Learning Contention Patterns and Adapting to Load/Topology Changes in a MAC Scheduling Algorithm

Yung Yi, Gustavo de Veciana, and Sanjay Shakkottai

***Abstract*— Aggregate traffic loads and topology in multi-hop wireless networks may vary slowly, permitting MAC protocols to 'learn' how to spatially coordinate and adapt contention patterns. Such an approach could reduce contention, leading to better throughput and energy consumption. To that end we propose a new family of distributed MAC scheduling algorithms combining synchronous two-level priority RTS/CTS handshaking with randomized time slot selection. We prove that for any fixed admissible load such algorithms converge to a feasible schedule (i.e., throughput-optimal). Furthermore, by adaptively biasing time-slot selection probabilities based on past history, one can develop variations that are also provably throughput-optimal and exhibit better convergence rates. Additionally under moderate loads local changes in load would lead to only local changes in contention patterns leading once again to fast convergence. This makes the case for adopting such protocols in wireless multi-hop networks, where aggregate loads and network topology are slowly varying.**

I. Introduction

The design of MAC protocols for wireless multi-hop networks has received much attention over the last decade. These protocols can be broadly classified into contention-based schemes (e.g., [1]) and scheduling-based schemes (see [2] for a survey). Contention-based schemes (e.g., IEEE 802.11 [1]) are based on random channel access with asynchronous data transmission. In the contention-based schemes, the throughput significantly degrades with increasing load due to collisions, but it enables more implementation simplicity. The scheduling-based schemes allocate channel resources (using either centralized or distributed strategies) in order to minimize resource contention. However, it requires complex control message exchange, leading to non-scalability to network dynamics (i.e., load or topology changes).

In this paper, we study a MAC scheduling algorithm, which leverages the advantages of both schemes[1]. To that end, we propose a *synchronous* contention-based MAC scheduling algorithm, which self-adapts to changes in traffic load and network topology, converging, if possible, to a conflict-free schedule by exchanging synchronous control messages, as data transmissions occur simultaneously (see Figure 1). Thus, our algorithm has reasonably high throughput even during transients, and automatically adapts to load changes without any explicit notification, by 'learning' *local* contention patterns. Further, our algorithm is a distributed one, which is amenable to simple implementation, where only four *one-hop* control message exchange (over each time-slot) is required.

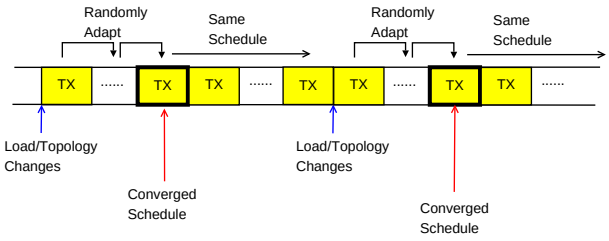


Fig. 1. Load/Topology Adaptive MAC Scheduling

Our research is motivated by the following factors:

(i) Slowly varying loads/topologies: Our premise in this paper is that although individual traffic loads may change quickly, the aggregates on congested nodes may, in many relevant applications, change more slowly. Similarly, node mobility leads to changes in topology (and thus changes in load), but again these changes might be slow enough to permit a MAC protocol to learn and exploit the offered traffic characteristics so as to quickly realize conflict-free schedules. A nice example is that of a wireless mesh network [3], where wireless mesh routers form a wireless multi-hop network, and wireless mesh routers receive/send aggregate traffic from/to wireless mesh clients.

(ii) Learning contention patterns: A scheduling-based MAC scheduling algorithm is known to achieve a high throughput after it finds a conflict-free schedule. However, whenever load or network topology changes, it has to re-initiate a "scheduling-decision" phase to find a conflict-free schedule. On the other hand, a contention-based protocol (e.g., IEEE 802.11) asynchronously transmits data, enabling easier implementation and better robustness to load or network topology changes. However, its throughput significantly degrades with increasing loads. Our goal in this paper is to leverage the advantages of both protocols, to realize high throughput and robust adaptability to load and/or network topology changes.

By using *synchronous* contention, we expect to learn contention patterns, such that time-slot allocation (chosen locally by nodes) can become close to an "efficient" schedule by progressively learning the past contention patterns. Synchronous

[1]We restrict our discussion to the case where shared resources are time-slots (i.e., a TDMA system). However, it is well-known in literature that resource allocation algorithms for a TDMA system immediately extend to FDMA or CDMA systems as long as the resource satisfies an orthogonality property (i.e., "non-overlapping" resources such as different time-slots, or orthogonal codes).

contention [4]–[6] is known to be a good strategy for efficient channel utilization, since it protects data transmissions as well as acknowledgments, leading to eliminating the need for maintaining protocol states (e.g., NAVs in IEEE 802.11 [1]), as compared to asynchronous schedules. Further, it provides a better framework to support priority access and better QoS [7]. In part, synchronous contention is crucial for throughput-optimality and enables us to break "deadlock" using a multi-level priority control messages (see Section III). The trade-off is the additional effort to synchronize nodes with equal time. However, this condition of tight synchronization can be relaxed with minor performance decrease [7], i.e., only "local" synchronization is needed among nearby neighbors..

(iii) Guaranteeing high throughput and fast convergence: If an adaptive MAC is to be useful, then high throughput and fast convergence (to a conflict-free schedule) should be guaranteed. Otherwise, most of the time will be devoted to searching for a conflict-free schedule with possibly low throughput. In particular, fast convergence is indispensable for tracking the time-varying loads and topologies. However, such algorithms to date [8]–[11] do not *provably* guarantee high throughput and fast convergence, or assume limited network topology (e.g., tree) as well as the restricted collision model. Thus, the challenge remains to devise an algorithm, which provably and quickly converges a conflict-free schedule for any feasible load, irrespective of network topology (i.e., throughput-optimal[2]).

In a typical time-slotted system, time is divided into transmissions slots (time-slots), which are grouped into frames. We consider a time-slotted system, where each link is subject to an offered traffic load, which is typically represented by the number of time-slots over a frame. Depending on the service supported by the network, information on the offered load could either be explicitly given to the nodes or be measured by the nodes. If we have a guaranteed-service network based on a resource reservation signaling (e.g., RSVP [12]), the amount of load could be known a priori by nodes in the path of a reserved flow. However, in a typical best-effort service network, the amount of load is not explicitly provided to the nodes, but the nodes could know the offered load by measuring/estimating it over a suitable time-period.

The problem of finding a conflict-free schedule in a time-slotted wireless multi-hop network, has been an active research topic by formulating the original problem into a graph-coloring problem . Since the graph-coloring problem is NP-complete, there exist a huge body of research proposing sub-optimal polynomial time heuristics (see [13] and references therein). Another approach is to propose distributed algorithms with provable (partial) throughput guarantees. Examples include distributed suboptimal schemes [14]–[17], which provide lower-bounds (typically from $1/8$ to $1/2$) on the throughput-region with no explicit knowledge of the offered load (i.e., statistics/knowledge of the load over any link in the topology), and require control message exchanges with $O(1)$ [17] or $O(\log^4 N)$ complexity [14], where N is the network size.

We note that all the above-mentioned strategies are for *static* scenarios, where the scheduling-decision phase and data transmission phase are separated. Thus, any network topology or load changes lead to a new scheduling-decision phase. Further, even when these are implemented as distributed algorithms, every node should be notified of the event of a change by a broadcasting of control messages. Most of research on this area assumes that such control messages are successfully transferred to nodes contention-free, which seems to be unrealistic in the resource-constrained wireless multi-hop networks. Our work differs from the above-mentioned work in that our algorithm adapts to load or topology changes without explicit notification of such changes.

An alternate approach is to devise a dynamic scheduling algorithm, where data transmission and scheduling-decision (time-slot allocation) occur simultaneously. Several dynamic algorithms have been proposed [8]–[11]. However, they require either two-hop connectivity information [8], or are not provably throughput-optimal [8]–[10]. The work in [11] is limited to networks with only "node-exclusive conflict model" (i.e., primary conflict), and is only shown to provably converge for a tree network topology. Our work also differs from these in that for *any* arbitrary network and feasible load, our algorithm converges to a conflict-free scheduling allocation (throughput-optimal), as proved in Section IV.

A. Main Contributions and Organization

The main contributions of this paper are as follows:

(i) We propose a synchronous contention-based load/topology-adaptive link scheduling algorithm (DCAMA: Dynamic Contention-Aware Multiple Access). In the DCAMA algorithm, two-level RTS/CTS synchronous control message are used together with randomized time-slot selection at each links. We prove that the DCAMA always converges to a conflict-free schedule (if there exists one) for any feasible load and any arbitrary network topology (i.e., throughput-optimal), and study its rate of convergence.

(ii) The importance of DCAMA algorithm is that it could act as a base-line algorithm, whose variations are also provably throughput-optimal and their rate of convergence is also exponential. Thus, we propose an adaptive variation to the DCAMA algorithm (ADCAMA: Adaptive DCAMA), which adaptively biases slot selection probabilities based on contention histories of the previous m frames. We prove that the ADCAMA algorithm also converges to a conflict-free schedule, and by simulation we show that only a three-frame history is necessary to significantly improve the rate of convergence and the transient throughput, resulting in a good adaptation to load/topology changes.

The paper is organized as follows. We begin with a description of the system model in Section II. Next, in Section III,

[2]A link scheduling scheme is said to be *throughput-optimal* if it can find a conflict-free schedule for any feasible offered load. An offered load is *feasible* if there exists a conflict-free schedule (see Section II for formal definitions).

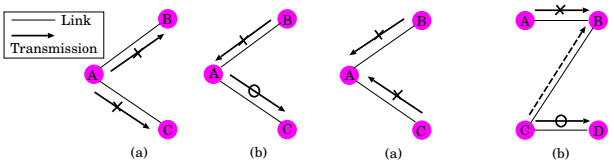


Fig. 2. (a) Unicasting and (b) Half-Duplex.

Fig. 3. (a) Primary Conflict and (b) Secondary Conflict.

we describe the DCAMA algorithm and its properties. In Section IV, we prove that the DCAMA converges to a conflict-free schedule for any feasible load, irrespective of network topology. We then propose an adaptive heuristic (adaptive DCAMA) to improve the convergence rate (Section V). Finally, in Section VI, we validate the results using simulations.

II. System Model

We model the wireless multi-hop network by a graph $\mathcal{G}(\mathcal{L}, \mathcal{V})$, where $\mathcal{L} = \{1, \cdots, |\mathcal{L}|\}$ denotes a set of directional links, and $\mathcal{V} = \{1, \cdots, |\mathcal{V}|\}$ denotes a set of nodes. We assume that for any link between two nodes there is a counter-part in the opposite direction. The wireless system has a *single* frequency/code, which is available for both data and control message transmission, and there is no separate physical channel for control messages (i.e., in-band signaling). Each node in the system is equipped with an omni-directional antenna, and is synchronized. We assume that each transmission is intended for only one receiver (unicasting constraint), and each node has only a single transceiver (half-duplex radio) (see Figure 2).

In our network model, if node $i \in \mathcal{V}$ is within the transmission range of $j \in \mathcal{V}$, then the link from i to j is established (denoted by $i \rightarrow j$). Thus, we have two transmission conflict scenarios: (i) primary conflicts, where either multiple nodes transmit simultaneously to the same receiver (Figure 3(a)), and (ii) secondary conflict, where a node receiving transmission is also within the transmission range of other transmissions not intended for it (Figure 3(b)). Further, due to a single transceiver, packet reception and transmission are not allowed to happen simultaneously (Figure 2(b)). Finally, the transmission is intended only for one receiver (Figure 2(a)). The access problem arises due to the above-mentioned four resource constraints between links[3].

In a TDMA based wireless ad-hoc network, time is divided into transmission slots (*time-slots*), which are grouped into *frames*. A time-slot duration is suitably chosen to accommodate the transmission of one fixed-size packet and includes a guard time corresponding to the maximum differential propagation delay between pairs of nodes in the network. We assume that the frame size in the network is fixed throughout system operation, where the frame size is chosen (heuristically) depending on the number of nodes, network load, and quality-of-service constraints.

[3]In a typical distributed link scheduling algorithm, a node is responsible for determining slot-schedules for its outgoing links. Thus, the unicasting constraint in Figure 2(a) is automatically resolved.

We further assume that a node can distinguish between the absence of any transmission and packet collisions (e.g., carrier sensing). For example, in Figure 3(a), when B and C are transmitting messages to A simultaneously in a same time-slot, A is unable to decode the message due to collision, but A is able to know that there was transmissions sent to itself.

We do not consider routing and transport-layer end-to-end flows in this study. We focus on "next neighbor transmissions" since multiple access problems depends solely on the next neighbor transmission requirements.

With this setup, we denote the *offered-load* on the network by $\vec{\rho} = (\rho_l : l = 1, \cdots, |\mathcal{L}|)$, where ρ_l is the number of the requested time-slots over the link l in a frame, i.e., $\vec{\rho} \in \mathcal{Z}_+^{|\mathcal{L}|}$, where $\mathcal{Z}_+$ is the set of non-negative integers.

The scheduling decision at each frame is represented by a *contention matrix* (CM), $C(F, \vec{\rho}) = (c_{ls} : l = 1, \cdots, |\mathcal{L}|, s = 1, \cdots, F)$ for a frame-size F and an offered load $\vec{\rho}$, where $c_{ls} = 1$ implies that a transmission is scheduled to contend over the link l on time slot s. For all $l \in \mathcal{L}$, $\rho_l = \sum_{s=1}^{F} c_{ls}$, i.e., the number of contending time-slots is equal to the load offered on that link.

Further, we use $\vec{c}_{l\cdot} = (c_{ls} : s = 1, \cdots, F)$ and $\vec{c}_{\cdot s} = (c_{ls} : l = 1, \cdots, |\mathcal{L}|)$ to refer to the l-th row and s-th column vector of C, respectively. We call $\vec{c}_{l\cdot}$ and $\vec{c}_{\cdot s}$ a *slot schedule* over l and a *link schedule* on time-slot s, respectively. The link l is said to be *satisfied* by $\vec{c}_{l\cdot}$, if all its scheduled transmissions by $\vec{c}_{l\cdot}$ are successful.

Definition 2.1: A contention matrix $C(F, \vec{\rho})$ is said to be feasible if all its links are satisfied. An offered load $\vec{\rho}$ is said to be feasible over a frame size F if there exists a feasible $C(\vec{\rho}, F)$.

Our primal goal is to devise a *distributed* algorithm, which converges to a feasible schedule (i.e., after it reaches a feasible schedule, it stays at that schedule over successive frames, before any change in traffic loads or network topology) for *any* feasible offered load and *any* network topology (i.e., provably throughput-optimal) under the following constraints: (i) only a small number of one-hop control message is permitted between the transmitter and the receiver at a link, and there does not exist a separate contention-free control channel (ii) data transmission and the process to find a feasible schedule are not separated.

III. DCAMA Algorithm

A. Overview

The frame and time-slot structure of the DCAMA algorithm are shown in Figure 4. A time-slot is divided into two parts: time to exchange control messages and time to transmit data to the receiver. We describe the DCAMA algorithm by dividing its behavior into two different time-scales: (i) per-frame operation (Section III-B.1), where at start of each frame, a node determines the slot-schedules for the transmissions over its adjacent outgoing links, following the offered loads, and (ii) per-slot operation (Section III-B.2), where a node initiates control message signaling to resolve contentions and transmit

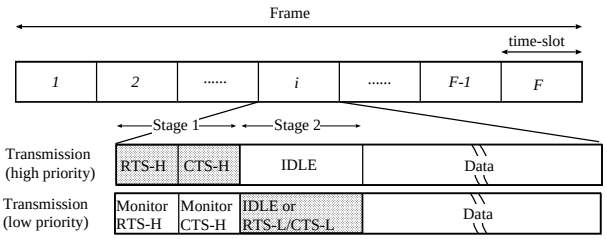


Fig. 4. Frame and Slot Structure

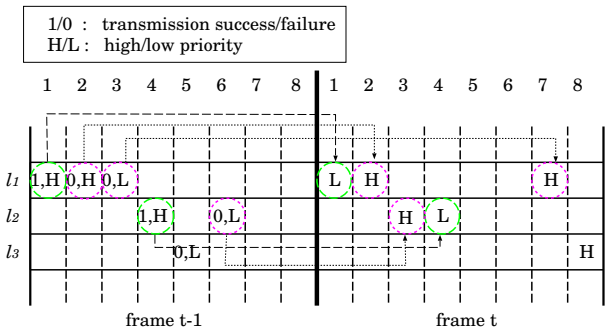


Fig. 5. Example of Determining Slot-Schedules

data if this time-slot is scheduled by slot-schedules over one of its outgoing link.

To resolve contention, we use a *synchronous* RTS/CTS based mechanism. However, unlike conventional contention based algorithms our contention resolution mechanism has *two-level priority: high and low,* i.e., every scheduled transmission on a slot is assigned one of low or high priority[4]. In other words, control message exchange is decomposed into two stages, where the first and the second stages are used for high and low priority transmissions, respectively. In particular, the transmitters and the receivers of low priority transmissions monitor control message signaling *at the first stage*, and determines whether it has to defer (i.e., release this time-slot) or contend on this time-slot (see Section III-B.2 for details). An issue with two-level RTS/CTS signaling is that of additional control messages overhead (as compared to the conventional RTS/CTS signaling in the absence of priority). However, as well shall see later in Section VI-B, our algorithm does not generate significant additional overheads. Further, due to synchronous contention, we do not require information fields for maintaining states needed by asynchronous protocols (e.g., NAV and DIFS in IEEE 802.11 [1]). In Section VI-B, we will quantitatively compute the additional overheads due to two-level RTS/CTS signaling, and show that the performance increase (about 25%) is much higher than the additional signaling overhead (about 3%).

The key mechanisms to achieve a goal to converge to a feasible schedule (for any feasible load and for any network topology) are summarized as follows:

Two level RTS/CTS priority. With a contention mechanism without priority, even for a feasible load, the algorithm could reach a deadlock, and thus it can be trapped in a "bad" schedule, forever (see Figure 6 for an example). The two-level priority RTS/CTS mechanism ensures that such a deadlock does not arise.

Synchronized contention. Synchronous contention enables receivers to infer the presence of RTS/CTS transmissions merely by sensing signaling activity over the appropriate time-intervals in a frame (corresponding to the RTS/CTS transmission "slots" within a frame). Note that this does not imply that these messages are successfully decoded by the receiver. Synchronization is useful in conjunction with the two-level priority signaling scheme, as it enables low priority transmissions to release a slot even if a signaling message collision occurs.

Randomized slot selection. The algorithm is randomized in determining slot-schedules (at the next frame) for unsuccessful transmissions. Incorporation of prioritized RTS/CTS mechanism with randomized slot selection strategy enables the system to reach any schedule, and thus to ultimately converge to a feasible schedule.

We note that a similar idea of using multiple priorities has been used in the TDMA scheduling used in Z-MAC [18], where a node can be in one of two modes: low contention level (LCL) or high contention level (HCL). However, the approach in Z-MAC differs from ours in that *(i)* it considers only the graph-based interference model, and (ii) the major objective of multiple priorities is to solve the hidden terminal problem without any provable throughput-guarantee, whereas we use two-level priority to get both provable convergence and throughput-guarantee.

We additionally introduce a control message priority matrix, $R = (r_{ls} : l \in \mathcal{L},\ s \in F)$ to represent the control message priority, where $r_{ls} = 1$ ($r_{ls} = 0$) if a transmission is scheduled over link l on time-slot s (i.e., $c_{ls} = 1$) and its priority is high (low), and NULL if $c_{ls} = 0$.

B. Algorithm Description

1) Determining Slot-Schedules: When each frame starts, a node (say, $v \in \mathcal{V}$) determines the slot-schedules and their RTS/CTS priorities for the transmissions over its adjacent outgoing links (denoted by $\mathcal{O}_v$). To do that, the following simple rules are used:

Rule 3.1 (Slot and Priority Selection Rule):

(i) Successful transmissions: The time-slots at which successful transmissions were realized at the previous frame are sustained with low control message priority at the current frame.

(ii) Unsuccessful transmissions: If more time-slots are required (i.e., for transmissions that were not successful at the previous frame), then they are selected at random among the remaining time-slots, with high control message priority.

[4]Throughout this paper, for notational simplicity, we use RTS-H/CTS-H and RTS-L/CTS-L to refer to control messages with high and low priority level, as needed.

To illustrate, consider the example in Figure 5, where $\mathcal{O}_v = \{l_1, l_2, l_3\}$ with $\rho_{l_1} = 3$, $\rho_{l_2} = 2$, $\rho_{l_3} = 1$, and the frame size is 8. Since at frame $t-1$, the transmission over l_1 on time-slot '1' was successful, this transmission is scheduled once again with *low* control message priority at the same time-slot positions at frame t. The same principle is applied to the transmission over l_2 on time-slot '4.' For the unsuccessful transmissions over l_1 on time-slots '2' and '3', we randomly choose two time-slots of the remaining time-slots, which was not "reserved" by the successful transmissions (i.e., v does not consider time-slots '1' and '4' in this random selection). In the example, time-slot '2' and '7' are selected, and they are scheduled with *high* control message priority from Rule 3.1(ii). The same rule is applied to other unsuccessful transmissions. Note that a slot where an unsuccessful transmission was realized at frame $t-1$ could be again scheduled at frame t (e.g., the transmission over l_1 on time-slot '2' at frame t).

2) Resolving Contentions: Following the determined slot-schedules, at each time-slot, nodes use the following two-stage RTS/CTS signaling mechanism to resolve contentions and transmit data. Only transmitters having successful RTS/CTS signaling with their receivers are allowed to transmit data.

Two-Stage RTS/CTS Signaling Mechanism
Stage 1: The transmitters and the receivers of high priority transmissions perform RTS-H/CTS-H signaling.
Stage 2: Depending on monitoring status at stage 1, the transmitters and the receivers of transmissions with low priority, which is not forced to release this time-slot by Rule 3.2, perform their RTS-L/CTS-L signaling.

Why is signaling in absence of priority inappropriate? First, we explain that signaling without priority could reach a deadlock condition (i.e., it could stay at an infeasible schedule forever, even if the offered load is feasible). Consider the example in Figure 6(a). At frame '0', the transmission over the link A→B on time-slot '1' is unsuccessful due to a collision of RTS messages from A and C at node B, whereas the transmission over link C→E is successful. At frame '1', as we discussed in Section III-B.1, successful transmissions will be sustained on the same time-slot. Note that any choice of either time-slot '1' or '2' over the link A→B results in unsuccessful transmission due to once again an RTS collision at node B. Thus, even if the offered load is feasible (thus, there exists a feasible schedule), an incorrect choice of initial schedule leads to a deadlock condition.

How does two priority level signaling help? However, if there are two priority levels for control signaling, we can avoid such deadlocks. The reason why we have a deadlock condition with signaling in absence of priority is that there exists a deterministic "winner-loser" relationship between links, such that if winners maintain their time-slots, no time-slots are available for the transmission by losers. For example, in Figure 6, the transmission over the link C→E always wins over link A→B, when both of them are scheduled on the same time-slot, if we use RTS/CTS signaling without priority.

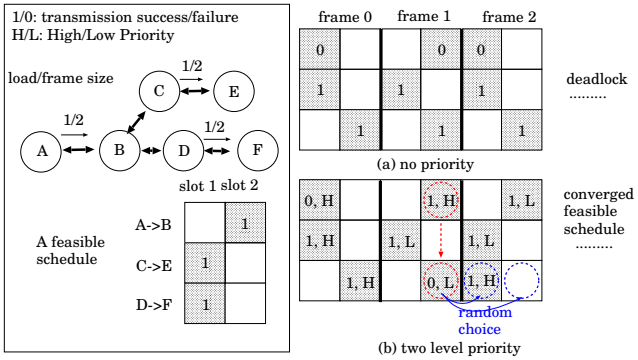


Fig. 6. RTS/CTS Signaling with and without Priority

To avoid this deadlock situation, link scheduling algorithms must have a mechanism, whereby there are no deterministic winner-loser relationships. In the DCAMA algorithm, this is achieved by a two priority level of RTS/CTS mechanism, i.e., a scheduled transmission with high priority at some link could "beat" a transmission with low priority at some other link (if those two transmissions have contention relationship as shown in Figures 2 and 3). The two level priority scheme enables an unsuccessful transmission to preempt a successful one by contending for the channel with high priority signaling. At the same time previously successful transmissions must contend using low priority RTS-L/CTS-L allowing, if need be, possible release of time-slots. This intuition is realized in Stage 2 of two-stage RTS/CTS signaling mechanism, where low priority transmission releases its time-slot (i.e., defers its transmission) by monitoring high priority signaling messages at Stage 1 and applying Time-Slot Release Rule, which will be explained next.

Time-slot release rule. We use $s(l)$ and $d(l)$ to refer to the source and destination of a link l, respectively. We say that a node *senses* a control message if it decodes a control message or receives non-decodable packet collision. Recall that in Section II, we assumed that a node can distinguish between the absence of any transmission and packet collisions. We say that a (low priority) transmission over link l *releases* a time-slot s if $s(l)$ or $d(l)$ does not perform RTS/CTS signaling, but the transmission is scheduled on slot s. A low priority transmission decides on its time-slot release (for conflicting high priority transmissions) by conforming to the following simple rule:

Rule 3.2 (Time-slot release rule): A low priority transmission on a given slot s over link l releases the slot s, if on slot s, (i) $s(l)$ senses CTS-H, (ii) $d(l)$ senses RTS-H, (iii) $s(l)$ transmits CTS-H, or (iv) $d(l)$ transmits RTS-H.

By applying Rule 3.2 to low priority transmissions (which will be active at Stage 2), we can easily show that if an high priority transmission has conflicts with a low priority transmission, and both of them are scheduled on the time-slot s, then the high priority transmission makes the low priority transmission release the slot s (see Figures 7(a)-(e) for the simple examples). We use "senses" (not "decodes")

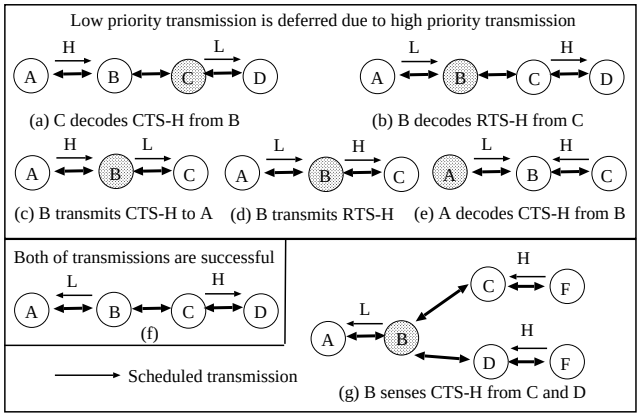


Fig. 7. Examples for Synchronous Two-Level Priority

in Rules 3.2(i) and (ii), as there are some cases when the low priority transmission has to release, even if the high priority signaling message is not decodable (see Figure 7(g) for an example). Rules 3.2(iii) and (iv) comes from half-duplex device constraint (see Figures 7(c) and (d)).

Note that the destination of a low priority transmission ($d(l)$) is oblivious to its identity as a receiver before it receives and decodes the corresponding RTS-L message intended for itself. Thus, Rule 3.2 (ii) seems to be non-sense. However, if the corresponding RTS-L message is not correctly received (due to packet collisions among low priority transmissions) at Stage 2, $d(l)$ will not send CTS-L message, leading to automatic slot-release.

Further, we reiterate that one of major advantage of *synchronous contention* in Rules 3.2(i) and (ii) is that a node is able to identify the kind of control message, irrespective of its decodability, which helps low priority transmission decide on its time-slot release.

IV. Convergence Results

In this section, we prove that for any feasible offered load, the DCAMA algorithm converges to a feasible schedule, and its rate of convergence is represented by a geometric random variable. Throughout this section, we implicitly assume that the given offered load is feasible and is denoted by $\vec{\rho}$, and the frame size is F.

Prior to describing the main theorem, we first define a "distance" function between two contention matrices, where distance represents how many different slot-schedules they have between two contention matrices.

Definition 4.1: With a same network topology, a load, and a frame size, consider two contention matrices, $C = (c_{ls})$ and $B = (b_{ls})$. We define

$$D(C,B) \quad = \quad \sum_{l=1}^{|\mathcal{L}|} \rho_l - \sum_{l=1}^{|\mathcal{L}|} \sum_{s=1}^{F} c_{ls} \times b_{ls}.$$

Intuitively, $D(C,B)$ corresponds to the number of scheduled transmissions by C, each of which is not scheduled by B.

It can be easily shown that if $D(C,B) = 0$, then two contention matrices B and C are equivalent. Thus, for a feasible contention matrix $C^\star$, the fact that $D(C,C^\star) = 0$ implies that C is also feasible.

Theorem 4.1 (Convergence): For an arbitrary graph $\mathcal{G}(\mathcal{L},\mathcal{V})$ with a feasible load $\vec{\rho}$ over the frame-size F, the DCAMA algorithm converges to a feasible contention matrix (i.e., throughput-optimal).

We first represent the system status at the frame t via $(C[t].R[t])$. We say that a control message priority matrix, $R = (r_{ls})$, is said to *low* if all the scheduled transmissions have low control message priority, i.e., $r_{ls} = 0$ whenever $r_{ls} \neq NULL, \forall s \in \{1,\ldots,F\}, \forall l \in \mathcal{L}$. It can be easily seen that $\{(C[t],R[t]),\ t \geq 0\}$ forms a Markov chain with at least one absorbing state, where an absorbing state corresponds to (C',R') for some feasible C', *low* R'. Note that any state $(C[t],R[t])$, where $C[t]$ is feasible and $R[t]$ is *not* low, goes to an absorbing state over one frame with probability '1' (i.e., $C[t+1],R[t+1]$ will become an absorbing state with probability '1') since $C[t]$'s feasibility ensures that all scheduled transmissions will be successful at frame $t+1$, and all their priorities will be low. Thus, to prove the main theorem, it suffices to show that there is a positive probability that from any initial state, we reach a feasible contention matrix within a finite time.

Our strategy to prove the theorem is that for any fixed feasible contention matrix $C^\star$, we will show that over (at most) two frames there is a positive probability that we get "closer" to $C^\star$ (i.e., $D(C[t+2],C^\star) = D(C[t],C^\star) - 1$), or $C[t+2]$ equals to some other feasible contention matrix $C^{\star\star}, C^{\star\star} \neq C^\star$ (as the feasible contention matrix is not necessarily unique). Since $D(C,C^\star)$ is upper-bounded by $\sum_{l=1}^{|\mathcal{L}|} \rho_l$, for any initial contention matrix C, strict decrease over two frames suffices to prove the convergence. In the proof, we will construct a converging path to a $C^\star$. The proof is presented in [19].

Now, we study the rate of convergence (i.e., time to converge to a feasible contention matrix). We first define a random variable $\tau(C)$, corresponding to a convergence time to a feasible contention matrix for a given initial contention matrix C. Then, we have the following exponential rate of convergence.

Theorem 4.2 (Rate of Convergence): For any initial contention matrix C, $\forall t \in \mathcal{Z}_+$, we have

$$\Pr\Big\{\tau(C) > tK\Big\} \quad \leq \quad p^t,$$

for some constants $0 < K < \infty$, and $0 < p < 1$.

The proof is presented in [19].

V. Adaptive DCAMA

Note that the DCAMA algorithm chooses a new time-slot (for an unsuccessful transmission) with equal probability in the subsequent frame. In fact, one can potentially increase the rate of convergence or adapt to load change more effectively by intelligently guessing which time-slot is likely to be successful

(using the past history), and biasing the time-slot access probabilities. As shown in Proposition 5.1 below, such variations of the DCAMA algorithm inherits the property of convergence and the rate of convergence.

In this section, we propose a general family of variations of DCAMA algorithm, the ADCAMA (Adaptive DCAMA) family, which adaptively assigns different time-slot access probabilities, depending on the past contention history, i.e., more efficient learning of local contention patterns. To that end, each link is assigned its own slot weight vector, and the individual nodes maintain slot weight vectors for its adjacent outgoing links. This slot weight vector is updated every frame by the associated node, depending on the transmission results (success or failure) at the past frames, or overhearing signaling messages around it. Let us denote the slot weight vector of link l at frame t by $\vec{w}^l[t] = (w_s^l[t] : \ s = 1. \cdots, F,)$.

To increase/decrease the slot weight vector based on the past contention histories, we define the *time-slot status*, which corresponds to the result of past contentions (e.g., success or failure) on the corresponding time-slots. Then, the slot access probability is set to be *inversely proportional* to the current weight. Also, by setting the minimum and maximum of weight, we can avoid pathological cases (e.g., the time-slot access probability could be arbitrarily small or close to '1'), i.e., there exist $\bar{w}$ and $\underline{w}$, such that $1 \leq \underline{w} < \bar{w} < \infty$ and $\forall s \in \{1, 2, \ldots, F\}, \forall l \in \mathcal{L}$, and $\forall t > 0$, $\underline{w} \leq w_s^l[t] \leq \bar{w}$.

Then, we define a m-frame history based ADCAMA algorithm, where each node stores and uses the previous m-frame slot status history, based on which slot weight vector is updated at every frame. Intuition behind the multi-frame history based algorithm is that we could potentially increase the rate of convergence or have the higher transient throughput by considering longer slot usage history. As an example, a time-slot with consecutive success is highly likely to be "safe", so that it would be beneficial to sustain the corresponding time-slot at the next frame[5]. In Section VI, we will show that even with a simple weight maintenance algorithm based on three frame contention history, we could have quite a performance increase, compared with the DCAMA algorithm.

Now, we have the following proposition to Theorem 4.1:

Proposition 5.1 (Convergence of ADCAMA): For an arbitrary graph $\mathcal{G}(\mathcal{L}, \mathcal{V})$ with a feasible load $\vec{\rho}$ over the frame-size F, any m-frame history based ADCAMA algorithm converges to a feasible contention matrix (i.e., throughput-optimal).

The proof is presented in [19].

VI. Simulation Results

In this section, we simulate wireless multi-hop networks with nodes which are randomly distributed in a 500 $\times$ 500 or 100 $\times$ 100 meter-square area. The number of nodes, their transmission range, and the frame size are parameterized, such that we can observe the performance of the proposed algorithms under different connectivity densities, time varying environments and MAC layer rate granularities.

[5]Thus, DCAMA algorithm corresponds an algorithm belonging to the ADCAMA family. However, we use the term 'DCAMA' to refer to an ADCAMA algorithm without frame history

TABLE I
Parameters Used for Weight Increase/Decrease

$S_s^l[t-3]$	$S_s^l[t-2]$	$S_s^l[t-1]$	inc/dec Weight
SUCC	SUCC	SUCC	$-D_1$
FAIL/IDLE	SUCC	SUCC	$-D_2$
FAIL	FAIL	FAIL	$+I_1$
SUCC/IDLE	FAIL	FAIL	$+I_2$

A. Weight Maintenance Algorithm

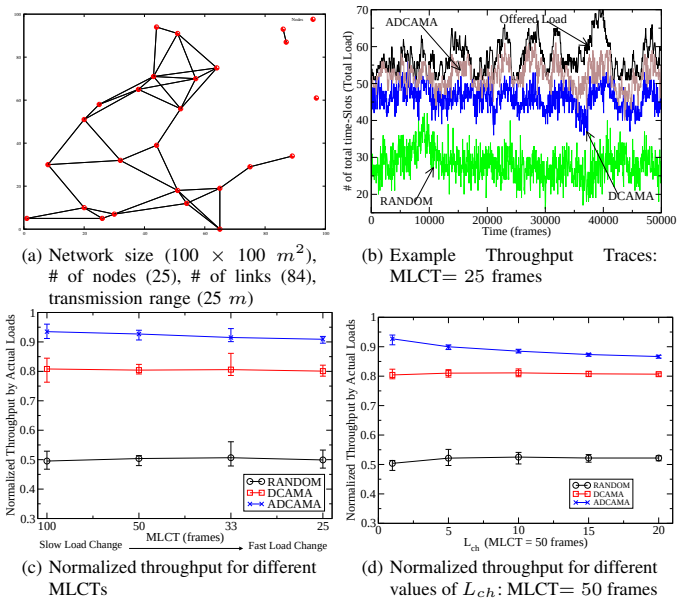
In Section V, we have proposed a family of DCAMA variations (ADCAMA) adaptively assigning different time-slot access probabilities. We now describe the details of a simple weight maintenance strategy based on three-frame history. To summarize our strategy, we increase/decrease slot weights (equivalently, the slot access probabilities, see Section V) based on observed success/failure of past time-slot requests. We show that even with a simple weight update mechanism, the performance of DCAMA can be improved significantly, and enables it to be more adaptive to load/topology changes.

We denote a slot status over link l at time-slot s at frame t by $S_s^l[t]$. We have three kinds of time-slot status: success (SUCC), failure (FAIL), and idle (IDLE). The IDLE status corresponds to the case when a node which did not sense any control message.

Table I shows the parameters used in the simulation for a typical link l. The parameters I_i and D_i are the (additive) weight increase/decrease constants used by nodes to adapt their slot weights based on past observations. Table I summarizes the observed state over the past three frames, and the corresponding weight change operation. These parameters are chosen such that $D_1 > D_2 > 0$, and $I_2 > I_1 > 0$. We have used $D_1 = I_1 = 3$, $D_2 = I_2 = 1$, in all simulation results, where the maximum and minimum weights (i.e., $\bar{w}$ and $\underline{w}$) are set to 30 and 1, respectively (recall that the time-slot access probabilities are inversely proportional to weights).

The intuition for these choices is that more back-to-back successes at a time slot indicate that the offered loads around the corresponding node at that time-slot are relatively low (i.e., less "congested"), and transmissions in that time-slot are likely to be successful in the future. Similar intuition is applied for back-to-back failures. However, empirical evidence based on simulations have indicated that responding to just a one-time success/failure by decreasing/increasing the weight was not very helpful, because such a success/failure could have happened due to transient movement of transmission schedules at other conflicting links (i.e., it does not capture congestion very well).

With regard to the IDLE status, it seems intuitive to schedule an unsuccessful transmission at the IDLE time-slot with higher probability (i.e., decrease the weights) in order to to "spread" the offered load over all the time-slots of a frame.



(a) Network size (100 × 100 m^2), # of nodes (25), # of links (84), transmission range (25 m)

(b) Example Throughput Traces: MLCT= 25 frames

(c) Normalized throughput for different MLCTs

(d) Normalized throughput for different values of L_{ch}: MLCT= 50 frames

Fig. 8. With frame size of 10 and the network topology (a), (b) shows an example traces of # of time-slots with successful transmissions, compared to the actual loads. (c) and (d) show the normalized throughput w.r.t the actual loads over 50000 frames for different values of MLCTs and L_{ch}.

However, weight decrease at the IDLE status could generate a synchronization effect, i.e., due to aggressive decrease (by multiple nodes), multiple transmissions are highly likely to be scheduled at this slot, leading to collision again. Based on empirical evidence using simulations, responding aggressively to SUCC and FAIL is the determining factor in providing fast convergence and good adaptability. We finally comment that using other numerical values for D_j, I_j based on the heuristics above also results in significant performance improvements (compared to DCAMA), thus indicating that these heuristics are quite robust to the actual numerical values. We do not present simulations for varying D_j, I_j due to space constraints.

B. Simulation Results

In this section, we evaluate the performance of DCAMA and ADCAMA algorithm by comparing them to the RANDOM algorithm, described below. The RANDOM algorithm determines slot-schedules (based on the requested loads) in a pure-random manner at each frame, and uses a single-level RTS/CTS signaling to gain access to the channel. The reason why we adopt the RANDOM algorithm as a baseline comparison is because *(i)* it is similar to Aloha-like strategy (which is a "standard" algorithm for TDMA link scheduling), and behaves like a slotted version of a CSMA-like contention-based scheme, and *(ii)* it is not clear how we can compare with some of the other dynamic coloring based algorithms, since their objective is to solve a variant of the coloring problem with different system models (such as two-hop control message exchange and different transmission conflict scenarios, see Section I).

Prior to presenting simulation results, we comment on the control overhead of the DCAMA/ADCAMA algorithm. Our approach has additional overheads as compared to a standard contention based MAC protocol (which has only one RTS/CTS signaling phase). Suppose that a MAC packet has 1000 bytes of data (note that in the 802.11 MAC, the size limit is 2312 bytes). The overhead of each RTS/CTS message pair with DCAMA is no more than 30 bytes (6 bytes each for source/destination addresses, and 3 bytes for signaling such as RTS priority level, stage, etc) will suffice for our protocol. Thus, the additional overhead (recall that the "standard" protocol has only one stage of RTS/CTS messages) is about 30 bytes, which corresponds to approximately 3%.

On the other-hand, we have significant throughput gains when compared to a baseline random access MAC, and our simulations indicate a 25-30% gain in various scenarios (changing topology, load requirements. steady-state, etc). Thus, it seems worthwhile to pay the penalty of additional overheads in order to accrue this additional throughput gain. Another overhead with a fixed slot-size based approach is due to partial-slot wastage, i.e., a small packet in one time slot wastes part of a time-slot and thus reduces time resource usage. However, this could be overcome by using "packet bursting" or "packet aggregation" [20], where a time-slot usage is maximized by aggregating the packets intelligently.

There are additional issues in comparing the DCAMA/ADCAMA algorithm with conventional "static" TDMA algorithms (please see Section I for the related work). In a static TDMA algorithm, with every load/topology change, the scheduling decision has to re-computed, for which control messages has to be exchanged, and most of the research in literature assumes that the control messages are successfully transfered to neighboring nodes contention-free. However, in a single channel wireless ad-hoc network, this assumption seems to be unrealistic. Thus, it may take some time to disseminate and share the newly generated scheduling decision. On the other hand, our approach does not make any such assumptions, and indeed RTS/CTS collisions could occur, leading to control message losses.

Adaptation to load changes. First, we investigate the effect of load changes on the performance of DCAMA and ADCAMA algorithm with frame size of 10 in the network topology of Figure 8(a). We generate time-varying loads by a random walk model, where we first determine a normalized offered load of 70% (by a randomly chosen maximally feasible load[6]). Then, at the beginning of each frame we randomly choose L_{ch} links and increase their link loads by one slot with probability P_L^I, decrease their link loads with probability P_L^D, or stay at the current load (i.e., no change) with probability $1 - P_L^I - P_L^D$. For simplicity, in the simulation, we set $P_L \triangleq P_L^I = P_L^D$. Thus, higher values of P_L corresponds to a faster load change with time. Then, the mean load change time (MLCT) over L_{ch} links is $1/(2 \times P_L)$ frames.

Figure 8(c) shows that the throughput (over 50000 frames)

[6]A load is said to be *maximally feasible* if the resulting system load becomes infeasible with *any* load increase anywhere in the network.

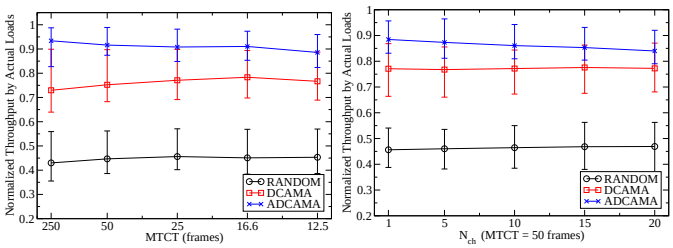
normalized by the actual (time-varying) offered load for different values of MLCTs ($L_{ch} = 1$) varying from 25 to 100 frames, where the error bars represent the maximum and minimum values of 10 simulations with different random seed values (i.e., different load changing patterns). For a network with a link capacity of 10 Mbps, and a frame-size of 10 (which corresponds to a 10 msec frame duration), this corresponds to a load change ranging from once every 250 msec to once every 1 seconds.

We observe that with ADCAMA algorithm, the normalized throughput is above 90%, whereas the RANDOM achieves about 50%. Figure 8(b) shows an example trace of throughput (i.e., number of successful transmission slots) for MLCT= 25 frames, where we observe that ADCAMA algorithm tracks the actual load very well, resulting in nice adaptation to time-varying load changes.

Figure 8(d) shows the normalized throughput by actual offered loads in faster load changing scenario, where with MLCT= 50 frames, L_{ch} varies from 1 to 20. Note that the actual mean load change time for $L_{ch} = 20$ is $50/20 = 2.5$ frames, which corresponds to 25 msec. As L_{ch} becomes larger, the throughput difference between DCAMA and ADCAMA becomes slightly smaller. This is because with faster changing loads, ADCAMA algorithm does not have sufficient time to completely adapt to changes. However, even in this fast changing regime, ADCAMA shows a 5% throughput improvement over DCAMA.

Adaptation to topology change. Second, we investigate the effect of topology changes on the performance of DCAMA and ADCAMA algorithm. With time-varying topology changes, new links and existing links are dynamically added and deleted in the network, which possibly changes the offered loads in the links. To simulate practical scenarios of such load/topology changes, we use end-to-end flows to generate offered loads in the network. For a fixed frame size, we first randomly generate a network topology. To initialize the end-to-end flows, we employ the following procedure: we randomly choose two nodes (source and destination of an end-to-end connection), and increase the load in the path of the chosen source-destination end-to-end connection. We assume that a shortest path routing is used to determine the path. We carry out this random selection of end-to-end connections $2*|\mathcal{V}|$ times (note that $|\mathcal{V}|$ is the number of nodes in the network), and skip the chosen end-to-end connection if the load increase of its path generates infeasible offered load. Thus, the initially generated loads are very close to a maximally feasible load.

For the evolving time-varying topology, we randomly choose N_{ch} nodes and move each of the nodes independently in one of four directions (north, south, east, or west) by one meter with probabilities of P_T^N, P_T^S, P_T^E, or P_T^W. In the simulations, we set them to be all equal, denoted by P_T. Then, the mean topology change time (MTCT) is $1/(4\times P_T)$ frames. If the topology changes due to node movement, we establish the new end-to-end routes based on the shortest path routing, for initially found end-to-end connections.

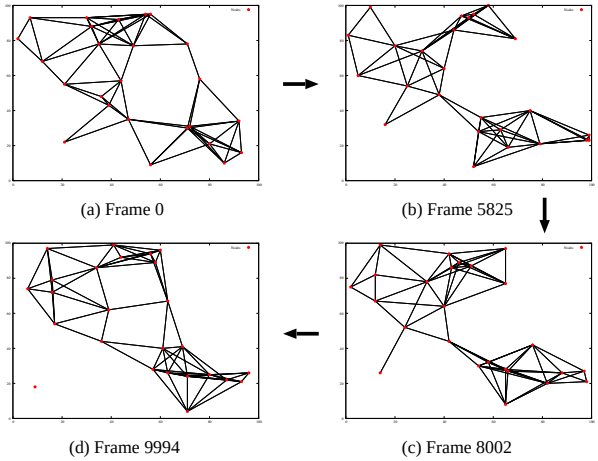


(a) Normalized throughput for different MTCTs

(b) Normalized throughput for different values of N_{ch}: MTCT= 50 frames

Fig. 9. With 100×100 m^2 network size, 30 nodes, 25m transmission range, and frame size of 10, we first uniformly place nodes in the plane, and generate end-to-end connections randomly. (a) and (b) show the normalized throughput w.r.t the actual loads for different values of MTCTs and N_{ch}.

Figure 9(a) shows that the throughput (over 10000 frames) normalized by the actual (time-varying) offered load (due to topology changes) for different values of MTCTs varying from 12.5 frames to 250 frames. Again, for a network with a link capacity of 10 Mbps with the frame size of 10, this corresponds to a change ranging from once every 125 msec to once every 2.5 seconds, which corresponds to a mobile terminal moving with an average velocity of 8 m/sec and 0.4 m/sec. This seems reasonable for a typical slowly varying environment. Similar to the figures in load change simulations, we represent the maximum and the minimum values of error bars in 10 simulations with different random seed values, leading to different initial network topologies and end-to-end connections.



(a) Frame 0

(b) Frame 5825

(d) Frame 9994

(c) Frame 8002

Fig. 10. This figure shows an instance of topology changes (due to the random-walk model) at four time instances.

Further, Figure 9(b) shows the normalized throughput when with MTCT= 25 frames, N_{ch} varies from 1 to 20 nodes. We again observe that with ADCAMA algorithm, the normalized throughput is above 90%, whereas RANDOM achieves at most 60%. Thus, ADCAMA algorithm exhibits good robustness to topology changes.

VII. Discussion and Concluding Remarks

In this paper, we have studied the problem of dynamic MAC scheduling for a time-slotted wireless networks with arbitrary topologies. We have developed a synchronous contention-based MAC algorithm (DCAMA) that provably converges to a conflict-free schedule for any feasible load and for arbitrary topologies. The key mechanisms that enables us to achieve this are a synchronous two-level priority RTS/CTS based contention scheme and randomized selection of time-slots. Based on this algorithm, we have proposed heuristics (which also provably converge) that improve the convergence time by biasing time-slot access probabilities based on past contention history.

An issue that we do not directly address in this paper is the behavior of the algorithms when the load is not feasible. To handle such cases, we will need to combine admission control strategies (long time-scale control) along with the MAC algorithms (short time-scale resource allocation) to ensure that a feasible solution exists. In any case, simulations (not presented in this paper due to space constraints) indicate that even in an overloaded scenario (with no admission control), the DCAMA and ADCAMA algorithms provide much larger "transient" network throughput (about 20 – 30% increase) when compared to the baseline RANDOM algorithm (however, neither DCAMA or ADCAMA converge because a feasible solution does not exist). Future work will focus on developing and studying admission control policies in conjunction with the (A)DCAMA algorithm.

References

[1] IEEE Standard 802.11, "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," 1997.

[2] V. S. R. S. Kumar and J. Deng, "Medium access control protocols for ad-hoc wireless networks: A survey," *Elsevier Ad-Hoc Networks Journal*, 2005, to appear.

[3] I. F. Akyildiz, X. Wang, and W. Wang, "Wireless mesh networks: A survey," *Computer Networks Journal (Elsevier)*, vol. 47, no. 4, 2005.

[4] T. J. Shepard, "A channel access scheme for large dense packet radio networks," in *Prooceeding of SIGCOMM*, 1996.

[5] R. Rozovsky and P. R. Kumar, "Seedex: a mac protocol for ad hoc networks," in *Proceeding of MobiHoc*, 2001.

[6] X. Yang and G. de Veciana, "Inducing spatial clustering in mac contention for spread spectrum ad hoc networks," in *Proceedings of MobiHoc*, 2005.

[7] J. Stine and G. de Veciana, "A paradigm for quality-of-service in wireless ad hoc networks using synchronous signaling and node states," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 7, pp. 1301–1321, September 2004.

[8] C. Zhu and M. S. Corson, "A five-phase reservation protocol (FPRP) for mobile ad hoc networks," *Wireless Networks*, vol. 7, no. 4, pp. 371–384, 2001.

[9] Z. Tang and J. J. Garcia-Luna-Aceves, "A protocol for topology-dependent transmission scheduling in wireless networks," in *Proceedings of WCNC*, 1999.

[10] L. Bao and J. J. Garcia-Luna-Aceves, "Distributed dynamic channel access scheduling for ad hoc networks," *Journal of Parallel Distributed Computing*, vol. 63, no. 1, pp. 3–14, 2003.

[11] T. Salonidis and L. Tassiulas, "Distributed dynamic scheduling for end-to-end rate guarantees in wireless ad hoc networks," in *Proceedings of MOBIHOC*, 2005.

[12] P. While, "RSVP and integrated services in the internet: A tutorial," *IEEE Communications Magazine*, May 1997.

[13] S. Ramanathan, "A unified framework and algorithm for channel assignment in wireless networks," *Wireless Networks*, vol. 5, no. 2, pp. 81–94, 1999.

[14] P. Chaporkar, K. Kar, and S. Sarkar, "Throughput guarantees through maximal scheduling in wireless networks," in *Proceedings of the 43rd Annual Allerton Conference on Communication, Control and Computing*, 2005.

[15] L. X. Bui, A. Eryilmaz, R. Srikant, and X. Wu, "Joint asynchronous congestion control and distributed scheduling for multi-hop wireless networks," in *Proceedings of INFOCOM*, 2006.

[16] X. Lin and N. B. Shroff, "The impact of imperfect scheduling on cross-layer rate control in wireless networks," in *Proceedings of INFOCOM*, 2005.

[17] X. Lin and S. Rasool, "Constant-time distributed scheduling policies for ad hoc wireless networks," Purdue University, Tech. Rep., 2006.

[18] I. Rhee, A. Warrier, M. Aia, and J. Min, "Z-mac: a hybrid mac for wireless sensor networks," in *Proceedings of the ACM conference on Embedded networked sensor systems (SenSys)*, 2005.

[19] Y. Yi, G. de Veciana, and S. Shakkottai, "Lattice-throughput-optimal MAC scheduling: Adapting to load and topology changes by learning neighborhood contention," University of Texas at Austin, Tech. Rep., 2006.

[20] B. Sadeghi, V. Kanodia, A. Sabharwal, and E. Knightly, "Opportunistic media sccess for multirate ad hoc networks," in *Proceedings of ACM MobiCom*, 2002.