The Abstract MAC Layer*

Fabian Kuhn, Nancy Lynch, and Calvin Newport

MIT CSAIL, Cambridge, MA
{fkhun,lynch,cnewport}@csail.mit.edu

Abstract. A diversity of possible communication assumptions complicates the study of algorithms and lower bounds for radio networks. We address this problem by defining an Abstract MAC Layer. This service provides reliable local broadcast communication, with timing guarantees stated in terms of a collection of abstract *delay functions* applied to the relevant contention. Algorithm designers can analyze their algorithms in terms of these functions, independently of specific channel behavior. Concrete implementations of the Abstract MAC Layer over basic radio network models generate concrete definitions for these delay functions, automatically adapting bounds proven for the abstract service to bounds for the specific radio network under consideration. To illustrate this approach, we use the Abstract MAC Layer to study the new problem of *Multi-Message Broadcast*, a generalization of standard single-message broadcast, in which any number of messages arrive at any processes at any times. We present and analyze two algorithms for Multi-Message Broadcast in static networks: a simple greedy algorithm and one that uses regional leaders. We then indicate how these results can be extended to mobile networks.

1 Introduction

The study of bounds for mobile ad hoc networks is complicated by the numerous possible communication assumptions: Do devices operate in slots or asynchronously? Do simultaneous transmissions cause collisions? Can collisions be detected? Is message reception determined by geographical distances? Or is it determined by a more complex criteria, such as signal-to-noise ratio? And so on. This situation causes problems. Results for one set of communication assumptions might prove invalid for a slightly different set. In addition, these low-level assumptions require algorithm designers to grapple with low-level problems such as contention management, again and again, making it difficult to highlight interesting high-level algorithmic issues. This paper proposes a possible solution to these concerns. (A technical report with more details is also available [19].)

* This work has been support in part by Cisco-Lehman CUNY A New MAC-Layer Paradigm for Mobile Ad-Hoc Networks, AFOSR Award Number FA9550-08-1-0159, NSF Award Number CCF-0726514, and NSF Award Number CNS-0715397.

The Abstract MAC Layer. We introduce an *abstract MAC layer* service for mobile ad hoc networks (MANETs). We intend this service to be implemented over real MANETs, with very high probability. At the same time, we intend it to be simple enough to serve as a good basis for theoretical work on high-level algorithms in this setting. The use of this service allows algorithm designers to avoid tackling issues as contention management and collision detection. They can instead summarize their effects with abstract delay bounds.

The abstract MAC layer service delivers transmitted messages reliably within its local neighborhood, and provides feedback to the sender of a message in the form of an acknowledgement that the message has been successfully delivered to all nearby receivers. The service does not provide the sender with any feedback about particular recipients of the message. The service provides guaranteed upper bounds on the worst-case amount of time for a message to be delivered to all its recipients, and on the total amount of time until the sender receives its acknowledgement. It also may provide a (presumably smaller) bound on the amount of time for a receiver to receive *some message* among those currently being transmitted by neighboring senders. These time guarantees are expressed using *delay functions* applied to the current amount of contention among senders that are in the neighborhoods of the receivers and the sender.

To implement our abstract MAC layer over a physical network one could use popular contention-management mechanisms such as carrier sensing, backoff, receiver-side collision detection with NACKs, or perhaps even network coding methods, such as the ZigZag Decoding approach of Gollakota and Katabi [11]. Our MAC layer encapsulates the details of these mechanisms within the service implementation, presenting the algorithm designer with a simple abstract model that involves just message delivery guarantees and time bounds.[1] We believe that this MAC layer service provides a simple yet realistic basis for theoretical work on high-level algorithms and lower bounds for MANETs.

Multi-Message Broadcast and Regional Leader Election. In this paper, we validate our formalism by studying two problems: *Multi-Message Broadcast (MMB)* and *Regional Leader Election (RLE)*. The MMB problem is a generalization of single-message broadcast; c.f., [1–4, 6, 5, 7, 8, 16, 14, 15, 17, 18]. In the MMB problem, an arbitrary number of messages originate at arbitrary processes in the network, at arbitrary times; the problem is to deliver all messages to all processes. We present and analyze two MMB algorithms in static networks, and indicate how the second of these can be extended to mobile networks.

Our first MMB algorithm is a simple greedy algorithm, inspired by the strategy of the single-message broadcast algorithm of Bar-Yehuda et al. [3]. We analyze this algorithm using the abstract MAC layer delay functions. We obtain an upper bound on the time for delivery of each message that depends in an

[1] Note that MAC layer implementations are usually probabilistic, both because assumptions about the physical layer are usually regarded as probabilistic, and because many MAC layer implementations involve random choices. Thus, these implementations implement our MAC layer with very high probability, not absolute certainty.

interesting way on the *progress bound*—the small bound on the time for a receiver to receive *some* message. Specifically, the bound for MMB to broadcast a given message m, is of the form $O((D+k)F_{prog} + (k-1)F_{ack})$, where D is the network diameter, k is a bound on the number of messages whose broadcast overlaps m, and F_{ack} and F_{prog} are upper bounds on the acknowledgement and progress delay functions, respectively. Note that a dependency on a progress bound was implicit in the analysis of the single-message broadcast algorithm in [3]. Our use of the abstract MAC layer allows us to make this dependency explicit.

Our second MMB algorithm achieves better time complexity by exploiting geographical information; in particular, it uses a solution to the RLE problem as a sub-protocol. In the RLE problem, the geographical area in which the network resides is partitioned statically into regions; the problem is to elect and maintain a leader in each occupied region. Regional leaders could be used to form a backbone network that could, in turn, be used to solve many kinds of communication and coordination problems. We give an RLE algorithm whose complexity is approximately bF_{prog}, where b is the number of bits required to represent process ids.

Using the RLE algorithm, our second MMB algorithm works as follows: After establishing regional leaders, the MMB algorithm runs a version of the basic greedy MMB algorithm, but using just the leaders. In order to transfer messages that arrive at non-leader processes to leaders, all the processes run a *collect* sub-protocol in parallel with the main broadcast algorithm. The complexity of the resulting MMB algorithm reduces to $O(D + k + bF_{prog} + F_{ack})$, a significant improvement over MMB without the use of leaders.

Finally, to extend our second MMB algorithm to the mobile case, we provide a preliminary theorem that says that the MMB problem is solved given certain restrictions on mobility and message arrival rates.

Contributions. The contributions of this paper are: (a) the definition of the abstract MAC layer, and the suggestions for using it as an abstract layer for writing mobile network algorithms, and; (b) new algorithms for Multi-Message Broadcast and Regional Leader Election, and their analysis using the abstract MAC Layer.

2 Model

We model a Mobile Ad Hoc Network (MANET) using the Timed I/O Automata (TIOA) formalism. Our model captures n user processes, which we label with $\{1, ..., n\}$, in a mobile wireless network with only local broadcast communication.

2.1 System Components

Our system model consists of three component automata, the *network automaton*, the *abstract MAC layer automaton*, and the *user automaton*, connected as

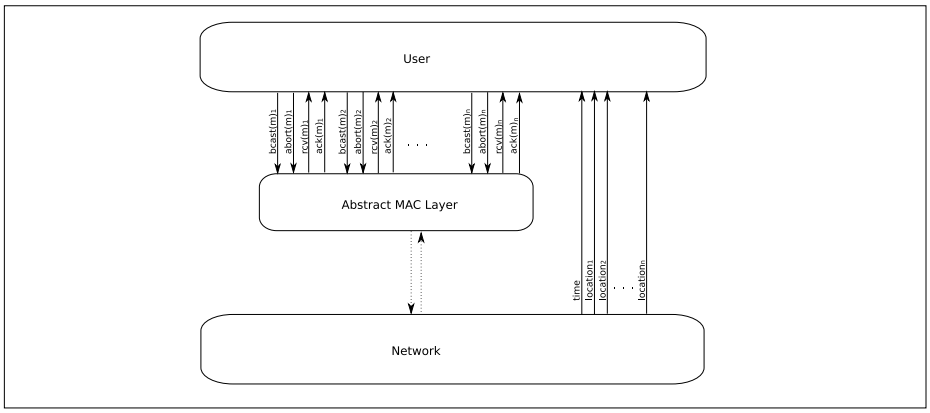

Fig. 1. The MANET system.

shown in Figure 1. The *network automaton* models the relevant properties of the physical world: time, mobile node locations, and physical network behavior. It provides a physical layer interface for low-level communication on the radio channel. It outputs the time and mobile node locations; we assume here that this information is accurate. The network automaton comes equipped with a pair of functions f_G and $f_{G'}$ that map from states to directed graphs whose vertices V are the mobile nodes. The graph $G = (V, E) = f_G(s)$ is the *communication graph* in state s, indicating the processes that are within communication range in s. The graph $G' = (V, E') = f_{G'}(s)$ is the *interference graph* in state s, indicating the processes within interference range. We consider communication separately from interference because in many practical radio network models the interference range exceeds the reliable communication range.[2]

The *abstract MAC layer automaton* mediates the communication of messages between the user processes and the network. Each user process i interacts with the MAC layer automaton via MAC layer inputs $bcast(m)_i$ and $abort(m)_i$ and MAC layer outputs $rcv(m)_i$ and $ack(m)_i$, where m is a message from some message alphabet. (The *abort* is used in cases where the sender is satisfied that "enough" neighbors have already received the message, and so is willing to terminate efforts by the MAC layer to continuing broadcasting.) The abstract MAC layer automaton connects to the network through the physical layer interface. Finally, the *user automaton* models n user processes, numbered $1, \ldots, n$. Each process i connects to the MAC layer through the (*bcast*, *abort*, *rcv*, *ack*) interface described above, and might also receive the network's location and time outputs.

[2] To capture some physical layer models, notably a Signal to Interference-plus-Noise Ratio model, we might need to extend our definition of G' to allow weights on the edges; that is, capture not just who might interfere but also how much interference they contribute. We do not make this extension here but leave it as interesting future work.

2.2 Guarantees for the Abstract MAC Layer

We assume that the user automaton guarantees some basic well-formedness properties of system executions, namely, that each execution is *user-well-formed* in the sense that: (a) it contains at most one *bcast* event for each message m (all messages are unique); (b) No process i performs more than one $abort(m)_i$ for any message m, and performs an $abort(m)_i$ only after a $bcast(m)_i$ but not after an $ack(m)_i$; and (c) No process submits a *bcast* until after its previous *bcast* (if any) ended with an *abort* or *ack*.

The composition of an abstract MAC layer and network automaton, which we call a *MAC layer*, must ensure the constraints described below, for any user-well-formed execution α. To begin, we assume a *cause* function that assigns to every $rcv(m)_j$ event in α a preceding $bcast(m)_i$ event, where $i \neq j$, and that assigns to each $ack(m)_i$ and $abort(m)_i$ a preceding $bcast(m)_i$. This function must satisfy:

1. **Receive correctness:** Suppose that $bcast(m)_i$ event π causes $rcv(m)_j$ event π' in α. Then: (a) *Proximity:* At some point between events π and π', $(i, j) \in E'$ (notice, we use the edge set from the interference graph, E', instead of the edge set from the communication graph, E, because the former captures edges where communication *might* occur, while the latter captures edges where communication *is guaranteed* to occur); (b) *No duplicate receives:* No other $rcv(m)_j$ event caused by π precedes π'; and (c) *No receives after acknowledgements:* No $ack(m)_i$ event caused by π precedes π';
2. **Acknowledgment correctness:** Suppose that $bcast(m)_i$ event π causes $ack(m)_i$ event π' in α. Then: (a) *Guaranteed communication:* If for every point between events π and π', $(i, j) \in E$ (the edge set of the communication graph), then a $rcv(m)_j$ event caused by π precedes π'; (b) *No duplicate acknowledgements:* No other $ack(m)_i$ event caused by π precedes π'; and (c) *No acknowledgements after aborts:* No $abort(m)_i$ caused by π precedes π';
3. **Termination:** Every $bcast(m)_i$ causes either an $ack(m)_i$ or an $abort(m)_i$.

We also impose upper bounds on the time from a $bcast(m)_i$ event to its corresponding $ack(m)_i$ and $rcv(m)_j$ events. These bounds are expressed in terms of the contention involving i and j during the broadcast interval. Let f_{rcv}, f_{ack}, and f_{prog} be monotonically non-decreasing functions from natural numbers to nonnegative reals. We use these to bound the delay for a specific message to be delivered, for an acknowledgement to be received, and for *some* message among many to be received, all with respect to a given amount of contention. For many MAC layer implementations, f_{prog} is smaller than f_{ack}, because the time to deliver some message is smaller than the time to deliver a specific message. Let ϵ_a be a small constant, used to bound the amount of time beyond an *abort* when the message could still be received somewhere.

We define a "message instance" to be a matched pair of $bcast_i$ and ack_i, or $bcast_i$ and $abort_i$ events. Let α be an execution, α' a closed execution fragment within α and j a process. Then $contend(\alpha, \alpha', j)$ is the set of message instances in α that intersect with fragment α', and such that $(i, j) \in E'$ at some point in

this intersection, where i is the sender from the instance in question. These are the message instances that might reach j during α'. Similarly, $\mathit{connect}(\alpha, \alpha', j) \subseteq \mathit{contend}(\alpha, \alpha', j)$ is the set of message instances such that α' is entirely contained between the corresponding bcast_i and ack_i events and $(i, j) \in E$ for the duration of α', where i is the sender. These are the messages instances that must reach j if α' is long enough. For an execution α and events π and π', $\alpha[\pi, \pi']$ denotes the execution fragment within α that spans from π to π'.

We can now formalize our time bounds with the *receive*, *acknowledgment*, and *progress* properties. These bound the time for a specific message to be received, a specific message to be acknowledged at the sender, and some message from among many to be received, respectively.

5. **Receive:** Suppose that a $\mathit{bcast}(m)_i$ event π causes a $\mathit{rcv}(m)_j$ event π' in α. Then the time between π and π' is at most $f_{rcv}(c)$, where c is the number of distinct senders of message instances in $\mathit{contend}(\alpha, \alpha[\pi, \pi'], j)$. Thus, the bound for j's receipt of m grows with the number of *nearby* processes (incoming neighbors, according to G') that have message instances intersecting with the instance in question. Also, if π causes an $\mathit{abort}(m)_i$ event π'', then π' occurs at most ϵ_a time after π''.
6. **Acknowledgement:** Suppose that a $\mathit{bcast}(m)_i$ event π causes an $\mathit{ack}(m)_i$ event π' in α. Let *ackcon* be the set containing i and every process j such that there exists a $\mathit{rcv}(m)_j$ with cause π. Then the time between π and π' is at most $f_{ack}(c)$, where c is the number of distinct senders of message instances in $\bigcup_{j \in ackcon} \mathit{contend}(\alpha, \alpha[\pi, \pi'], j)$. This bound is similar to the receive bound, except that we now consider the contention at the sender and at all receivers. This is intended to allow enough time for the receivers to somehow communicate their receipt of the message back to the sender.
7. **Progress:** For every closed fragment α' within α, for every process j, and for every integer $c \geq 1$, it is not the case that *all* three of the following conditions hold:

   (a) The total time described by α' is strictly greater than $f_{prog}(c)$; (b) The number of distinct senders of message instances in $\mathit{contend}(\alpha, \alpha', j)$ is at most c, and $\mathit{connect}(\alpha, \alpha', j)$ is non-empty; and (c) No $\mathit{rcv}(m)_j$ event from a message instance in $\mathit{contend}(\alpha, \alpha', j)$ occurs by the end of α'.

   Thus, the time bound for j to receive *some* message (when at least one message is being sent by an incoming neighbor in G), grows with the total number of processes that are in interference range.

Fixed Bounds on Message Delivery. In some results, we will use constant upper bounds F_{rcv}, F_{ack}, and F_{prog} on f_{rcv}, f_{ack}, and f_{prog}, respectively, all defined with respect to a particular execution α. These upper bounds take the maximum values of the functions over all graphs that occur in α and all possible amounts of contention, as defined by the node degrees that occur in those graphs. In the design of algorithms, we sometimes use F^+_{rcv}, F^+_{ack}, F^+_{prog}, which are defined with respect to all *executions* of a given network automaton.

2.3 Implementing an Abstract MAC Layer

It is beyond the scope of this paper to offer a detailed implementation of an abstract MAC layer automaton. Here we discuss, only informally, some basic ideas for implementations with the aim of providing some intuition regarding the type of concrete definitions our delay functions might adopt in practice. We consider the simple case where G and G' are the same for all network states (that is, the network is static) and undirected, and $G = G'$. (See [12] for an example of how a scheme could be adapted to tolerate mobility and transient faults.) We assume a physical network that corresponds to the slotted radio broadcast model of [2, 3, 10, 20, 22, 13]. This model assumes that communication occurs in synchronized slots, and that a message from a sender i is correctly received by a neighbor j in a time slot s if and only if i is the only neighbor of j broadcasting during s. The model includes no collision detection—a collision cannot be distinguished from silence.

In this setting, a simple *Decay* strategy [2, 3] can be used to implement the abstract MAC layer. In this approach, time is divided into synchronized epochs of $\Theta(\log \Delta)$ time slots, where Δ is the maximum degree in G. A process with a message to broadcast starts broadcasting at the beginning of the next epoch. During an epoch, a sending process decreases its probability of broadcasting exponentially, from 1 to $1/\Delta$. It is guaranteed that every process with at least one neighbor sending a message during an epoch receives at least one message, with constant probability. Thus, the progress delay function f_{prog} is $O(\log \Delta)$ (with high probability). The receive and acknowledgement delay functions, f_{rcv} and f_{ack}, are both $O(\Delta \log \Delta)$.

2.4 Multiple Abstract MAC Layer Automata

To simplify the analysis of multiple user protocols running on the same physical network, it is sometimes useful to include several independent abstract MAC automata in the same system. In this scheme, each protocol connects with its own MAC automaton, all of which connect with the same network automaton. Each MAC automaton satisfies the specifications given above, with respect to the common network. This approach allows an algorithm designer to prove properties of the behavior of the individual protocols and assert that they still hold when the protocols are combined, thus evading issues of contention among the protocols. Note that there are practical realizations of multiple MAC automata. For example, most radio-equipped computing devices have access to many communication frequencies. If a device has several transmitters, it can execute several simultaneous MAC protocols on independent frequencies. If the device has a single transceiver and/or access to only a single frequency, it can use a Time-Division Multiplexing scheme to partition use of the frequency among the logical MAC layers.

3 Multi-Message Broadcast

The Multi-Message Broadcast (MMB) problem assumes that the environment submits messages to the user processes at arbitrary times during an execution. The goal is to propagate every such message to *all* of the users in the network. In this section we assume a *static* network, that is all states generate the same G and G' graphs. Furthermore, we assume $G = G'$ and the graphs are undirected. We use the notation $D(G)$ to refer to the diameter of graph G.

An *MMB protocol* for a message alphabet $\mathcal{M}$ is a user automaton whose external interface includes an $arrive(m)_i$ input and $deliver(m)_i$ output for each user process i and message m. We say that an execution of an MMB protocol is *MMB-well-formed* if it contains at most one $arrive(m)_i$ event for each m. (Each broadcast message is unique). An MMB protocol *solves the MMB problem* if, for every MMB-well-formed execution: (a) For every $arrive(m)_i$ and every process j, there is a $deliver(m)_j$; and (b) For every m and j, there is at most one $deliver(m)_j$, and it comes after some $arrive(m)_i$.

Our first MMB algorithm is a simple greedy algorithm, inspired by the single-message broadcast algorithm of Bar-Yehuda et al. [2, 3].

> **The Basic Multi-Message Broadcast (BMMB) Protocol**
> Every process i maintains a FIFO queue named *bcastq* and a set named *rcvd*. Both are initially empty. If process i is not currently broadcasting a message (i.e., not waiting for an *ack* from the MAC layer) and *bcastq* is not empty, it broadcasts the message at the head of the queue. If i receives an $arrive(m)_i$ event it immediately performs a $deliver(m)_i$ output and adds m to the back of *bcastq*. It also adds m to *rcvd*. If i receives a broadcast message m from the MAC layer it first checks *rcvd*. If $m \in rcvd$ it discards it. Else, i immediately performs a $deliver(m)_i$ event, and adds m to the back of *bcastq* and to the *rcvd* set.

Theorem 1. *The BMMB protocol solves the MMB problem.*

The proof is presented in the full version of this paper [19]. We continue with a collection of definitions used by our complexity proof. In the following, let α be some MMB-well-formed execution of the BMMB protocol composed with a MAC layer.

The get Event. We define a $get(m)_i$ event with respect to α, for some arbitrary message m and process i, to be one in which process i first learns about message m. Specifically, $get(m)_i$ is the first $arrive(m)_i$ event if message m arrives at process i, otherwise, $get(m)_i$ is the first $rcv(m)_i$ event.

The clear Event. Let $m \in \mathcal{M}$ be a message for which an $arrive(m)_i$ event occurs in α. We define $clear(m)$ to describe the final $ack(m)_j$ event in α for any process j.[3]

[3] Notice, by the definition of BMMB if an $arrive(m)_i$ occurs then i eventually broadcasts m, so $ack(m)_i$ occurs. Furthermore, by the definition of BMMB, there can be at most one $ack(m)_j$ event for every process j. Therefore, $clear(m)$ is well-defined.

The Set $K(m)$. Let $m \in \mathcal{M}$ be a message such that $arrive(m)_i$ occurs in α for some i. We define $K(m) = \{m' \in \mathcal{M} : \text{an } arrive(m') \text{ event precedes the last } deliver(m) \text{ event and the } clear(m') \text{ event follows the } arrive(m)_i \text{ event}\}$. That is, $K(m)$ is the set of messages whose processing overlaps the interval between the the $arrive(m)_i$ event and the last $deliver(m)$ event.

The obvious complexity bound would guarantee the delivery of a given message m in $O(D(G)kF_{ack})$ time, for $k = |K(m)|$, as there can be no more than k messages ahead of m at each hop, and each message is guaranteed to be sent, received, and acknowledged within F_{ack} time. The complexity theorem below does better. By separating kF_{ack} from the diameter, $D(G)$, instead multiplying by the smaller progress bound, F_{prog}. This captures an implicit pipelining effect that says *some* message always makes progress in F_{prog} time.

Theorem 2. *Let k be a positive integer and α be an MMB-well-formed execution of the BMMB protocol composed with a MAC layer. Assume that an $arrive(m)_i$ event occurs in α. If $|K(m)| \leq k$ then the time between the $arrive(m)_i$ and the last $deliver(m)_j$ is at most:*
$(D(G) + 2k - 2)F_{prog} + (k-1)F_{ack}$.

Theorem 2 is a direct consequence of the following lemma.

Lemma 1. *Let α be an MMB-well-formed execution of the BMMB protocol composed with a MAC layer. Assume that at time t_0, $arrive(m)_{i_0}$ occurs in α for some message $m \in \mathcal{M}$ and some process i_0. Let j be a process at distance $d = d_G(i_0, j)$ from the process i_0. Further, let $\mathcal{M}' \subseteq \mathcal{M}$ be the set of messages m' for which $arrive(m)_{i_0}$ precedes $clear(m')$. For integers $\ell \geq 1$, we define*

$$t_{d,\ell} := t_0 + (d + 2\ell - 2) \cdot F_{prog} + (\ell - 1) \cdot F_{ack}.$$

For all integers $\ell \geq 1$, at least one of the following two statements is true:

(1) The $get(m)_j$ event occurs by time $t_{d,\ell}$ and $ack(m)_j$ occurs by time $t_{d,\ell} + F_{ack}$.
(2) There exists a set $\mathcal{M}'' \subseteq \mathcal{M}'$, $|\mathcal{M}''| = \ell$, such that, for every $m' \in \mathcal{M}''$, $get(m')_j$ occurs by time $t_{d,\ell}$, and $ack(m')_j$ occurs by time $t_{d,\ell} + F_{ack}$.

Proofs of Lemma 1 and Theorem 2 are presented in the full version of the paper [19].

4 Regionalized Networks

Our general model specifies that the network automaton reports node locations, but does not constrain the geography of these locations or their relationship to G and G'. Here we define such constraints; we use these to study the leader election and optimized MMB protocols in Sections 5 and 6, respectively.

Fix L, a set of *locations* (e.g., points in the plane), R, a set of *regions ids*, and reg, a *region mapping* that maps locations to region ids. Let $N_R \subseteq N'_R$ be two symmetric neighbor relations among regions in R. We call the graph $G_{region} =$

(R, N_R) a *region communication graph* and the graph $G'_{region} = (R, N'_R)$ a *region interference graph*. We assume that G_{region} is connected and that the maximum node degree in G'_{region} is constant.

We define a physical network $\mathcal{N}$ to be *regionalized* (with respect to L, R, reg, N_R, and N'_R) provided that the following hold. $\mathcal{N}$ uses locations in L; in any particular state of $\mathcal{N}$, let $loc(i)$ denote the location of node i as encoded by $\mathcal{N}$. Then at any point in any execution of $\mathcal{N}$: (a) If $reg(loc(i)) = reg(loc(j))$ or $(reg(loc(i)), reg(loc(j))) \in N_R$, then $(i, j) \in E$; and (b) If $(i, j) \in E'$, then either $reg(loc(i)) = reg(loc(j))$ or $(reg(loc(i)), reg(loc(j))) \in N'_R$. That is, if two nodes are in the same region or neighboring regions in the region communication graph G_{region}, then they must be connected in G, and if two nodes are connected in G' then they are in the same or neighboring regions in the region interference graph G'_{region}. Thus, G_{region} describes which regions must be in communication range while G'_{region} describes which regions might be in interference range.

Fixing a Regionalized Network. For Sections 5 and 6 we fix a static network $\mathcal{N}$ that is regionalized with respect to some parameters L, R, reg, N_R, and N'_R. As in Section 3 we assume that $G = G'$ and the graphs are undirected. We also assume that the network occupies every region in every execution. When we refer to MAC layers in these sections, we implicitly mean MAC layers that include $\mathcal{N}$. When we refer to any region r, we implicitly assume that $r \in R$.

5 Leader Election

The BMMB protocol does not take advantage of location information. In Section 6 we describe a new MMB algorithm, the *Regional Multi-Message Broadcast algorithm*, which leverages this information to achieve a better complexity bound. The Regional MMB algorithm uses a backbone of leaders—one per region of the regionalized network—that are each elected using a local leader election protocol. This leader backbone forms a connected dominating set (CDS), as studied, for example, in [23, 21, 20, 24, 9]. Our algorithm, however, is simpler than those in prior work, because we use location information and the abstract MAC layer masks contention.

An *Regional Leader Election (RLE) protocol* is a user automaton that has a $leader(r)_i$ and $notleader(r)_i$ output for every process i and every region r. Such a protocol *solves the RLE problem for region r by time t* if in every execution, by time t, exactly one process i in region r outputs $leader(r)_i$, and every other process j in region r outputs $notleader(r)_j$.

We begin by describing the Fast Regional Leader Election (FRLE) protocol whose complexity depends only on F^+_{prog} (which we typically assume to be much smaller than F^+_{ack}), and the size of the id space. In the following, let b be the number of bits needed to describe the id space, and let ϵ_b be a fixed small

constant. We use this latter value in both leader election protocols to add a small *buffer* after the time required to receive a message.[4]

The r-Fast Regional Leader Election (FRLE) Protocol
In the r-FRLE protocol for some region r, each process i in r behaves as follows. Let $\epsilon'_a = \epsilon_a + \epsilon_b$. Divide the time interval from 0 to $b(F^+_{prog} + \epsilon'_a)$ into b phases each of length $F_{prog} + \epsilon'_a$. We associate phase p with bit p of the id space. At the beginning of phase 1, process i broadcasts the phase number and its id if it has a 1 bit in location 1 of its id. Otherwise it does not broadcast. After F^+_{prog} time has elapsed in the phase, if i broadcast and has not yet received an *ack*, it submits an *abort*. At the end of the phase (i.e., ϵ'_a time after the potential *abort*), i processes its received messages. If i did not broadcast in this phase yet received at least one message, it outputs $notleader(r)_i$ and terminates the protocol. Otherwise, it continues with the next phase, which proceeds the same as before with respect to bit position 2. This continues until i terminates with a $notleader(r)_i$ output or finishes the last phase without terminating. In the latter case, i submits a $leader(r)_i$ output.

Theorem 3. *For any region r, the r-FRLE protocol solves the RLE problem for region r by time $b(F^+_{prog} + \epsilon_a + \epsilon_b)$.*

FRLE works correctly because it is impossible for all processes that are non-terminated at the beginning of a phase to submit $notleader(r)$ outputs at the end of the phase. Moreover, two or more processes cannot survive all b phases to become leaders, because their ids differ in at least one bit position. The formal correctness proof is presented in the full version of the paper [19].

We continue by describing the Complete Regional Leader Election (CRLE) Protocol, which elects a leader in *every* region. It uses FRLE within each region and a Time-Division Multiplexing (TDM) strategy to avoid interference among the FRLE instances. As before, let b be the bits needed to describe the id space. This protocol uses a *minimal-sized region TDMA schedule* T defined with respect to the region interference graph for the regionalized network.[5] (Notice, by the definition of regionalized, $|T| = O(1)$.)

The Complete Regional Leader Election (CRLE) Protocol
In the CRLE protocol each process i behaves as follows. We dedicate $b(F^+_{prog} + \epsilon'_a)$ time to each set in T. Process i does nothing until the start of the time dedicated to the single set in T that contains i. Process i runs the $reg(loc(i))$-FRLE protocol during the time interval dedicated to this set. It first adds, however, a fixed offset to the time input used by FRLE to transform the time at the beginning of the interval to evaluate to 0, as expected by FRLE.

Theorem 4. *The CRLE protocol solves RLE problem for* every *region by time* $\Theta\left(b \cdot (F^+_{prog} + \epsilon_a)\right)$

The proof is presented in the full version [19].

[4] This is required by a technicality of the TIOA definition that allows multiple events to occur at the same time.

[5] That is, T describes minimally-sized sequence of sets of region ids such that: (a) every region id shows up in exactly one set; (b) no set contains two region ids that are neighbors in the region interference graph.

6 Regional Multi-Message Broadcast

The *Regional MMB (RMMB)* protocol runs a version of the basic greedy MMB algorithm over a connected backbone of leaders elected by the CRLE protocol. To transfer messages that arrive at non-leader processes to leaders, the processes run a *Collect* protocol in parallel with the main broadcast algorithm. The complexity of RMMB is just $O\left(D + k + bF_{prog} + F_{ack}\right)$, a significant improvement over Basic MMB. The improvement arises because RMMB confines the propagation of messages to the low-degree backbone of leaders elected by CRLE.

The Regional Multi-Message Broadcast (RMMB) Protocol
The protocol uses three independent MAC automata (see Section 2.4), which we call the *Collect*, *Leader*, and *Broadcast* MAC automata. We use the *Leader MAC* to elect regional leaders using CRLE, the *Broadcast MAC* to run BMMB on the leader backbone once CRLE terminates, and the *Collect MAC* to transfer messages that arrive at non-leaders to the regional leaders. The Collect protocol runs concurrently with the CRLE and BMMB protocols. Before CRLE completes, all processes running Collect queue messages in case they are elected leader. Each process i in region r maintains a *broadcast* queue and an *arrive* queue, both initially empty. It also maintains a *leader* flag, initially $false$, and two sets, *delivered* and *rcvd*, both initially empty.
Leader Election: Starting at time 0, process i executes the CRLE leader election protocol, using the *Leader* MAC. At the end of the protocol, process i sets its *leader* flag to *true* if and only if it performed a $leader(r)_i$ output.
Collect: When an $arrive(m)_i$ or $rcv((m,r))_i$ event occurs, process i adds the message (m or (m,r)) to its *arrive* queue. When i's *arrive* queue is non-empty it does the following. If the element at the head of the queue is a single message m', process i removes m' from the *arrive* queue, outputs $deliver(m')_i$, adds m' to the *delivered* set and to the *broadcast* queue, and *propagates* m'. Then it moves on to the next element in the *arrive* queue. The propagate step depends on the value of the *leader* flag: If $leader = true$, then propagate is a *noop*. If $leader = false$ then i broadcasts (m', r) using the Collect MAC, and then waits for the corresponding $ack((m', r))_i$. If the element at the head of the *arrive* queue is (m', r), then i removes (m', r) from the queue, outputs $deliver(m')_i$, adds m' to the *delivered* set and to the *broadcast* queue. (It does not propagate in this case.)
Broadcast: Process i waits for the fixed amount of time required for the CRLE protocol to complete. If i has $leader = true$ at this point, then it executes the BMMB protocol using the Broadcast MAC, using the *broadcast* queue maintained by the Collect protocol, and using its *delivered* set *in addition* to the list *rcvd* used by BMMB to determine when to pass along a message. If i is not a leader, then for each m received from the Broadcast MAC, if m is not in the delivered set then it outputs $deliver(m)_i$ and adds m to the *delivered* set.

The proofs to the following theorems are presented in the full version of the paper [19].

Theorem 5. *The RMMB protocol solves the MMB problem.*

For the time complexity, the key observation is that RMMB executes on a backbone of leaders. So the contention on the broadcast MAC automaton is at most the maximum degree of G'_{region}, which is constant, reducing the F_{ack} and F_{prog} to constants. For the following theorem, we assume that the rate of *arrive* events at each process is $O(1/F_{ack})$, preventing any process from having more than a constant number of messages in its *arrive* queue at once. Let $K(m)$ and $D(G)$ be defined the same as in Section 3.

Theorem 6. *Let k be a positive integer and α be an MMB-well-formed execution of the RMMB protocol composed with three MAC automata and a network. Assume that an $arrive(m)_i$ event occurs in α. If $|K(m)| \leq k$ then the length of the interval between $arrive(m)_i$ and the last $deliver(m)_j$ is:*

$$O\left(max\{b(F^+_{prog} + \epsilon_a), F_{ack}\} + D(G) + k\right).$$

7 Adapting RMMB for Mobile Networks

In the full version of this paper [19] we describe mobile RMMB—a modification of RMMB for a mobile setting. In addition to the protocol description, we prove a preliminary theorem that establishes bounds on RMMB's message delivery, under certain mobility constraints. We reproduce the theorem below to provide intuition regarding the type of results that can be proved in a mobile setting. (The full details of the protocol, and the proof of theorem, are in [19].)

In the statement below, we assume each process maintains a *region exit bound* state variable. This variable contains a time value that is no later than the time when the process will next exit its current region. We assume that while a process remains within a region, this value does not change. We say a network is *T-stable*, for some nonnegative real T, if and only if every process calculates an exit bound at least T past the current time upon entering a new region, and for all regions and for all times there exists at least one process with an exit bound at least T past the current time. Finally, we use t_{CF} to describe the running time of CRLE and D to describe the maximum diameter of G in the mobile network.

We obtain the following theorem:

Theorem 7. *Let k be a positive integer, F^{max}_{ack} and t^{max}_{CF} be nonnegative reals, and $T = (D+1)2kF^{max}_{ack} + kF^{max}_{ack}$. If we restrict the rate of arrive events such that no more than k such events happen in any interval of length T, and consider only regionalized $(2kF^{max}_{ack} + max\{kF^{max}_{ack}, t^{max}_{CF}\})$-stable networks with $F_{ack} \leq F^{max}_{ack}$, and $t_{CF} \leq t^{max}_{CF}$, then the mobile RMMB protocol, executed with $kF^{max}_{ack} + t^{max}_{CF}$ passed as the parameter to the mobile leader election sub-protocol, solves the MMB problem.*

8 Conclusion

We presented the abstract MAC layer for MANETs. This service is intended to be implemented over real MANETs, with high probability. It abstracts the complexities of programming for this environment—including contention management and collision behavior—allowing the algorithm designer to focus on the issues unique to the problem being solved.

This approach generates many interesting open questions. For example, exploring how we can use the layer to implement basic primitives such as neighbor discovery and unicast communication, or complex protocols such as spanning trees and dominating sets. Extensions to the MMB problem, such as calculating throughput bounds and the cost of sender acks, are also important. Another direction is to improve the abstract MAC layer formalism itself. We might generalize the G and G' model to capture the effects of signal to interference-plus-noise ratios (SINR), or perhaps replace the deterministic delay functions with probability distributions over the different possible delays. This latter change would support more advanced analysis of the system's probabilistic behavior. Finally, it will prove useful to analyze specific MAC layer strategies for specific radio network models, providing concrete definitions for the delay functions.

Acknowledgements

We thank those who contributed comments and suggestions towards this project. In particular, we acknowledge Jennifer Welch and Seth Gilbert for their careful readings and helpful suggestions, and thank Rotem Oshman and Majid Khabbazian for their helpful discussions and comments.

References

1. N. Alon, A. Bar-Noy, N. Linial, and D. Peleg. On the complexity of radio communication. In *Proceedings of the ACM Symposium on Theory of Computing*, 1989.
2. R. Bar-Yehuda, O. Goldreich, and A. Itai. Efficient emulation of single-hop radio network with collision detection on multi-hop radio network with no collision detection. *Distributed Computing*, 5:67–71, 1991.
3. R. Bar-Yehuda, O. Goldreich, and A. Itai. On the time-complexity of broadcast in multi-hop radio networks: An exponential gap between determinism and randomization. *Journal of Computer and System Sciences*, 45(1):104–126, 1992.
4. I. Chlamtac and S. Kutten. On broadcasting in radio networks - problem analysis and protocol design. *IEEE Transactions on Communications*, 33(12):1240–1246, 1985.
5. B. S. Chlebus, L. Gasieniec, A. Gibbons, A. Pelc, and W. Rytter. Deterministic broadcasting in unknown radio networks. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, 2000.
6. B. S. Chlebus, L. Gasieniec, A. Gibbons, A. Pelc, and W. Rytter. Deterministic broadcasting in ad hoc radio networks. *Distributed Computing*, 15(1):27–38, 2002.

7. A. Clementi, A. Monti, and R. Silvestri. Round robin is optimal for fault-tolerant broadcasting on wireless networks. *Journal of Parallel and Distributed Computing*, 64(1):89–96, 2004.
8. A. Czumaj and W. Rytter. Broadcasting algorithms in radio networks with unknown topology. In *Proceedings of the Symposium on Foundations of Computer Science*, 2003.
9. B. Das and V. Bharghavan. Routing in ad-hoc networks using minimum connected dominating sets. In *Proceedings of the IEEE International Conference on Communications*, 1997.
10. L. Gasieniec, A. Pelc, and D. Peleg. The wakeup problem in synchronous broadcast systems. *SIAM Journal of Discrete Mathematics*, 14(2):207–222, 2001.
11. S. Gollakota and D. Katabi. Zigzag decoding: Combating hidden terminals in wireless networks. In *Proceedings of the ACM SIGCOMM Conference*, 2008.
12. T. Hernman and S. Tixeuil. A distributed TDMA slot assignment algorithm for wireless sensor networks. In *Proceedings of the International Workshop on Algorithmic Aspects of Wireless Sensor Networks*, 2004.
13. T. Jurdzinski and G. Stachowiak. Probabilistic algorithms for the wakeup problem in single-hop radio networks. In *Proceedings of the Symposium on Algorithms and Computation*, 2002.
14. D. Kowalski and A. Pelc. Broadcasting in undirected ad hoc radio networks. In *Proceedings of the International Symposium on Principles of Distributed Computing*, 2003.
15. D. Kowalski and A. Pelc. Time of radio broadcasting: Adaptiveness vs. obliviousness and randomization vs. determinism. In *Proceedings of the Colloquium on Structural Information and Communication Complexity*, 2003.
16. D. Kowalski and A. Pelc. Time of deterministic broadcasting in radio networks with local knowledge. *SIAM Journal on Computing*, 33(4):870–891, 2004.
17. D. R. Kowalski and A. Pelc. Deterministic broadcasting time in radio networks of unknown topology. In *Proceedings of the Symposium on Foundations of Computer Science*, 2002.
18. E. Kranakis, D. Krizanc, and A. Pelc. Fault-tolerant broadcasting in radio networks. In *Proceedings of the Annual European Symposium on Algorithms*, 1998.
19. F. Kuhn, N. Lynch, and C. Newport. The abstract MAC layer. Technical Report, MIT-CSAIL-TR-2009-021, 2009. http://hdl.handle.net/1721.1/45515
20. F. Kuhn, T. Moscibroda, and R. Wattenhofer. Initializing newly deployed ad hoc and sensor networks. In *Proceedings of the International Conference on Mobile Computing and Networking*, 2004.
21. F. Kuhn, T. Moscibroda, and R. Wattenhofer. Fault-tolerant clustering in ad hoc and sensor networks. In *Proceedings of the IEEE International Conference on Distributed Computing Systems*, 2006.
22. T. Moscibroda and R. Wattenhofer. Maximal independent sets in radio networks. In *Proceedings of the International Symposium on Principles of Distributed Computing*, 2005.
23. C. Scheideler, A. Richa, and P. Santi. An o(log n) dominating set protocol for wireless ad-hoc networks under the physical interference model. In *Proceedings of the International Symposium on Mobile Ad Hoc Networking and Computing*, 2008.
24. P.-J. Wan, K. Alzoubi, and O. Frieder. Distributed construction of connected dominating set in wireless ad hoc networks. *Mobile Networks and Applications*, 9(2):141–149, 2004.