Computer Science and Artificial Intelligence Laboratory
Technical Report

MIT-CSAIL-TR-2009-021 May 11, 2009

The Abstract MAC Layer

Fabian Kuhn, Nancy Lynch, and Calvin Newport

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

The Abstract MAC Layer

Fabian Kuhn[*] Nancy Lynch[†] Calvin Newport[‡]

May 1, 2009

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.

[*]MIT CSAIL, fkuhn@csail.mit.edu
[†]MIT CSAIL, lynch@csail.mit.edu
[‡]MIT CSAIL, cnewport@csail.mit.edu

1 Introduction

The study of bounds for mobile ad hoc networks is complicated by the large number of 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 by 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 of assumptions. 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.

The Abstract MAC Layer. We introduce a simple *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, or receiver-side collision detection with NACKs. A completely different kind of implementation might involve network coding methods, such as the ZigZag Decoding method 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. For instance, one might use it to study problems of communication (such as network-wide broadcast or point-to-point message routing); problems of establishing and maintaining basic structures (clusters, leaders, or spanning trees); problems of implementing higher-level services (group membership, resource management, data management, or consensus); or application-inspired problems (such as robot or vehicle control). More fundamentally, since our MAC layer does not provide senders with feedback about who received their messages, some basic problems must also be studied, such as local unicast with acknowledgement and neighbor discovery. In this sense, our layer exists at a lower level than existing practical layers, such as 802.11, which implement local unicast as a primary primitive. We treat such primitives as higher-level problems to be solved using our basic layer.

[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.

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, 2, 3, 4, 6, 5, 7, 8, 17, 15, 16, 18, 19]. 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 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 $O\left((D+k)F_{prog} + (k-1)F_{ack}\right)$, 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 the acknowledgement and progress bounds, 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 $O\left(D + k + bF_{prog} + F_{ack}\right)$, 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.

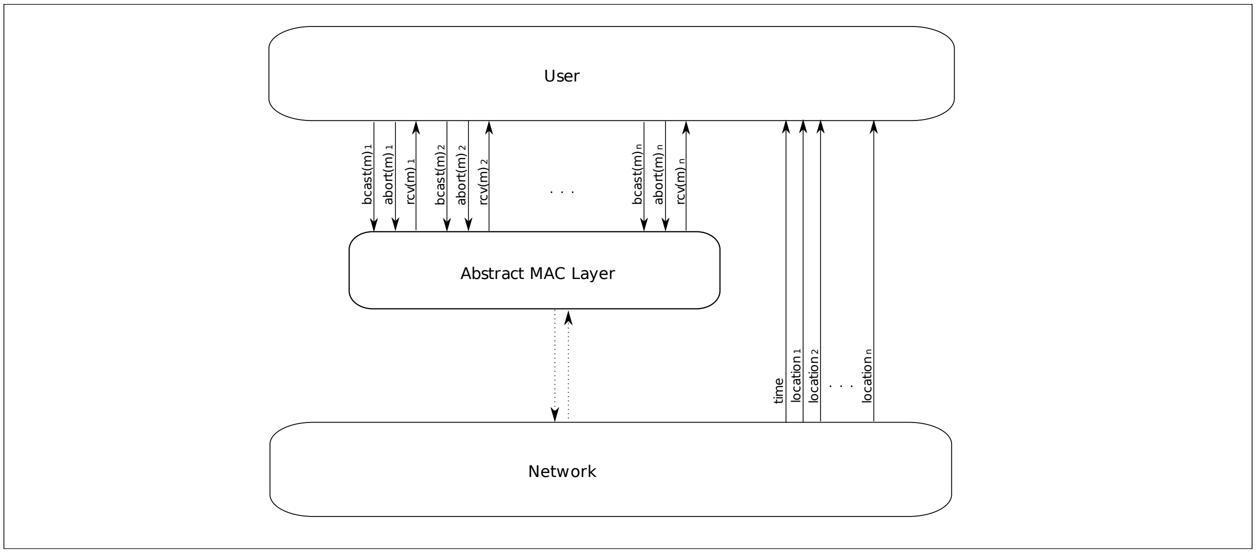

Figure 1: The MANET system.

2.1 System Components

A MANET system consists of three main components: the *network* automaton, the *abstract MAC layer* automaton, and the *user* automaton, connected as shown in Figure 1. We briefly describe each component:

The Network Automaton. The *network* automaton models the relevant properties of the real world: time, location, and physical layer behavior. We assume this automaton provides a physical layer interface that captures the low-level communication on the radio channel. It might also output user location and time. We do not assume an external interface for controlling motion. That is, we model mobility as entirely encapsulated within the network automaton, and independent of the behavior of other system components. In this paper, we assume that the location and time is accurate. It might be more practical to guarantee only an approximation of this information. For the protocols we consider, however, such a change would not generate significant modifications.

For every network automaton we assume there exists a pair of functions f_G and $f_{G'}$ that map from states to directed node interaction graphs (V, E) where $V = \{1, ..., n\}$ and $E \subseteq V \times V$. We call the graph $G = f_G(s)$, for some network state s, the *communication graph.* It captures the processes that are within communication range in that state. We call $G' = f_{G'}(s)$ the *interference graph.* It captures the processes within interference range. We separate communication from interference because in many practical radio network models the interference range exceeds the reliable communication range.[2] The algorithms we consider in this paper assume the common special case where $f_G(s) = f_{G'}(s)$, for all s. We introduce both graphs in our definitions, however, because we believe the examination of the general case, where the two graphs can differ, to be interesting future work. When we refer to the edge set E at a given point in an execution of a MANET system, we refer to the edge set from the graph $f_G(s)$ where s is the network state at that point. The same holds for E' with respect to $f_{G'}$. Throughout the paper, we use the term *network*

[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 future work.

as a shorthand to refer to the Network Automaton.

The Abstract MAC Layer Automaton. 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 via 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 fixed 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. Though real world MAC layers do not usually include an abort functionality, it seems both useful and feasible to implement, so we include it in our interface. As mentioned, the abstract MAC layer automaton connects to the network through the physical layer interface. It might also receive the network's location and time outputs. In Section 2.2, we describe the properties an abstract MAC layer automaton composed with a network automaton must satisfy to be considered an *abstract MAC layer service.*

The User Automaton. The *user* automaton models n user processes with unique labels from $\{1, ..., n\}$. Each process i connects to a MAC layer through the *bcast*, *abort*, *rcv*, and *ack* interface described above. It might also receive the network location and time outputs, depending on what is needed by the protocol being modeled.

2.2 Guarantees for the Abstract MAC Layer Service

Here we provide a set of properties that constrain the behavior of the abstract MAC layer automaton composed with a network automaton. An *abstract MAC layer service* is a composition that satisfies the constraints below. For simplicity we use the shorthand *MAC layer* to refer to this service throughout the paper. In these properties, and in the rest of the paper, we assume all executions are infinite.

Well-Formedness Properties. To define meaningful properties for an abstract MAC layer we must first assume some well-formedness conditions for the user automaton interacting with the layer. Fix an execution α of a MANET system. We say α is *user-well-formed* if and only if the following hold: (a) α contains at most one *bcast* event for each message m. (That is, all messages are unique.) (b) No process i submits more than one $abort(m)_i$ for any message m, and only submits an $abort(m)_i$ after a $bcast(m)_i$ but *not* after an $ack(m)_i$. (c) No process i submits a *bcast* until after its previous *bcast* (if any) ended with an *abort* or had a matching *ack* returned.

Let α be a user-well-formed execution. We continue with what properties the abstract MAC layer automaton must satisfy in this execution to be considered an abstract MAC layer service:

The Cause Function. We assume there exists a "cause" function that maps every $rcv(m)_j$ event to a preceding $bcast(m)_i$ event, where $i \neq j$, and that maps each $ack(m)_i$ and $abort(m)_i$ to a preceding $bcast(m)_i$.

Constraints on Message Behavior. We now define two safety conditions and one liveness condition regarding the relationships captured by the cause function:

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', $(i, j) \in E'$ (the edge set of the interference graph).

(b) **No duplicate receives:** No other $rcv(m)_j$ event caused by π precedes π'.

(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'.

   (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 $abort(m)_i$.

Notice, in the proximity and guaranteed communication bounds above, we use the interference graph G' to describe who *might* be able to communicate and the communication graph G to describe who is *guaranteed* to communicate. In practice, the former includes more processes than the latter.

Time Bounds. We now 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 interval of the broadcast. To help define these bounds, we provide a few auxiliary definitions:

Let f_{rcv}, f_{ack}, and f_{prog} be functions from natural numbers to nonnegative real numbers.

We will use these to bound delivery times for a specific message being delivered, an acknowledgement being received, and *some* message from among many being received, respectively, with respect to a given amount of contention. We call these the *delay functions.* Notice, in many MAC implementation we expect f_{prog} to yield smaller values than f_{ack}, as the time to deliver *some* message among many is typically better than the time to deliver a specific message.

We assume that f_{rcv}, f_{ack}, and f_{prog} are monotonically non-decreasing. That is, as the contention increases, so does the time to receive a specific message, an acknowledgement, and some message from among many, respectively.

Let ϵ_a be a non-negative constant. We use this constant to bound the amount of time beyond an *abort* when a message from the originating broadcast can still be received. We intend ϵ_a to cover messages that are already on the channel, or in the hardware send or receive buffers—thus unreachable by higher layers. We assume this constant to be small.

We use the term "message instance" to refer to a matched pair of $bcast_i$ and ack_i, or $bcast_i$ and $abort_i$ events. Let α be an execution, α' be a closed execution fragment within α[3], and j be a process. We then define $contend(\alpha, \alpha', j)$ as follows. This function returns the set of message instances in α that intersect with fragment α', 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, we define $connect(\alpha, \alpha', j)$ as follows. This function returns the set of message instances in α such that α' is 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 sufficiently long. Notice that $connect(\alpha, \alpha', j)$ is a subset of $contend(\alpha, \alpha', j)$.

[3] Formally, that means that there exist fragments α'' and α''' such that $\alpha = \alpha''\alpha'\alpha'''$, and moreover, the first state of α' is the last state of α''. Notice, this allows α' to begin and/or end in the middle of a trajectory.

Given an execution α and two events π and π' in α, the notation $\alpha[\pi, \pi']$ to describes the execution fragment within α that spans from π to π'.[4] We continue with the time bounds:

5. **Receive:** Suppose that a $bcast(m)_i$ event π causes a $rcv(m)_j$ 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 $contend(\alpha, \alpha[\pi, \pi'], j)$. In other words, the bound for when m must be received at j grows with the number of other *nearby* processes (e.g., connected in G') that have message instances intersecting with the instance in question. Furthermore, if there exists an $abort(m)_i$ event π'' such that π causes π'', then π' cannot occur more than ϵ_a time after π''. The ϵ_a constant requires that that an *abort* actually aborts the corresponding message within a bounded amount of time.

6. **Acknowledgement:** Suppose that a $bcast(m)_i$ event π causes an $ack(m)_i$ event π' in α. Let *ackcon* be the set containing i and every process j such that there exists a $rcv(m)_j$ 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 $\bigcup_{j \in ackcon} contend(\alpha, \alpha[\pi, \pi'], j)$. The acknowledgement bound is defined similarly to the receive bound, with the exception that we now include the contention at the sender and every receiver. This captures the intuition that an acknowledgement requires 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 $contend(\alpha, \alpha', j)$ is at most c, and $connect(\alpha, \alpha', j)$ is non-empty.

   (c) No $rcv(m)_j$ event from a message instance in $contend(\alpha, \alpha', j)$ occurs by the end of α'.

   In other words, the bound on when j should receive *some* message (when there is a least one message being sent by a neighbor in G), grows with the total number of processes that are in interference range. As mentioned in the introduction, this style of progress property was implicitly used in previous work—e.g., [3]—to derive bounds that are tighter than could be generated from a basic acknowledgement bound alone.

   A stronger version of (c) could require that the received message is sent by a neighbor with an edge to j in G, rather than just G', at some point during α'. This stronger property, if needed, might be implemented on top of a MAC layer guaranteeing the weaker property from above. The details would depend on the radio network model.

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, using, for example, one of the popular radio network models from the existing theoretical literature (e.g. [3, 10, 12, 13, 24]). Here we discuss, only informally, some basic ideas for implementations with the aim of providing some intuition regarding the type of concrete definitions

[4] Formally, by the definition of a fragment this must span from the point trajectory immediately preceding π to the point trajectory immediately following π'.

our delay functions might adopt in practice. For simplicity, we assume G and G' are fixed (e.g., the network is static) and undirected.

A common radio network model is the slotted broadcast model of [3, 10, 21, 23, 14]. This model assumes that the communication graph G and the interference graph G' are identical, that there is no collision detection (i.e., a collision cannot be distinguished from silence), and that a message from a sender i is correctly received by a neighbor j in a particular time slot if and only if i is the only neighbor of j broadcasting during this time slot. If we assume synchronized clocks, allowing for synchronized slots, a simple strategy derived from the *decay* function in [3] can be used to implement our abstract MAC layer service. In this approach, time is divided into synchronized epochs of $\Theta(\log \Delta)$ time slots, where Δ is the maximum node degree in G. Processes that have a message to broadcast, start broadcasting at the beginning of the next epoch. During an epoch, the probability of broadcasting is exponentially decreased from 1 to $1/\Delta$. It is guaranteed that every process that has at least one neighbor sending a message during an epoch receives at least one message with constant probability. We therefore get a progress delay function f_{prog} this is $O(\log \Delta)$ for all contention parameters, with probability $1 - \epsilon$. Similarly, our receive and acknowledgement delay functions, f_{rcv} and f_{ack}, are both $O(\Delta \log \Delta)$ for all contention parameters. Notice, this simple scheme requires time proportional only to the worst-case contention. More sophisticated MAC layer implementations, we expect, would include the contention parameter c in the delay function definitions.

If time is not synchronized, a technique similar to the one used in the context of the wake-up problem in single-hop networks [10, 14] and multi-hop networks [21, 23] can be used to synchronize the start of decay phases. Specifically, starting at a small probability, processes exponentially increase their sending probability until they either hear a message from a neighbor or decide to broadcast in a particular time slot. As soon as a process decides to broadcast, it resets its probability to the small start value. Let us assume that there are two physical communication channels.[5] As soon as a process hears a message, it starts the decay routine on the second communication channel. If we assume that the communication graph is a unit disk graph (or more generally a bounded growth graph as defined in [20]) and if the parameters are chosen correctly, it can be shown that with reasonable probability, the number of different start times of the decay routine is bounded in the neighborhood of every process. An adapted version of the decay function then allows a progress delay function that is polylogarithmic in Δ, and receive and acknowledgement delay functions that are $O(\Delta \cdot \text{polylog}(\Delta))$.

For the case where $G \neq G'$, similar techniques can be used. The details depend on how the radio network model captures communication and interference between nodes that are neighbors only in G'.

2.4 Multiple Abstract MAC Layer Automata

To simplify the analysis of multiple user protocols running on the same network it proves useful to allow for multiple independent abstract MAC automata in the same system (see Figure 2). In this scheme, each protocol (or perhaps even sub-protocol) connects with its own MAC automaton. These automata all connect to the same single network automaton. Each MAC automaton satisfies the specifications of the abstract MAC layer service with respect to the network.

Such an approach certainly simplifies analysis, but we also argue that it matches reality. Indeed, there exist a variety of practical realizations of multiple MAC automata. For example, most radio-equipped computing devices have access to many communication frequencies. If a device has several

[5] In [21], it is described how to simulate different communication channels at the cost of a polylogarithmic factor.

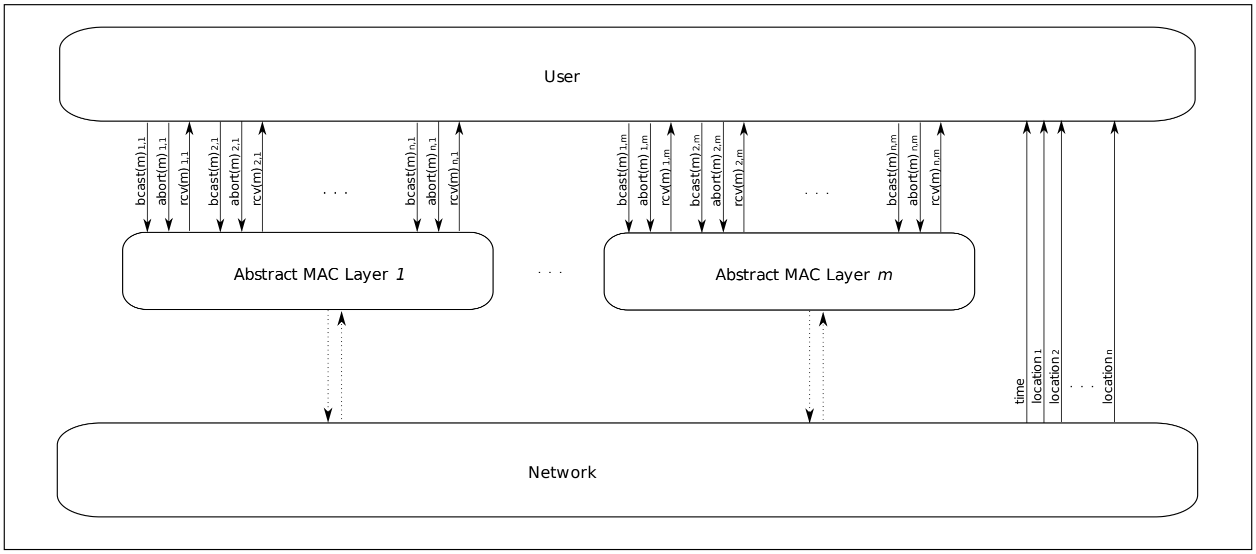

Figure 2: The MANET system with m abstract MAC layers.

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 TDM scheme to partition use of the frequency among the different logical MAC layers.

In other words, by allowing multiple independent abstract MAC automata in our model we remove the need for the protocol designer to tackle issue of contention between protocols and focus instead on proving properties about their individual behavior. The complexity of this contention will be captured by the concrete implementation of the multiple layers using a single network.

2.5 Upper Bounds on Message Delivery Times

We describe upper bounds on the message delivery time bounds of Section 2.2. We use these for algorithm development and analysis. We begin, however, with some graph notation used by these upper bounds and elsewhere in the paper.

Graph Notation. For any graph G, we use $D(G)$ to describe the *diameter* of G, that is the maximum $d(u, v)$, over all vertex pairs (u, v), where $d(u, v)$ describes the length of the shortest directed path between u and v.

Upper Bounds. We describe three constants that prove useful when describing message delivery times in our later analysis and definition of algorithms. For the following definitions, fix α to be a user-well-formed execution of a MAC layer and let Δ_α equal the maximum $\Delta(G')$ over all G' that appear in a state of α.

1. We define F_{prog} with respect to α to be $f_{prog}(\Delta_\alpha + 1)$. This bound captures the maximum amount of time until a process receives *some* message, starting from a point in the execution at which: (a) there is at least one message it should eventually receive (i.e., the message is sent by a process that is and will remain a neighbor in G); and (b) the message instances for previously received messages have completed. (The addition of 1 to Δ_α, in this definition and the two below, captures the possibility that the receiver is also sending a message, which would be counted in the *contend* set.)

2. We define F_{rcv} with respect to α to be $f_{rcv}(\Delta_\alpha + 1)$. This bound captures the maximum amount of time until a process receives a message sent by a neighbor in G.
3. We define F_{ack} with respect to α to be $f_{ack}((\Delta_\alpha + 1)^2)$. This bound captures the maximum amount of time until a process receives an acknowledgement for a message it sent. (The square is necessary to bound the worst-case value of the union calculated in the acknowledgement bound definition—a union that includes the contention at the sender *and* at all of its neighbors in G.)

Sometimes we need to define constants over all possible executions of a MAC layer (e.g., if we want to use them as upper bounds in a protocol definition). With this in mind we define F^+_{prog}, F^+_{rcv}, and F^+_{ack} to be the maximum values of F_{prog}, F_{rcv}, and F_{ack}, respectively, over all executions of the MAC layer under consideration.

3 The Multi-Message Broadcast Problem

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.

3.1 Preliminaries

In this section we assume a *static* network; i.e., for any given execution, the location of each node, and the G and G' graphs never change. Furthermore, we assume that $G = G'$, and the graphs are undirected; i.e., all communication links are bidirectional. We note that the algorithm and proof below would work for the general case, where $G \neq G'$, if one could guarantee the slightly stronger progress property that requires that the message received is from a neighbor in G.

We assume a message set $\mathcal{M}$ of possible broadcast messages. A user automaton is considered to be an *MMB protocol* only if its external interface includes an $arrive(m)_i$ input and $deliver(m)_i$ output for each user process i and message $m \in \mathcal{M}$.

We a say an execution of an MMB protocol is *MMB-well-formed* if and only if it contains at most one $arrive(m)_i$ event for each $m \in \mathcal{M}$. (That is, each broadcast message is unique). We say an MMB protocol *solves the MMB problem* if and only if for every MMB-well-formed execution of the MMB protocol composed with a MAC layer, the following hold: (a) For every $arrive(m)_i$ event and every process j, there exists a $deliver(m)_j$ event. (b) For every $m \in \mathcal{M}$ and process j, there exists at most one $deliver(m)_j$ event and it comes after an $arrive(m)_i$ event for some i.

3.2 The Basic Multi-Message Broadcast Protocol

We describe a simple MMB protocol that nonetheless achieves efficient runtime.

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 3.1. *The BMMB protocol solves the MMB problem.*

Proof. Let α be an MMB-well-formed execution of the BMMB protocol composed with a MAC layer. We first note that α is user-well-formed, as the definition of the protocol has each process wait for an *ack* before submitting its next *bcast*. There are no *aborts*. It follows that the abstract MAC layer properties are satisfied by the MAC layer.

Let $arrive(m)_i$ be an event in α. At the point when this event occurs, the size of the MMB queue at process i is of some finite size. Let us call this q. After at most $(q+1)F_{ack}$ time after this point, i will have succeeded in transmitting all q elements ahead of m in its queue, and then m itself, to its neighbors. We can reapply this argument D times for each message, to show that it eventually arrives (and is delivered) at all processes. □

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 in case 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.[6]

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 $K(m) = \{m' \in \mathcal{M}$: *an* $arrive(m')$ *event precedes the last* $deliver(m)$ *event and the* $clear(m')$ *event follows the* $arrive(m)_i$ *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, by contrast, does better. It separates kF_{ack} from the diameter, $D(G)$, instead multiplying this term only 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 3.2. *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 3.2 is a direct consequence of the following lemma.

Lemma 3.3. *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*

[6] 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.

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

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 $t_{d,\ell} + F_{ack}$.

Proof. We first define sets $G_i(t) \subseteq \mathcal{M}'$ and $C_i(t) \subseteq \mathcal{M}'$ of messages for every process i and time $t \geq 0$ in execution α of the BMMB protocol. $G_i(t)$ is the set of messages $m' \in \mathcal{M}'$ for which a $get(m')_i$ event occurs by time t. $C_i(t) \subseteq G_i(t)$ is the set of messages $m' \in \mathcal{M}'$ for which the $ack(m')_i$ event occurs by time t. Hence, $G_i(t)$ is the set of messages that have been received by process i and $C_i(t)$ is the set of messages that process i has finished processing by time t. From the MAC layer properties and the definition of the BMMB protocol, we obtain the following three statements:

a) Consider a process i. If $G_i(t) \cap (G_{i'}(t) \setminus C_{i'}(t)) = \emptyset$ for every neighbor process i' of i and there is a neighbor i' for which $G_{i'}(t) \setminus C_{i'}(t) \neq \emptyset$, then a $get(m')_i$ event occurs after time t and by time $t + F_{prog}$. Hence if the neighbors of i only process messages m' at time t for which $get(m')_i$ has not occurred, i will soon receive a new message.

b) For every process i and message $m' \in C_i(t)$, a $get(m')_{i'}$ event occurs at all neighbors i' of i by time t.

c) Assume that a message m' is in the queue of a process i at time t and let $\mathcal{Q}$, $|\mathcal{Q}| = q$, be the set of messages in i's queue ahead of m' at time t. For $k \leq q$, by time $t + k \cdot F_{ack}$, there are $ack(m'')_i$ events for k messages $m'' \in \mathcal{Q}$ and by time $t + (q+1)F_{ack}$, an $ack(m')_i$ event occurs.

We prove the lemma by induction on ℓ and for every ℓ by induction on d. For all ℓ, as a base case for the induction on d, we start by considering the case $d = 0$, i.e., the case where $j = i_0$. Let $\mathcal{M}'_0 \subseteq \mathcal{M}'$, $|\mathcal{M}''_0| = \ell_0$, be the set of messages $m' \in \mathcal{M}'$ that are in i's queue at the time of the $arrive(m)_i$ and thus of the $get(m)_i$ event. For $\ell \leq \ell_0$, by statement c) above, by time $t_0 + \ell \cdot F_{ack}$, $ack(m')_i$ events occur for ℓ messages in $\mathcal{M}''_0$ and for $\ell > \ell_0$, $ack(m)_i$ occurs by time $t_0 + (\ell_0 + 1) \cdot F_{ack}$. Hence, for $\ell \leq \ell_0$, statement (2) is true whereas for $\ell > \ell_0$, statement (1) is true.

We now come to the base case of the induction on ℓ, $\ell = 1$ and $d > 0$. By induction (on d), there is a neighbor j' of j for which $G_{j'}(t_{d-1,1})$ is non-empty. If $G_j(t_{d-1,1})$ is non-empty, there is a message m' for which a $get(m')_j$ event occurs by time $t_{d-1,1}$ and by property c), an $ack(m')_j$ event occurs by time $t_{d-1,1} + F_{ack}$. If $G_j(t_{d-1,1}) = \emptyset$, there is a message $m' \in G_{j'}(t_{d-1,1})$ for which a $get(m')_j$ occurs by time $t_{d-1,1} + F_{prog} = t_{d,1}$ by property a) and an $ack(m')_j$ occurs by time $t_{d,1} + F_{ack}$ by property c). The lemma is therefore also true for all d and $\ell = 1$.

Let us now consider the case $d > 0$ and $\ell > 1$. Assume that for some neighbor j' of j, statement (1) is true for $\ell - 1$. Then, by applying property b), there is a $get(m)_j$ event by time:

$$\begin{aligned} t_{d+1,\ell-1} + F_{ack} &= (d + 1 + 2(\ell - 1) - 2) \cdot F_{prog} + (\ell - 2) \cdot F_{ack} + F_{ack} \\ &< (d + 2\ell - 2) \cdot F_{prog} + (\ell - 1) \cdot F_{ack} \\ &= t_{d,\ell}. \end{aligned}$$

If the $ack(m)_j$ event does not occur by time $t_{d,\ell-1} + F_{ack} < t_{d,\ell}$, by induction (on ℓ), $ack(m')_j$ events for a set $\mathcal{M}'_0$ of $\ell - 1$ messages $m' \neq m$ happen before time $t_{d,\ell}$. If m reaches the front of

j's queue by time $t_{d,\ell}$, the $ack(m)_j$ event happens by time $t_{d,\ell} + F_{ack}$ by statement c). Otherwise, let m' be the first message in j's queue at time $t_{d,\ell}$. Note that $m' \notin \mathcal{M}'_0$. By statement c), the $ack(m')_j$ event occurs by time $t_{d,\ell} + F_{ack}$.

It remains to consider cases where for all neighbors j' of j only statement (2) is true for $\ell - 1$. Then, by time $t_{d+1,\ell-1} + F_{ack}$, $\ell - 1$ different $ack(m')_{j'}$ events occur for each neighbor j'. Assume that there are two neighbors j' and j'' for which the sets of the first $\ell - 1$ messages $m' \in \mathcal{M}'$ for which $get(m)_{j'}/get(m)_{j''}$ events and thus (by the FIFO ordering) $ack(m')_{j'}/ack(m')_{j''}$ events occur are different. Because by statement b), there is a $get(m')_j$ event by time t for every message for which there is an $ack(m')_{j'}$ event at a neighbor j' by time t, there are at least ℓ different $get(m')_j$ events by time $t_{d+1,\ell-1} + F_{ack} \leq t_{d,\ell}$ in this case. By induction (on ℓ) the first $\ell - 1$ $ack(m')_j$ events occur by time $t_{d,\ell-1} + F_{ack} < t_{d,\ell}$ and by statement c), there is thus another $ack(m')_j$ event by time $t_{d,\ell} + F_{ack}$.

Finally, let us consider the last remaining case where all neighbors of j receive the same first $\ell - 1$ messages $m' \neq m$ by time $t_{d+1,\ell-1}$. Then, there is a $get(m')_j$ event by time $t_{d+1,\ell-1} + F_{ack}$ for all these messages by statement b). If by this time, there is another message m'' for which there is a $get(m'')_j$ event, the conclusion of the lemma is true for the same reason as in the previous case. If there is no such message m'', for $t = t_{d+1,\ell-1} + F_{ack}$, we have $G_j(t) \cap (G_{j'}(t) \setminus C_{j'}(t)) = \emptyset$ for every neighbor j' of j. By induction (on d), there is a neighbor j' at distance $d - 1$ to i_0 for which a $get(m'')_{j'}$ event for an additional message m'' occurs by time $t_{d-1,\ell} = t_{d+1,\ell-1} + F_{ack}$. Hence, by property a), there is a $get(m'')_j$ event by time $t_{d-1,\ell} + F_{prog} = t_{d,\ell}$ for an ℓ^{th} message m''. By induction on ℓ, the $ack(m')_j$ events for the first $\ell - 1$ messages occur by time $t_{d,\ell-1} + F_{ack} < t_{d,\ell}$ and thus by statement c), the $ack(m'')_j$ event occurs by time $t_{d,\ell} + F_{ack}$ which completes the proof. $\square$

Proof of Theorem 3.2. Let t_0 be the time when the $arrive(m)_i$ event occurs. Assume that $|K(m)| \leq k$. We show that the last $deliver(m)$ event occurs by time $t_1 = t_0 + (D(G) + 2k - 2)F_{prog} + (k-1)F_{ack}$.

Let $\mathcal{M}' \subseteq \mathcal{M}$ be the set of messages m' for which $arrive(m)_i$ precedes $clear(m')$. By Lemma 3.3 for all processes j by time t_1, a $get(m)_j$ event occurs or there exists a set $\mathcal{M}'' \subseteq \mathcal{M}'$ of size $|\mathcal{M}''| = k$ such that $get(m')_j$ events occur for all messages $m' \in \mathcal{M}''$. In the first case, since the $deliver(m)_j$ event occurs at the same real time as the $get(m)_j$ event, we have satisfied the Theorem bound.

We can therefore assume that there is a set $\mathcal{M}'' \subseteq \mathcal{M}'$ of size k such that for all messages $m' \in \mathcal{M}''$, either a $get(m')_j$ event happens for all processes j by time t_1. By the definition of $K(m)$ and $\mathcal{M}'$, we have that $K(m) \subseteq \mathcal{M}'$. Hence, from the assumption that the last $deliver(m)$ event happens after time t_1 and the definition of $K(m)$, we have $m' \in K(m)$ for every message $m' \in \mathcal{M}'$ for which a $get(m')$ event occurs by time t_1. As a consequence, $\mathcal{M}'' \subseteq K(m)$. Because $|\mathcal{M}''| = k$ and $|K(m)| \leq k$, this implies that $\mathcal{M}'' = K(m)$. Because $m \in K(m)$, there is a $get(m)_j$ event and thus a $deliver(m)_j$ for every process j by time t_1, as needed. $\square$

4 Regionalized Networks

Recall that our model requires the network automaton to encode and report the location of every node at all times. It does not, however, place any constraints on the geography in which these locations reside or their relation to G and G'. In this section we define such general constraints, which we use in Section 6 to improve the complexity of our MMB solutions.

Preliminaries. Let L be a set of locations. (For example, this could describe points in the 2D plane.) Let R be a set of regions ids. Let *region mapping reg* be a mapping $reg : L \rightarrow R$. And let N_R be a neighbor relation among regions in R. Consider the graph $G_{region} = (R, N_R)$. We call G_{region} a *region communication graph* if and only if it is connected. Let N'_R be some neighbor relation such that $N_R \subseteq N'_R$. Consider the graph $G'_{region} = (R, N'_R)$. We call G'_{region} a *region interference graph* if and only if $\Delta(G'_{region}) = O(1)$.

Regionalized Network. Fix a network $\mathcal{N}$. Let L describe the set of locations used by $\mathcal{N}$. Given any state s of $\mathcal{N}$, and node i, we use the notation $loc(i)$ to refer to the location of node i encoded by $\mathcal{N}$ in s. Let R be a set of region ids, reg be a region mapping from L to R, and N_R and N'_R be neighbor relations for R such that $N_R \subseteq N'_R$. Assume $G_{region} = (R, N_R)$ is a region communication graph and $G'_{region} = (R, N'_R)$ is a region interference graph.

We say network $\mathcal{N}$ is *regionalized* with respect to L, R, reg, N_R, and N'_R, if and only if for every execution of $\mathcal{N}$, and every point in the execution:

1. For every pair of nodes i and j such that $reg(loc(i)) = reg(loc(j))$ or $(reg(loc(i)), reg(loc(j))) \in N_R$: $(i, j) \in E$.

2. For every pair of nodes i and j such that $(i, j) \in E'$, 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, then they must be connected in G, and if two nodes are connected in G' (i.e., can interfere with each other) then they are in the same region or in regions that are neighbors in the region interference graph. It follows that the region communication graph captures which regions are *always* in communication range while the region interference graph captures which regions *could* be in interference range. For example, a simple grid topology where we set the length of the grid square diagonal to be half the reliable broadcast range, and classifying grid squares sharing an edge as neighbors, might be used to define a regionalization that matches these constraints.

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 The Leader Election Problem

Notice that the BMMB protocol did not make use of location information or synchronized clocks. This begs an obvious question: can we do better if the processes know this information? To answer this question, we first study the problem of local leader election in a single region of our fixed regionalized network, then use this protocol to elect a leader in every region. Our solutions rely on synchronized clocks, running at the same fixed rate, to coordinate the beginning and end of the relevant phases among the different processes. The resulting leader backbone forms a connected dominating set (CDS). The use of a CDS in radio networks is a common strategy; c.f.,[25, 22, 21, 26, 9]. Unlike this existing work, however, our task is simplified by both the availability of location information and the lack of need to deal explicitly with contention on the channel.

5.1 Solving the RLE Problem

We say a user automaton is a *RLE protocol* if and only if it has a $leader(r)_i$ and $notleader(r)_i$ output for every process i and every region r. We say a RLE protocol solves *solves the RLE problem for region r by time t* if and only if for every execution α of the protocol composed with a MAC layer the following hold:

1. By time t in α exactly one process i such that $reg(loc(i))) = r$ outputs $leader(r)_i$ and every process $j \neq i$ such that $reg(loc(j)) = r$, outputs $notleader(r)_j$.

2. For every process i such that $reg(loc(i)) = r$, there exists at most one event π in α such that $\pi = leader(r')_i$ or $\pi = notleader(r')_i$ for some region r'.

We continue with some RLE protocol definitions. Throughout the definitions that follow, we assume a fixed positive constant ϵ_b, which we will use in multiple protocols to add an extra buffer to the end of intervals calculated to match the length of the message receive time bounds. (The use of this extra buffer is a technicality required by the fact that the TIOA model allows multiple events can occur at the same time; the ϵ_b is used to make sure a check of received messages happens *after* every relevant message receive event has occurred.) We also assume that processes know the value of F^+_{ack} and F^+_{prog} in advance, as these are upper bounds on the delay for all possible executions of the network, and can therefore be seen as modeling system constants.

5.2 The Basic RLE Protocol

The Basic Regional Leader Election (BRLE) protocol is described below:

> **The r-Basic Regional Leader Election (BRLE) Protocol**
> In the r-BRLE protocol for some region r, each process i in r behaves as follows. At time 0, i broadcasts its own id. At time $F^+_{ack} + \epsilon_b$, i processes its set of received messages. If i is greater than every id described in a received message, then i triggers a $leader(r)_i$ output. Else it triggers $notleader(r)_i$. The output happens instantaneously at time $F^+_{ack} + \epsilon_b$ (i.e., we assume a processing time of 0).

Theorem 5.1. *For any region r, the r-BRLE protocol solves the RLE problem for region r by time $F^+_{ack} + \epsilon_b$.*

Proof. We begin by noting that the protocol preserves user-well-formedness, so we can assume the abstract MAC layer properties hold.

Fix some pair of processes i and j in r. We first prove that i and j receive each other's messages. Both processes broadcast at time 0. According to the *termination* property of the MAC layer, each broadcast is the cause of either an *abort* or *ack*. Because neither aborts, each must eventually cause an *ack*.

By the definition of a regionalized network, we know $(i, j) \in E$. Therefore, by the *acknowledgement correctness* property, i receives j's message and j receives i's message, both before their *ack* events.

By the *acknowledgement* time bound, these receive events and the subsequent *ack*s, must occur by time F^+_{ack}. It follows that at time $F^+_{ack} + \epsilon_b > F^+_{ack}$, the messages have been received. Apply this argument to all pairs in the region to show that all processes in r receive all messages in r, therefore they make a common leader decision. $\square$

5.3 The Fast Regional Leader Election Protocol

As mentioned, we assume that in many MAC layer implementations, f_{prog} will be much smaller than f_{ack}. To accommodate this possibility we describe a Fast Regional Leader Election (FRLE) protocol that relies only on f_{prog} and the size of the id space. For the following, let b be the number of bits needed to describe the id space. (A common assumption is that $b = \lceil \lg n \rceil$, but this might not always hold.)

> **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 5.2. *For any region r, the r-FRLE protocol solves the RLE problem for region r by time $b(F^+_{prog} + \epsilon_a + \epsilon_b)$.*

Proof. We begin by noting that the protocol preserves user-well-formedness, so we can assume the abstract MAC layer properties hold.

We prove the following: if *any* process broadcasts at the beginning of phase p, then *every* process that does not broadcast receives at least one phase p message by the end of the phase.

Fix a phase p. Let Π be the set of *bcast* events that occur at the beginning of p. These broadcasts are the only messages that intersect with the interval defined by phase p (remember: all broadcasts from previous stages were aborted by the end of those phases).

Fix a process i that does not broadcast in this phase. The definition of a static regionalized network provides that all processes in r are neighbors of i in G for this entire interval. So far we have satisfied conditions (a) and (b) of the *progress* time bound.

It follows that condition (c) *cannot* also be satisfied. Therefore, a $rcv(m)_i$ caused by some broadcast in Π must occur for i in this interval. Satisfying our claim.

We are left to prove that the search logic produces a single leader. This follows from two observations. First, it is impossible for all processes that are non-terminated at the beginning of some phase to submit $notleader(r)$ outputs at the end of the phase. Assume for contradiction that this occurs at the end of some phase p. To terminate a process must not broadcast and also receive a message. It follows that *some* non-terminated process broadcasts during phase p. (Recall: from our above argument that the only message intervals that intersect a given phase p are those that contain phase p messages.) By definition, a process that broadcasts in a given phase cannot be terminated in that phase. A contradiction.

Second, we show that two or more processes cannot both survive all b phases to become leader. Assume for contradiction that both i and j become leader. Because their ids are unique, there must be one bit position in which they differ. Without loss of generality assume it is position k and that i has a 1 in this position while j has a 0. It follows that in phase k, i broadcasts and j does not. By our claim at the beginning of the proof, j will receive some message in this phase—leading it to terminate. Another contradiction.

It follows that the protocol elects one and only one leader as required. □

5.4 The Complete Regional Leader Election Protocol

The RLE solutions described above work for a single region. It proves useful, however, to elect such a leader in *every* region in a regionalized network. Below we describe a protocol that elects a leader in every region. As with FRLE, let b be the minimum number of 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. 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.

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, adding a fixed offset to the time input used by FRLE such that the transformed time at the beginning of the interval evaluates to 0.

Theorem 5.3. *The CRLE protocol solves RLE problem for* every *region by time* $\Theta\left(b \cdot (F^+_{prog} + \epsilon_a)\right)$

Proof. Using standard techniques we can construct a minimal TDMA schedule T to contain $\Delta(G'_{region}) = O(1)$ sets. By the definition of T, each process runs FLRE exactly once, during the time allocated to the slot containing its region. No two regions running the protocol concurrently are within interference range, and all outstanding messages were aborted before the protocol begins (by definition of FLRE, every process aborts any non-*ack*'d messages by the end of each slot), so from the perspective of the processes running FLRE it is as if their region is running it alone starting at time 0. The correctness of their outputs follows from Theorem 5.2. □

6 Regional Multi-Message Broadcast

We combine the CRLE protocol from the previous section with the BMMB protocol to generate a new protocol we call Regional Multi-Message Broadcast (RMMB). The resulting protocol improves the performance of BMMB by confining the propagation of messages to the low-degree backbone of leaders elected by CRLE.

The Regional Multi-Message Broadcast (RMMB) Protocol
To simplify analysis, the RMMB protocol makes use of three independent MAC automata (see Section 2.4 for more on the use of multiple MAC automata). We label the automata *collect*, *leader election*, and *broadcast*. At a high-level, we use the *leader election* MAC automaton to elect a leader in each region using CRLE. We use the *broadcast* automaton to run BMMB on this leader backbone once CRLE terminates. And we use the *collect* automaton to transfer messages from *arrive* events at non-leaders to the leader in the region. This collect protocol runs concurrently with the CRLE and BMMB protocols. Before CRLE completes, all processes running collect will queue the messages in case they are elected leader.

State. Each process i maintains a *broadcast* queue and an *arrive* queue. Both are initially empty. It also maintains a *leader* flag which is initially $false$, and two sets, *delivered* and *rcvd*, both initially empty.

Leader Election. Each process i in region r behaves as follows with the leader election MAC automaton. Starting at time 0, i executes the ϵ-CRLE leader election protocol for some fixed positive constant ϵ. At the end of the protocol (i.e., after the time dedicated to the last set in the TDMA schedule has transpired), i sets its *leader* flag to *true* if and only if it triggered a $leader(r)_i$ output during the CRLE protocol.

Collect. Each process i in region r behaves as follows with the collect MAC automaton. When an $arrive(m)_i$ or $rcv((m,r))_i$ event occurs,[7] i places the message (m or (m,r)) on the back of its *arrive* queue. As soon as i's *arrive* queue becomes non-empty it does the following. If the element at the head of the queue is a single message m', it removes m' from the *arrive* queue, performs a $deliver(m')_i$ output, adds m' to the *delivered* set, places m' on the back of the *broadcast* queue, and then *propagates* m' before moving on to the next element in the *arrive* queue (if any). The propagate step depends on the status of the *leader* flag. If $leader = true$, then propagate is a *noop* and takes up 0 time. If $leader = false$ then i broadcasts (m', r) and then waits for the corresponding $ack((m', r))_i$.

If the element at the head of the *arrive* queue is an (m', r) message, then i removes (m', r) from the queue, performs a $deliver(m')_i$ output, adds m' to the *delivered* set, places m' on the back of the *broadcast* queue. (There is no propagate step for this case.)

Broadcast. Each process i in region r behaves as follows with the broadcast MAC automaton. Process i waits the fixed amount of time required for the CRLE protocol executed on the leader election MAC automaton to complete. If i has $leader = true$ at this point, then it executes the BMMB protocol using the *broadcast* queue maintained by the protocol running on the collect MAC automaton, 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 automata, if m is not in the delivered set it performs a $deliver(m)_i$ output and then adds m to the *delivered* set.

We continue with the relevant theorems.

Theorem 6.1. *The RMMB protocol solves the MMB problem.*

Proof. Let α be an MMB-well-formed execution of the RMMB protocol composed with the specified MAC automata and our fixed regionalized network. We note that α is user-well-formed, so it follows that the abstract MAC layer service properties are satisfied for each MAC automaton.

Let $arrive(m)_i$ be an event in α. At the point when this event occurs, the size of the *arrive* queue at process i is of some finite size. Let us call this q. After at most $(q+1)F_{ack}$ time after this point, i will have succeeded in transmitting all q elements ahead of m in its queue, and then m itself, to its neighbors in G, a set which includes all processes in its same region. All processes in its region add these messages to their *arrive* queue and therefore eventually their *broadcast* queue, in the order they arrive. This includes the single process j that eventually sets its *leader* flag to *true* in this region.

There exists a point in α, therefore, where j both has $leader = true$ and m in its *broadcast* queue. At this point, we can apply the argument from the correctness theorem for BMMB (Theorem 3.1) to argue that this message eventually gets to a leader in every region, and from there, to every process in every region, in finite time. □

We now turn our attention to the time complexity of RMMB. The key observation is that RMMB executes on a backbone of leaders. The contention on the broadcast MAC automaton is at most $\Delta(G'_{region}) = O(1)$; therefore the relevant delay functions, f_{ack} and f_{prog}, both evaluate to constants.

For the following theorem, we assume a bound on the rate of of *arrive* events at individual processes. Specifically, we bound the rate of *arrive* events at each process at $O(1/F_{ack})$—preventing

[7] The $rcv((m,r))_i$ inputs describes a message (m,r) arriving from the MAC layer at i.

any process from having more than a constant number of messages in its *arrive* queue at any one point.

The bound presented below improves the BMMB by removing the F_{ack} and F_{prog} as multiplicative factors on k and $D(G)$, respectively.

We note that our restriction on arrival rates would not change the bound for the original BMMB protocol (increased message arrivals is captured by $K(m)$). In the regionalized setting, however, without this restriction we would have to complicate our analysis to take into account the possible sizes of the arrive queues of non-leader processes. For faster arrival rates these queue sizes would increase our time bound, perhaps toward infinity as the execution continues and queues continue to grow faster then they can be emptied. Even with no arrival rate bound, we can still show that with a simple improvement (the leader acknowledging collect messages using a separate MAC automaton), *some* messages will arrive at the leader every F_{prog} time if there are any messages to be sent. This style of analysis might lead to a theorem that captures a bound on throughput, not the fate of a specific message. It remains interesting future work to formalize such bounds.

Finally, as before, we use b to describe the number of bits required to describe the id space.

Theorem 6.2. *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 $O\left(max\{b(F^+_{prog} + \epsilon_a), F_{ack}\} + D(G) + k\right)$.*

Proof. We apply Theorem 3.2 to BMMB running on the leaders to get a bound on the interval from when m arrives at a leader until the final $deliver(m)$ at a leader. We then modify this bound to take into account the time required for m to move between non-leaders and leaders.

Let j be the process that is elected leader in i's region. Let π be the later of the following two events: m being received (or arriving in the case $i = j$) or CRLE completes. By Theorem 3.2, the interval between π and the last $deliver(m)$ event at a leader is at most $(D(G) + 2k - 2)F_{prog} + (k-1)F_{ack}$. We can improve this further by observing that on the broadcast MAC automaton only the leaders broadcast. This constraint improves the parameters passed to f_{prog} and f_{ack} to be no larger than $\Delta(G'_L)$, where G'_L is the subgraph of G' including only leaders. By the definition of a regionalized network, $\Delta(G'_L) = O(1)$, therefore we replace F_{prog} and F_{ack} with two $O(1)$ terms yielding a bound that is $O\left(D(G) + k\right)$.

We now increase the bound to include the time that might elapse between $arrive(m)_i$ and π. There are two cases. If π describes m being received by j, then by our arrival rate assumption when the $arrive(m)_i$ occurs, the size of the *arrive* queue at i is $O(1)$. We apply the same argument as for claim (c) from the proof of Lemma 3.3 to establish that m arrives at the leader in i's region within $O(F_{ack})$ time.

If on the other hand π describes CRLE completing then by Theorem 5.3 this requires $O\left(b \cdot (F^+_{prog} + \epsilon_a)\right)$ time. In either case the time is at most $max\{b(F^+_{prog} + \epsilon_a), F_{ack}\}$.

We increase the bound further to include the time that might elapse between the last $deliver(m)$ event a leader and the last $deliver(m)$ at any process. Consider the $deliver(m)_l$ event for any leader l. At this point in the execution l has at most k messages ahead of m in its *broadcast* queue. Because the contention on the broadcast MAC automata is constant l will broadcast and receive an *ack* for m in $O(k)$ time. (Notice, it follows that non-leader processes receive the broadcast messages automatically simply by being in the same region as a leader that broadcasts.) This does not change the asymptotics of our bound, leaving us with a result that matches the theorem statement. □

7 Adapting RMMB for Mobile Networks

Our previous results assumed static networks. Here we adjust RMMB to tolerate (bounded) mobility. The original RMMB protocol was comprised of three sub-protocols: leader election, collect, and broadcast. Our mobile RMMB protocol uses the same structure, with these sub-protocols modified as needed to accommodate mobility. As before, we make use of leader election, collect, and broadcast MAC automata. We now include a fourth MAC automaton called *leader-ack* used by the collect sub-protocol in addition to the collect automaton.

7.1 The Region Exit Bound

We assume each process maintains a *region exit bound* state variable which in all execution states contains a time value no later than the time when the process will next exit the current region. We assume that while a process remains within a region, this value does not change. In some settings, enough information might be available to calculate a non-trivial bound (e.g., as a function of velocity and position); in other cases, such information might not be available. However, even without any information on process mobility, the constraints of the bound can be trivially satisfied by setting the variable equal to the current time upon first entering a region.

We continue by describing each of the sub-protocols used by our mobile RMMB protocol.

7.2 Mobile leader election

Our mobile leader election sub-protocol should satisfy the following properties in any given execution α. There exist two time bounds t_1 and t_2, such that, for every region r: (a) at every point in α, there is at most one leader in r; (b) if at every point in α, there is some process in r whose region exit bound is at least t_1 greater than the current time, then there is always a leader in r; and (c) for every process j that transforms from leader to not leader while in r, the exit bound of j in r is less than t_2 beyond the current time when the transformation occurs.

Below we describe the modified leader election sub-protocol of RMMB. In the following, let $t_{CF} = t_{CRLE} + t_{FLRE}$, where t_{CRLE} describes the time required to complete an instance of CRLE and t_{FLRE} describes the time required to complete an instance of FRLE.

> **The t-Mobile Leader Election Sub-Protocol.**
> In the t-mobile leader election sub-protocol, for some time bound t, each process i runs the leader election sub-protocol from RMMB with the following mobility-inspired modifications. Process i now runs CRLE continuously, launching a new instance immediately following the global completion of the previous. (Recall that an instance of CRLE requires a fixed amount of time, so all processes are trivially synchronized.) At the beginning of each TDMA slot s in a given instance of CRLE, i executes the $(reg(loc(i)), \epsilon)$-FRLE sub-protocol, for some fixed constant ϵ, if and only if $reg(loc(i)) \in s$. It does not join ongoing instances of FRLE when entering a new region in the middle of a slot. In addition, instead of using its own id in the sub-protocol, i calculates and uses a *leader id* (lid) which is a triple (ℓ, x, i), where ℓ is a boolean flag. The flag ℓ is set to 1 if and only if: (a) i is currently the leader in the region; and (b) its exit bound is at least $max\{t, t_{CF}\}$ beyond the current time. The value x encodes the region exit bound for i at this point. Process i calculates its lid at the beginning of a slot and uses that same lid throughout the FRLE instance.

Mobile leader election satisfies desired property (a) by the safety of FRLE and the restriction that processes execute FRLE only if they are in the region at the beginning of the corresponding slot. Property (b) is satisfied for $t_1 = t_{CF}$. To see why, consider any region r, and the first TDMA slot for r. By assumption, at least one process knows it will be around for at least t_{CF} more time.

Because the lid encodes this dwell time, and FRLE favors higher ids, then the leader elected will remain in the region for at least t_{CLRE} additional time ($t_{CF} = T_{CRLE} + T_{FRLE}$, and t_{FRLE} was used by the slot)—enough to make it to the end of the next slot for this region. At the beginning of this next slot, there will be at least one process remaining with an exit bound at least t_{CF} in the future. If the leader also has at least this much dwell time, it will remain leader and remain in the region until the next election, otherwise, a new process will be elected leader that will remain in the region until the next election. The argument can be extended to all slots. Finally, we note that property (c) holds for $t_2 = \max\{t, t_{CF}\}$.

7.3 Mobile Collect

Our mobile collect sub-protocol should satisfy the following property in any given execution for a particular function f_{col} from natural numbers to positive reals: (a) if an $arrive(m)_i$ event occurs for some process i in region r with leader j, such that i's *arrive* queue is of size q at this point, and both i and j remain in r for at least $f_{col}(q)$ time after the event, and j remains a leader during this time, then within this interval m is removed from i's *arrive* queue and added to j's *broadcast* queue.

Below we describe the modified collect sub-protocol of RMMB.

The Mobile Collect Sub-Protocol.
In the mobile collect sub-protocol each process i runs the collect sub-protocol from RMMB with the following mobility-inspired modifications. For every $arrive(m)_i$ event occurring at some time t, we label m with a timestamp (t, k, i), where k describes the number of *arrive* events that occurred at i at time t before this event. We add the labelled message to the *arrive* queue which we now maintain sorted lexicographically on the timestamps. In addition, we now require each leader to send an acknowledgement message for each message received from a non-leader. The leader can use the *leader-ack* MAC automaton for these acknowledgements. (Because contention on this MAC automaton is constant, these acknowledgements can be sent and received fast.) A non-leader receiving an acknowledgement message from a leader removes the message from its *arrive* queue. It also aborts the broadcast of this message if it has not already received a corresponding MAC *ack*. A non-leader continually rebroadcasts the message at the head of its queue (waiting for a MAC *ack* before broadcasting again) until it receives an acknowledgement from a leader.

Mobile collect satisfies desired property (a) for $f_{col}(q) = F_{ack} \cdot (q + 1)$.

Notice, that if only process i is broadcasting on the collect automaton, this function can be improved to $O(F_{prog} \cdot (q+1))$, as the leader's acknowledgements arrive in $O(1)$ time on the leader-ack MAC automaton.

7.4 Mobile Broadcast

Our mobile broadcast sub-protocol should satisfy the following property in any given execution for a particular function f_{bcast} from natural numbers to positive reals: (a) if m is added to a *broadcast* queue of length q at leader process i in region r, and this process, as well as the leaders of all regions neighboring r at this point, remain leaders for $f_{bcast}(q)$ time, then by the end of this interval m has been removed from i's *broadcast* queue and has been present in the *broadcast* queue of *some* leader in each region neighboring r at *some* time by the interval's end. (Notice, this property is satisfied even if a former leader in a neighboring region had m in its *broadcast* queue a long time before i added m to its queue. Informally, it says that for each neighboring region r', either i gets m to the current leader of r', or this leader, or a former leader in r', has already received m.)

Below we describe the modified broadcast sub-protocol of RMMB.

The Mobile Broadcast Sub-Protocol.
In the mobile broadcast sub-protocol each process i behaves the same as in the broadcast sub-protocol of RMMB with the following mobility-inspired modifications. Process i keeps the *broadcast* queue sorted on the timestamps added to each message by mobile collect. If i is a leader then becomes a non-leader, it empties the contents of its *broadcast* queue into the *arrive* queue used by its collect sub-protocol. If i is a non-leader and then becomes a leader, it empties its *arrive* into its *broadcast*. In both cases it sorts the resulting non-empty queue as usual. Process i can deliver any broadcast message it receives that it has not yet delivered.

Property (a) is satisfied for some $f_{col}(q) = \Theta(q)$, as each message on a *broadcast* queue requires only a constant amount of time to be sent from a leader to its neighboring leaders.

7.5 The Mobile RMMB Protocol

We combine the sub-protocols described above to generate the mobile RMMB protocol. Below we prove a preliminary theorem that proves RMMB solves the MMB problem under a certain set of constraints on the rate of *arrive* events and the mobility of nodes.

In the following, 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. We also reference the t_{CF} time bound defined in our discussion of the mobile leader election sub-protocol. In the following, we use the constant D to refer to the maximum diameter over all G and all executions of the network.

We begin with a key lemma which we use to prove the more general theorem that follows.

Lemma 7.1. *Let k be a positive integer and X be a nonnegative real. Let α be a MMB-well-formed execution of the mobile RMMB protocol, composed with four MAC automata and a regionalized $(2X + max\{X, t_{CF}\})$-stable network, with $X + t_{CF}$ passed as the parameter to the mobile leader election sub-protocol.*
Assume an $arrive(m)_i$ event occurs in α at time t, there are at most k messages with timestamps smaller than m in process arrive and broadcast queues over all states labelled with time t, and $X \geq kF_{ack}$. It follows that a $deliver(m)_j$ event occurs for all j by time $t + (D + 1)2X + X$.

Proof. We first establish the following three claims:

1. At all points in α, there exist no more than k messages ahead of m in any *broadcast* or *arrive* queue.

2. At all points in α, and every region r, there is a leader r that has at least X time left in the region.

3. If a message m arrives at a leader in region r at time t', then for every neighboring region r', at some time $t'' \leq t' + 2X$ a leader in r' receives m.

Claim 1 follows from our theorem assumption which states that at most k messages with smaller timestamps than m are in process queues in any state labelled with time t. Any messages that arrive after time t will have a later timestamp, and because the queues are sorted by timestamps, they will be placed behind m.

To prove the first part of claim 2—that there is always a leader—we defer to property (b) of the leader election sub-protocol. To prove the second part—that this leader has at least X time left—fix some leader i in some region r at some point in α. Consider the last leader election TDMA slot for r. Let t' be the time this election began. We know i won this election. There are two cases for i's victory.

In the first case, i had its high order lid bit set to 1. By the definition of the leader election protocol, and the parameter specified in the theorem statement, i keeps this bit at 1 only if it had an exit bound value of at least $t' + t_{CF} + X$ at the beginning of the slot.

In the second case, no process executed FRLE with the high order bit set to 1. Here, FRLE favors the process with the highest exit bound at the beginning of the slot. By assumption, at least one process has a bound of $t' + (2X + max\{X, t_{CF}\}) > t' + t_{CF} + X$, at this point. Therefore, if i won, its bound was at least this large. In both cases, because no more than t_{CF} time has elapsed since the beginning of the last election, process i has at least X time left in the region.

To prove claim 3, consider a leader i that receives m in r for the first time at time t'. If i remains leader for the next X time, this is sufficient for i to clear its *broadcast* queue and successfully broadcast m. (Recall: $X = kF_{ack}$, and by our first claim, at most k messages can get ahead of m in the queue.) By our second claim, there is always a leader in every neighboring region, so there will be some leader process in each region to receive m when the broadcast occurs.

On the other hand, if i loses its leadership at some time $t'' \geq t'$, before broadcasting m, its *broadcast* queue transforms into an *arrive* queue. The new leader in the region, which we label j, has an exit bound of at least $t'' + (2X + max\{X, t_{CF}\})$. By our second claim, we know i will remain in this region for at least X more time after losing its leadership—enough to finish clearing its queue and get m to j. Leader j still has more than X time as leader at this point, enough time to clear its *broadcast* queue and broadcast m. The total time for i to get m to j is X, and the total time for j to broadcast m is also bounded by X, providing the needed $2X$.

With our claims established, we can now finish the proof. Consider the $arrive(m)_i$ event from the lemma statement. Within X time the message m arrives at a leader's *broadcast* queue. We then repeatedly apply the third claim from above to show that the message arrives at, and is subsequently broadcast by, a leader in every region within $(D+1)2X$ additional time.

Now consider some arbitrary process j and the interval of length less than or equal to $X + (D + 1)2X$ between the $arrive(m)$ event and the message being received and broadcast by every leader. During every point of this interval, label each region r as *done* if and only if m has been received by a leader in r either from a neighboring leader or a process in r. Consider the first point at which j inhabits a region labelled *done*. There are two cases. The first case is that j is in a region r that transforms from not done to done. Here, m is coming from a neighbor or a process in r, meaning that j will also receive (and subsequently deliver) message m. The second case is that j moves from a region r' that is not done into a region r that is done. At this point, the leader in r has not yet broadcast m (if it had, we would be in the first case with respect to r'). By our stability assumption, j remains in r for longer than the $2X$ upper bound on the time required for the leader in r to broadcast. When this broadcast occurs, j will receive (and subsequently) deliver message m. □

We conclude with a theorem capturing a particular restriction on *arrive*-rates that ensures mobile RMMB solves the MMB problem:

Theorem 7.2. *Let k be a positive integer, F_{ack}^{max} and t_{CF}^{max} be nonnegative reals, and $T = (D + 1)2kF_{ack}^{max} + kF_{ack}^{max}$. 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 $(2kF_{ack}^{max} + max\{kF_{ack}^{max}, t_{CF}^{max}\})$-stable networks with $F_{ack} \leq F_{ack}^{max}$, and $t_{CF} \leq t_{CF}^{max}$, then the mobile RMMB protocol, executed with $kF_{ack}^{max} + t_{CF}^{max}$ passed as the parameter to the mobile leader election sub-protocol, solves the MMB problem.*

Proof. We can show by induction on message arrival events that for each $arrive(m)$ we satisfy the queue size constraint needed by Lemma 7.1—thus ensuring the delivery of m to all processes.

In more detail, order the *arrive* events by their timestamps. We can describe them as an ordered sequence $e_1, e_2, \ldots$. For each e_i, let $m(e_i)$ be the message associated with the i^{th} *arrive* event in this order, let $t(e_i)$ be the time at which this *arrive* occurred, and let $q(e_i)$ be the number of messages with smaller timestamps than $m(e_i)$ in queues in any state labelled with time $t(e_i)$.

Our inductive hypothesis for a given $i > 0$ states that for all $j, 1 \leq j \leq i, q(e_j) \leq k$. (Notice, this tells us that each satisfies the constraints of Lemma 7.1 for this k and will therefore be delivered to all processes, and thus cleared out of all queues, in T time.)

To prove the inductive step for $i + 1$, given the hypothesis holds for i, we first note that there are no more than $k - 1$ events e_j such that $j \leq i$ and $t(e_j) \geq t(e_{i+1}) - T$. If k such events existed we violate the theorem assumption (this would be an interval of size T with $k + 1$ *arrive* events.) However, for events that took place more than T time before e_{i+1} by our hypothesis and Lemma 7.1 they would have delivered and cleared out of the system before e_{i+1}. It follows that $q(e_{i+1}) \leq k$, as needed.

Finally, we note that the base case is trivial for $i \leq k$. □

8 Future Work

The abstract MAC layer approach to studying radio network algorithms generates a variety of interesting open questions and future work, for example, the study of additional basic communication primitives such as neighbor discovery and unicast communication, and the exploration of more advanced protocols, such as the construction of spanning trees or dominating sets without the use of location information. It would also be interesting to study additional questions and extensions to the MMB problem, such as the formulation of general throughput bounds or calculating the costs of adding sender acknowledgements.

Improvements to the formalism itself provide another important area of study. An obvious modification is to replace the deterministic delay functions with probability distributions, allowing for more advanced analysis of the system's probabilistic behavior. Another direction to investigate is adding edge weights to the communication and interference graphs, allowing for more subtle distinctions in the definition of interference. Such improvements will likely prove necessary for the successful modeling of basic radio network models such as those that make use of signal to interference-plus-noise ratios (SINR).

It is also important to study definitions of the abstract MAC layer that allow some properties to fail. For example, consider a natural variant of the model that sometimes generates acknowledgements even though some neighbor(s) did not receive the message. Can we design protocols that degrade gracefully under such failures—perhaps always maintaining safety and relying on the correct acknowledgements only for liveness?

In addition, this new model introduces new questions concerning fundamental limitations: what can and cannot be solved using an abstract MAC layer, and under what conditions? Such lower bounds can concern both high-level problems—e.g., is BMMB an optimal MMB solution for a static network with no location information—and low-level distributed computation—e.g., is it possible to break symmetry if the amount of possible contention, and therefore the amount of potential delay before an acknowledgement, is unknown?

Finally, it will prove useful to analyze specific MAC layer strategies for specific radio network models, providing concrete definitions for the delay functions. This work can span from the formalization of existing strategies, like the *decay* approach described in Section 2.3, to novel strategies such as those based on network coding.

9 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 *STOC '89: Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 274–285. ACM Press, 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 *SODA '00: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 861–870, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

[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 *Proc. of FOCS*, October 2003.

[9] B. Das and V. Bharghavan. Routing in ad-hoc networks using minimum connected dominating sets. In *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 *The Proceedings of the ACM SIGCOMM Conference*, 2008.

[12] O. Goussevskaia, T. Moscibroda, and R. Wattenhofer. Local broadcasting in the physical interference model. In *Joint Workshop on Foundations of Mobile Computing (DIALM)*, 2008.

[13] P. Gupta and P. R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, IT-46(2):388–404, 2000.

[14] T. Jurdzinski and G. Stachowiak. Probabilistic algorithms for the wakeup problem in single-hop radio networks. In *Proceedings of the 13th International Symposium on Algorithms and Computation*, 2002.

[15] D. Kowalski and A. Pelc. Broadcasting in undirected ad hoc radio networks. In *The Proceedings of the International Symposium on Principles of Distributed Computing*, pages 73–82, Boston, 2003.

[16] D. Kowalski and A. Pelc. Time of radio broadcasting: Adaptiveness vs. obliviousness and randomization vs. determinism. In *In Proceedings of the 10th Colloquium on Structural Information and Communication Complexity*, 2003.

[17] D. Kowalski and A. Pelc. Time of deterministic broadcasting in radio networks with local knowledge. *SIAM Journal on Computing*, 33(4):870–891, 2004.

[18] D. R. Kowalski and A. Pelc. Deterministic broadcasting time in radio networks of unknown topology. In *Foundations of Computer Science*, pages 63–72, 2002.

[19] E. Kranakis, D. Krizanc, and A. Pelc. Fault-tolerant broadcasting in radio networks. In *The Proceedings of the Annual European Symposium on Algorithms*, pages 283–294, 1998.

[20] F. Kuhn, T. Moscibroda, T. Nieberg, and R. Wattenhofer. Fast deterministic distributed maximal independent set computation on growth-bounded graphs. In *DISC*, 2005.

[21] F. Kuhn, T. Moscibroda, and R. Wattenhofer. Initializing newly deployed ad hoc and sensor networks. In *the 10th annual international conference on Mobile computing and networking*, 2004.

[22] F. Kuhn, T. Moscibroda, and R. Wattenhofer. Fault-tolerant clustering in ad hoc and sensor networks. In *The IEEE International Conference on Distributed Computing Systems*, 2006.

[23] T. Moscibroda and R. Wattenhofer. Maximal independent sets in radio networks. In *PODC*, 2005.

[24] T. Moscibroda and R. Wattenhofer. The complexity of connectivity in wireless networks. In *INFOCOM*, 2006.

[25] 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 *The 9th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2008.

[26] 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.