling in polynomial time. *IEEE Transactions on Information Theory*, 34(5):910–917, 1988.
- [9] M. Hanczowski, M. Karonski, and A. Panconesi. On the distributed complexity of computing maximal matchings. In *SODA*, 1998.
- [10] C. Joo and N. Shroff. Performance of random access scheduling schemes in multi-hop wireless networks. In *INFOCOM*, 2007.
- [11] K. M. Jung and D. Shah. Fast averaging via non-reversible random walk. In *Proceedings of Information Theory Workshop*, 2006.
- [12] K. M. Jung and D. Shah. Low delay scheduling in wireless networks. In *Proceedings of IEEE ISIT*, 2007.
- [13] J. Li, C. Blake, Douglas S. J. De Couto, H. I. Lee, and R. Morris. Capacity of ad hoc wireless networks. In *Mobile Computing and Networking*, pages 61–69, 2001.
- [14] X. Lin and N. B. Shroff. The impact of imperfect scheduling on cross-layer rate control in wireless networks. In *INFOCOM*, volume 3, pages 1804–1814, 2005.
- [15] X. Lin, N. B. Shroff, and R. Srikant. A tutorial on cross-layer optimization in wireless networks. *IEEE Journal on Selected Areas in Communications*, 24(8):1452–1463, 2006.
- [16] B. A. Miller and C. Bisdikian. *Bluetooth Revealed: The Insider's Guide to an Open Specification for Global Wireless Communications*. Prentice Hall, 2001.
- [17] E. Modiano, D. Shah, and G. Zussman. Maximizing throughput in wireless networks via gossiping. In *ACM SIGMETRICS*, pages 27–38, 2006.
- [18] T. Moscibroda, R. Wattenhofer, and Y. Weber. Protocol Design Beyond Graph-Based Models. In *Hotnets*, 2006.
- [19] A. Penttinen, I. Koutsopoulos, and L. Tassiulas. Low-complexity distributed fair scheduling for wireless multi-hop networks. In *First Workshop on Resource Allocation in Wireless Networks (RAWNET), WiOpt*, 2005.
- [20] S. Pettie and P. Sanders. A simpler linear time $2/3 - \epsilon$ approximation for maximum weight matching. *Information Processing Letters*, 91(6):271–276, 2004.
- [21] S. Sanghavi, L. Bui, and R. Srikant. Distributed link scheduling with constant overhead. In *ACM SIGMETRICS*, 2007.
- [22] S. Sarkar and K. Kar. Achieving $2/3$ throughput approximation with sequential maximal scheduling under primary interference constraints. In *44th Annual Allerton Conference on Communication, Control and Computing, Allerton, Monticello, Illinois*, September 2006.
- [23] G. Sharma, R. R. Mazumdar, and N. B. Shroff. On the complexity of scheduling in wireless networks. In *ACM Mobicom*, 2006.
- [24] A.L. Stolyar. Maxweight scheduling in a generalized switch: State space collapse and workload minimization in heavy traffic. *Annals of Applied Probability*, 14, 2004.
- [25] L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Transactions on Automatic Control*, 37(12), 1992.
- [26] M. Wattenhofer and R. Wattenhofer. Distributed weighted matching. In *Proceedings of DISC*, 2004.
- [27] X. Wu, R. Srikant, and J. R. Perkins. Queue-length stability of maximal greedy schedules in wireless networks. In *Workshop on Information Theory and Applications, UCSD*, 2006.

VII. APPENDIX

A. A Technical Lemma

Lemma 7.1: If a sequence of non-increasing integer random variables X_k satisfy $E[X_{k+1}|X_k] \leq \phi X_k$, $X_0 \leq n$ for some $\phi < 1$, then there exists a constant $C > 0$ such that for all $k \geq C \log n$, $X_k = 0$ w.h.p..

Proof: The proof is simple and follows from Markov Inequality. We skip the proof for want of space. \square

B. Proof of Lemma 3.3

Step 6 in Procedure 2 is not too different from the distributed maximal matching algorithm in [11] except for some minor technicalities owing to the fact that the graph A' is obtained by collapsing certain matched node pairs belonging to M' (we use the notations in Procedure 2). We outline these technicalities in the following. The distributed maximal matching algorithm in [11] proceeds in a request-grant fashion: in every round nodes choose to be left (L) or right (R) at random, the unmatched L-nodes send a request for matching to one of the unmatched R-nodes, followed by the unmatched R-nodes picking up a request at random. This can be easily mimicked in the auxiliary graph A' . For every unmatched node in G' that is also in A' there is no difference to the maximal matching algorithm in [11]. Consider a node v that is essentially formed by collapsing two nodes m and $m' = \text{pair}(m)$. In this case, the only difference comes when an L-node in A' , that is formed by collapsing m and m' , has to request (or grant a request if it is a right marked collapsed node) to an R-node in A' at random from $N \cup N'$, where N and N' are the set of unmatched (*i.e.* not yet in the maximal matching in A') right marked neighbors of m and m' respectively. This can be achieved by a two step procedure: either choose m with probability $|N|/(|N'| + |N|)$, or choose m' with probability $|N'|/(|N| + |N'|)$. If m is chosen, then m requests to a node in N uniformly at random and m' keeps silent; else if m' is chosen, then m' requests to a node in N' uniformly at random and m keeps silent. However, as m and m' can have common right neighbors and also can be adjacent to both nodes of a matched pair, we can show that the probability that any of the right marked neighbors of v is sent a request for matching is at least $1/(4|N \cup N'|)$ (instead of $1/(|N \cup N'|)$). However, the proof of the maximal matching still goes through with this modification and hence the lemma follows.

C. Distributed Algorithm for Computing Maximal Matching

This algorithm [11], [26] involves $C \log n$ repetitions of the following steps for some constant $C > 0$:

- (i) Each node randomly chooses to be left(L) or right(R).
- (ii) Every L-marked node sends a request to all its R-marked neighbors.
- (iii) R-marked nodes randomly grant one of all incoming requests to form a matched edge.
- (iv) All matched nodes remove themselves from the computation procedure. All unmatched nodes remove the matched nodes from their neighbor list.