Reliable Neighbor Discovery for Mobile Ad Hoc Networks

Alejandro Cornejo
CSAIL
Massachusetts Institute of Technology
Cambridge MA 02139 USA
acornejo@csail.mit.edu

Saira Viqar
CSE Department
Texas A&M University
College Station TX 77843 USA
viqar@cse.tamu.edu

Jennifer L. Welch
CSE Department
Texas A&M University
College Station TX 77843 USA
welch@cse.tamu.edu

ABSTRACT

We define a reliable neighbor discovery layer for mobile ad-hoc networks and present two algorithms that implement this layer as a service with varying progress guarantees. Our algorithms are implemented atop an abstract MAC layer [13], which deals with the lower level details of collision detection and contention. Specifically, we first describe a basic region-based neighbor discovery protocol with weak progress guarantees. Informally speaking, this protocol does not guarantee communication links when nodes move quickly across region boundaries. To overcome this limitation, we describe a technique that uses a basic neighbor discovery protocol as a black box and boosts its progress guarantees. The key idea behind this technique is to use multiple partitions overlayed in a specific way, and associate with each partition an instance of the basic neighbor discovery protocol. We show the output of these instances can be composed in a way that provides stronger progress guarantees. In the last section of the paper we discuss different user layer algorithms which would benefit from the reliable neighbor discovery layer described in this paper.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms

Algorithms, Theory

Keywords

Mobile Ad Hoc Networks, Neighbor Discovery

1. INTRODUCTION

In mobile ad hoc networks (MANETs), the underlying communication graph changes over time. In this setting, it is not obvious how to define the *neighbor set* of a node in a way which is useful for user layer algorithms. For example, if two nodes are within communication range at a time instant, should they be considered neighbors even if they will not remain in communication range for enough time to exchange a message?

This paper defines a reliable neighbor discovery layer which establishes links over which message delivery is guaranteed. We present two algorithms that implement such a layer with varying progress properties.

These algorithms are implemented on top of a Medium Access Control (MAC) Layer which provides upper bounds on the time for message delivery thereby abstracting away the lower level details of collision detection, contention and scheduling. We follow the specification of an abstract MAC layer presented in [13] (with implementation details provided in [12]). This modular approach makes the algorithm easier to design, understand and verify. However, dealing with arbitrary mobility patterns while trying to maximize the time that links remain up, is still non-trivial. A performance comparison of this modular approach versus an approach where the neighbor discovery layer and MAC layer are merged is still an open problem.

We first implement a basic region-based neighbor discovery protocol which relies on sending notification messages when nodes enter and exit regions to set up the communication links. The main challenge is figuring out when messages need to be sent to guarantee they reach their intended destination despite the continuous motion of the nodes. However, this basic neighbor discovery protocol does not guarantee communication links when nodes are moving quickly across region boundaries. To this end, we use a technique that overlays multiple region partitions, associating with each region partition a basic neighbor discovery protocol instance. The output of each instance is then composed in a way which provides stronger progress guarantees.

Motivation. A neighbor discovery service which provides reliability guarantees is required for many user level algorithms. For example, the leader election algorithm of [10], the token circulation algorithm of [15], and the mutual exclusion algorithm of [22], all require an underlying neighbor discovery service. All these algorithms are important primitives in distributed computing. In addition to these, even the most basic of tasks in mobile ad-hoc networks, such as routing [2, 17, 16] or broadcasting [3, 18, 1], require accurate and up-to-date knowledge about neighbor nodes. For example, [19] implements coordinate based routing by assuming nodes know the location of their two-hop neighbors. Similarly, [16] describes a routing algorithm for multi-hop

DIALM-POMC'10, September 16, 2010, Cambridge, MA, USA.

wireless network that assumes one-hop neighbor information.

Contributions. The main contributions of this paper are: 1.) We describe a specification for a reliable neighbor discovery layer. We consider two different progress conditions. 2.) We present a basic region-based neighbor discovery protocol for MANETs which meets the above specification with the weaker progress guarantee. 3.) We describe a technique to boost the progress guarantees of a neighbor discovery protocol using overlayed region partitions. 4.) We discuss how the reliable neighbor discovery service can be used with existing user layer algorithms for MANETs.

Related Work. There has been a lot of previous work related to neighbor discovery. For example in hello protocols [2], nodes transmit periodic hello messages to discover neighbors. The set of neighbors is updated to reflect the information received in the hello message. If a hello message is not received from a neighbor for too long a time then it is discarded from the neighbor set. However, these approaches provide no formal guarantees and require sending messages periodically. In contrast, in our approach the number of messages sent depends on the frequency with which nodes cross region boundaries. Therefore, for example, if two nodes remain in the same regions forever, they need not exchange additional messages to maintain the status of the link between them.

Much previous work focuses on static networks. For example, in [4] a deterministic algorithm for computing two-hop neighbors in static networks is presented. In [14] a technique is presented for secure neighbor discovery for static networks. Similarly, [11] presents a deterministic protocol for neighbor discovery in static cognitive radio networks. Lastly, [20] considers neighbor discovery in static networks with directional antennas.

A topology discovery algorithm for mobile nodes is given in [5]; however, it is assumed that few nodes move at any given time and the speed of movement is very limited.

An asynchronous neighbor discovery and rendezvous protocol is presented in [8]. However, the focus of this protocol is to allow the nodes to operate at low duty cycles. Also, the protocol only caters to a rendezvous between just two nodes. An energy-efficient algorithm for node discovery is also presented in [9]. However, the emphasis in that work is on detecting the temporal patterns of node arrivals and scheduling a wake-up based on expected hourly activity.

In [21] the authors focus on maintaining neighbor knowledge in mobile nodes; however, they do not address the problem of nodes discovering neighbors at system start-up. An algorithm for neighbor discovery similar to ours, but with weaker progress guarantees, is presented in [6]. Specifically, a pair of nodes need to remain in the same region in order to set up a communication link. Although this is useful when all communication occurs between nodes in the same region, it cannot be used in more general settings. Even if all nodes are static and very close to each other, if they are dispersed across regions, the resulting neighbor graph will always be disconnected.

2. SYSTEM MODEL

The Timed I/O Automata (TIOA) modeling formalism [7] is used to model the mobile ad hoc network (MANET). We consider a system with n nodes (or users) which are executing in a MANET and communicate using local broadcast.

We use R to denote the physical space in which the nodes reside, also referred to as the deployment space. We assume R to be a closed, bounded and connected subset of $\mathbb{R}^2$. We assume all nodes agree on some partition of the deployment space into regions. This region partition is defined as follows:

Definition 1. *Let $\mathcal{U}$ be the index set for regions in the deployment space. A region partition divides R into a set of regions $\{R_u\}_{u\in U}$ such that: 1) For each $u \in \mathcal{U}$, R_u is a closed and connected subset of R. 2) For any $u, v \in U$, R_u and R_v may overlap only at their boundaries. 3) Each point in R must occur in at least one region. If any pair of regions R_u and R_v have a nonempty intersection, we say they are neighboring regions.*

We refer to the graph induced by the neighborhood relation of the region partition scheme as the *region graph*. We say region R_i and region R_j (or a node a in region R_i and a node b in region R_j) are *ℓ hops apart* if the shortest path between R_i and R_j in the region graph is of length ℓ.

We assume that nodes have access to their current location, which can be achieved, for example, through GPS etc. Note that this is not an unrealistic assumption for VANETs (Vehicular Ad Hoc Networks). There is a trajectory function for each node which specifies the motion of the node by giving its location at an instant of time. We assume that a node's trajectory function is known to that node with enough anticipation to communicate with other nodes before leaving or entering a region. Since in real deployments the speed of motion is much slower compared to the communication speed, this is not a limiting assumption for MANETs where mobility is controlled by a motion planner. Also in vehicular ad hoc networks (VANETs) where the motion is not directly controlled by a motion planner, the movement is not erratic and it is usually slow enough (compared to the communication speed) to be reliably predicted.

We now describe the five components in the system: the *network layer*, the *abstract MAC layer*, the *MAC Broker layer*, the *neighbor discovery layer*, and the *user layer* (cf. Figure 1).

2.1 The Network Layer

The network layer captures the physical behavior of the network. We assume that it provides other system components with location and time information.

We use G_{comm} to denote the directed graph whose vertices are the nodes and whose directed edges indicate which nodes are within the communication range of which other nodes. Similarly, G_{interf} denotes the directed graph whose vertices are the nodes and whose directed edges indicate which nodes are within the interference range of which other nodes. Since the communication and interference graphs can change dynamically over time during the execution, we can view G_{comm} and G_{interf} as mappings from network states to directed graphs.

Note that two nodes may have different broadcast and interference ranges. Let r_{min} be the minimum broadcast radius among all the nodes. For the region partition in use, we assume there exists a fixed parameter k such that any two points which are k hops apart in the region graph, are at distance at most r_{min}. This in turn implies that when two nodes are in regions separated by at most k hops, they are within communication range.

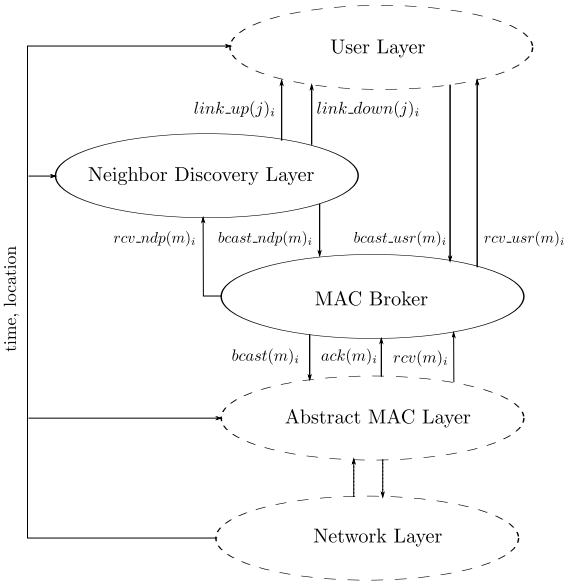


Figure 1: MANET system block diagram.

2.2 The Abstract MAC Layer

The abstract MAC layer provides reliable local broadcast with timing guarantees. It also provides acknowledgment that a message has been delivered with success to all nodes in the local neighborhood. This is done through interface actions $bcast(m)_i$, $ack(m)_i$, and $rcv(m)_i$ [1]. There is a guaranteed upper bound on the worst-case time for message delivery to nearby recipients given by F^+_{rcv}. Similarly, F^+_{ack} gives the upper bound on the total time for the sender to get an acknowledgment. These time bounds are functions of the level of contention. We assume that they are constant and available to algorithms implemented on top of the abstract MAC layer. Note that this implies that the dynamic communication graph (G_{comm}) induced by the motion of the nodes has a constant upper bound on the maximum degree. The cost of implementing this abstract MAC layer exactly as described in [13] might be prohibitively large. However, it is possible to provide similar guarantees with a high probability (cf. [12]).

The MAC layer assumes well-formedness conditions for upper layers. In particular, it assumes that a user process does not submit a *bcast* until after its previous *bcast* has had a matching *ack* returned. There are also constraints on message behavior. In particular, if a $bcast(m)_i$ event causes a $rcv(m)_j$ event, then at some point between these events nodes i and j have to be within interference range. If a $bcast(m)_i$ event causes an $ack(m)_i$ event and for every point in between these two event nodes i and j are in communication range, then a $rcv(m)_j$ caused by the *bcast* is guaranteed to precede the *ack*. Additionally, there are no duplicate receives or acknowledgments, and no receives after acknowledgments. Finally, every $bcast(m)_i$ causes an $ack(m)_i$.

2.3 The MAC Broker Layer

This layer enforces the well-formedness property required by the MAC layer. The layer guarantees the following three properties: 1) Well-formedness: A message is not broadcast through the abstract MAC layer before the *ack* of the preceding messages has been received. 2) Priority: User messages are only sent when there is no pending neighbor discovery message. 3) Routing: Received messages are routed correctly to either the neighbor discovery automaton or the user layer automaton.

To prioritize, the messages received through $bcast\_usr(m)_i$ and $bcast\_ndp(m)_i$ are pushed into different queues. Whenever the $bcast(m)_i$ action is triggered, a user message is only routed when the neighbor discovery message queue is empty. We assume that neighbor discovery messages are infrequent as compared to user messages. Hence, there is no starvation of the user messages caused by too many neighbor discovery messages. To limit the bandwidth requested by the user layer (and prevent starvation), we impose the restriction that the size of both message queues should not exceed some constant q. It is assumed that the user layer respects this restriction.

To route the messages an identifier is attached to the message before pushing it in the queue. This is then removed when a message is received with $rcv(m)_i$, and is used to trigger either a $rcv\_usr(m')_i$ or a $rcv\_ndp(m')_i$ action. The pseudo-code for the Mac Broker layer has been omitted due to lack of space.

2.4 The Reliable Neighbor Discovery Layer

The reliable neighbor discovery layer automaton for node i has four actions, $bcast\_ndp(m)_i$, $rcv\_ndp(m)_i$, $link\_up(j)_i$ and $link\_down(j)_i$ where $j \neq i$. The first two are used to broadcast and receive messages through the MAC broker. The $link\_up(j)_i$ action signals the user that a *reliable* communication link has been established between node i and j *from the perspective of node* i. Similarly the $link\_down(j)_i$ action signals the user that a previously established communication link between node i and j is down *from the perspective of node* i.

Definition 2 (Well-Formedness). *At a node i, for any j, the actions $link\_up(j)_i$ and $link\_down(j)_i$ alternate.*

Let $action^j_i(t) \in \{link\_up(j)_i, link\_down(j)_i\}$ be the most recent action involving node j observed by node i at time t. If $action^j_i(t) = link\_up(j)_i$ then we say link (i,j) is Up at time t, otherwise we say link (i,j) is Dn at time t.

To avoid unhelpful solutions where all links remain Dn independent of the environment we define a progress condition.

Definition 3 (Weak Progress). *There exist constants $a, b \in \mathbb{R}^+$, such that for all times t_1 and t_2 where $t_2 \geq t_1 + a + b$, and for any nodes i and j: if i is in region R_i and j is in region R_j throughout $[t_1, t_2]$, where R_i and R_j are at most k hops apart, the links (i,j) and (j,i) are Up during the time interval $[t_1 + a, t_2 - b]$.*

The previous progress definition has some limitations (which are discussed in detail in section 3). Hence we define the following (stronger) progress condition which does not require nodes to stay in the same region throughout the time interval; instead they only need to stay close enough to each other throughout the time interval.

[1] We present a slight simplification of the MAC layer specification [13] by ignoring the $abort(m)_i$ functionality

Definition 4 (Uniform Progress). *There exist constants $a, b \in \mathbb{R}^+$ such that for all times t_1 and t_2 where $t_2 \geq t_1 + a + b$, and for any nodes i and j: if at every time $t \in [t_1, t_2]$ nodes i and j remain at most k hops apart, the links (i, j) and (j, i) are Up during the time interval $[t_1 + a, t_2 - b]$.*

We introduce a validity condition to avoid unhelpful solutions where all links are kept in the Up state independent of the environment.

Definition 5 (Validity). *If (i, j) is Up at time t, then nodes i and j are in regions which are at most k hops apart at time t (and thus they are within distance r_{min}).*

Finally, we add a condition to guarantee reliable message delivery between neighboring nodes.

Definition 6 (Reliability). *If node i broadcasts a message at time t, and the link (i, j) is Up, the message is delivered to j exactly once. Also if a message is delivered to node j, then it was previously sent by some node.*

2.5 The User Layer

The user layer automaton is a composition of separate (and non-interacting) automata for the users $\{1, \ldots, n\}$. User i learns about the state of its neighbors through the *link_up* and *link_down* output actions of the neighbor discovery automaton. To broadcast and receive messages it communicates with the MAC broker automaton.

3. BASIC NEIGHBOR DISCOVERY PROTOCOL

Here we describe the basic neighbor discovery protocol, which satisfies the reliable neighbor discovery layer specifications with weak progress (the detailed TIOA code is presented in the appendix). At a high-level, the protocol relies on nodes sending notification messages tagged with their ids, whenever nodes are about to change regions.

When a node i is about to exit a region, it broadcasts a *leave* message some time before leaving. This *leave* message includes the region i will be moving into, or *null* if i will not be in the next region sufficiently long to establish a link. Using the information received in the *leave* message, i's neighbors determine if they should begin tearing down the corresponding link with i.

When a node i enters a new region and determines that it is going to remain there for sufficiently long, it broadcasts a *join* message. The recipients of the *join* message may start setting up the corresponding link to i if they have not already done so. The *join* message also serves as a request to learn the ids of the recipients. Specifically, when a node receives a *join* message from i, it first checks if it is going to remain in its current region for sufficiently long, in which case it responds with a *join_reply* message. The timing of these messages ensures that the proper semantics of the corresponding links are maintained. This means that the overhead for setting up and tearing down links is taken into account, and reliable message delivery is guaranteed when a link is in the Up state.

Suppose the time overhead for setting up a link between two neighbors is given by δ_{LU}, and the time overhead for tearing down a link is given by δ_{LD}. A node broadcasts a

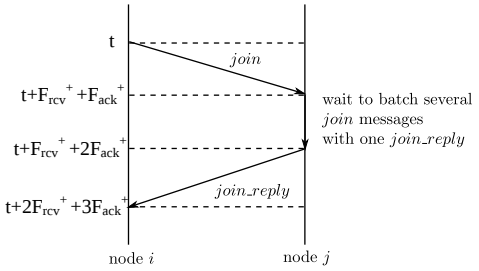


Figure 2: Time required for setting up a link.

join message upon entering a new region only if it is going to remain there for at least the amount of time required to set up a link and to tear it down. Thus a node broadcasts a *join* message if it is going to remain in its new region for at least $\delta_{LU} + \delta_{LD} + L$ time in the future where $L \geq 0$ is an user-provided parameter. Similarly, a node should broadcast a *leave* message δ_{LD} time before leaving the region to make sure the link is destroyed before the nodes are (potentially) out of transmission range. A node sends a *join_reply* message in response to a *join* message if it will remain in its region for $\delta_{LU} + \delta_{LD}$ time, to allow sufficient time to set the link up and tear it down at both ends before either node leaves the region.

The exact time overhead for setting up a link (δ_{LU}) can be determined in terms of the delays provided by the underlying MAC layer. This is the time overhead incurred in sending the *join* message and getting back the corresponding *join_reply* message. Now we argue that $\delta_{LU} = 2F^+_{rcv} + 3F^+_{ack}$ (cf. Figure 2). Recall that all messages are sent through the MAC Broker, which could wait up to F^+_{ack} time to get the *ack* from the preceding message before sending a new message. Hence, from the time the *join* message is sent by the neighbor discovery protocol, it might take up to $F^+_{ack} + F^+_{rcv}$ time before it is received. When a receiver gets the *join* message, it will respond with a *join_reply* message to the sender if it determines both nodes will remain within k hops for sufficient time. However, to prevent the receiver from being swamped with pending *join* messages (each of which might require a *join_reply* message), *join_reply* messages are buffered in intervals of F^+_{ack} so that multiple *join* messages can be answered with a single *join_reply* message.

Consider the following scenario. Suppose that node i is present in region R_i of the network. Now suppose that $n-1$ nodes move into region R_i and send *join* messages. Node i will then have to send $n-1$ *join_replies*. This will result in overflow in the Neighbor Discovery Protocol message queue in the MAC Broker layer. Thus i waits for F^+_{ack} time and collects the *join* messages and responds with one *join_reply* message, which guarantees that no more than one message every F^+_{ack} units is sent by the Neighbor Discovery Protocol layer to the MAC Broker layer. As before, it may take up to $F^+_{ack} + F^+_{rcv}$ units of time from when the *join_reply* message gets sent to the MAC Broker to when it gets received.

The time overhead for tearing down a link (δ_{LD}) can similarly be determined. This time bound should be sufficient to allow the *leave* message to be received. Moreover, it should also allow any node which receives the *leave* message to deliver all messages which were previously sent to the originator of the *leave* message. Specifically $\delta_{LD} = 2F^+_{rcv} + (q+1)F^+_{ack}$ (cf. 3), where q is the size of the queue.

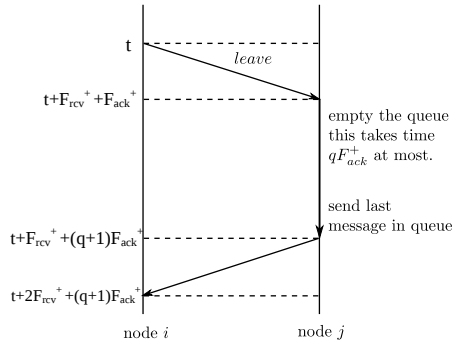


Figure 3: Time required for taking down a link.

As before, the first $F^+_{rcv} + F^+_{ack}$ time units allow the *leave* to be processed by the MAC Broker and to be delivered to its destination. Depending on the information received in a *leave* message a node may decide to tear down the link to the originator of the message. Regardless of this, the next qF^+_{ack} time units allow the MAC broker to send any messages which were queued before the *leave* message was received. (Recall that the maximum queue size is given by q and each message can incur a maximum delay of F^+_{ack} before it is sent.) The remaining F^+_{rcv} time units allow the last message of the queue to reach its destination.

Note that nodes that remain in regions for less than $\delta_{LU} + \delta_{LD}$ never establish links in the described protocol, since due to their motion across region boundaries they might not be able to receive messages reliably. In order to allow such nodes to maintain links we introduce in the next section the overlayed partition scheme. Here additional partitions are introduced to ensure that there are always some boundaries with respect to which nodes are stable enough to maintain links.

Correctness Proof. The basic neighbor discovery protocol described satisfies the well-formedness, weak progress, validity and reliability conditions defined in Section 2. Specifically, it satisfies the weak progress condition with constants $a = \delta_{LU}$ and $b = \delta_{LD} + L$.

PROOF **(Well-formedness)**. Consider nodes i and j. Observe that node j is only added to the neighbor set of node i with executing the $link\_up(j)_i$ action, and it is only removed when executing the $link\_down(j)_i$ action.

Suppose that node i performs a $link\_up(j)_i$, thereby adding node j to its neighbor set. Node i can only perform another $link\_up(j)_i$ if it receives a *join* or a *join_reply* message. In both cases it first checks if j is already in the neighbor set, and therefore it cannot perform two consecutive $link\_up(j)_i$ actions.

Now suppose that node i performs a $link\_down(j)_i$, thereby removing node j from its neighbor set. Node j can only perform another $link\_down(j)_i$ if it receives a *leave* message from node j or performs a *leave_region* action. For both cases it checks its neighbor set to see if j is present in it before doing a $link\_down(j)_i$, and therefore it cannot perform two consecutive $link\_down(j)_i$ actions. □

PROOF **(Weak Progress)** . Let $a = 2F^+_{rcv} + 3F^+_{ack} = \delta_{LU}$ and $b = 2F^+_{rcv} + (q+1)F^+_{ack} + L = \delta_{LD} + L$. Fix time t_1 and t_2 where $t_2 \geq t_1 + a + b$, and assume throughout the interval $[t_1, t_2]$ node i is in region R_i, node j is in region R_j, where R_i and R_j are at most k hops apart.

Let $t \leq t_1$ be the earliest time such that i and j are k hops apart throughout the interval $[t, t_1]$. At time t it follows that either node i entered region R_i, or node j entered region R_j (or both events happened). Without loss of generality, suppose i entered region R_i at time t. Then node i would have initiated the link establishment procedure by sending a *join* message at time t. Moreover, this procedure takes time $a = \delta_{LU}$ by construction, and hence starting at time $t + a \leq t_1 + a$ both (i, j) and (j, i) are Up.

The link tear down is not initiated by either endpoint until $b = \delta_{LD}$ time before leaving their respective regions, which by assumption, is no earlier than $t_2 - b$ (since $b = \delta_{LD}$). So (i, j) and (j, i) remain up until at least $t_2 - b$ and the theorem follows. □

PROOF **(Validity)** . Suppose by contradiction that at time t link (j, i) is Up, but node i and node j are more than k hops apart.

Since (j, i) is Up and since the algorithm drops messages from nodes which are more than k hops away, then at some time $t' < t$ node j received a message from node i, while they were within k hops of each other.

Moreover, since by assumption at time t they are more than k hops apart, let $t'' \in (t', t]$ be the first time after t' that one of them left a region as to become at most k hops apart at time t.

By construction, a *leave* message would have been sent by i, j or both, at time $t'' - \delta_{LD}$. If node j sent the leave message, then it would have set link (j, i) to Dn immediately and it would remain down until time t – a contradiction. Otherwise, node i sent the leave message, which would have been received at time $t'' - \delta_{LD} + F^+_{rcv} + F^+_{ack} < t''$ at which point they were still k hops apart, and therefore would have set the state of link (j, i) to Dn – a contradiction. □

PROOF **(Reliability)** . Suppose that link (i, j) is Up at time t and node i sends a message at time t. We will show this message is delivered by j. The fact that the message is delivered exactly once, and messages are only delivered if they were in fact generated by a node follows from the properties of the MAC layer.

Since (i, j) is Up at time t, then by the validity condition node i is in region R_i, and node j is in region R_j, where R_i and R_j are at most k hops apart. If throughout the interval $[t, t + \delta_{LD} - F^+_{rcv} - F^+_{ack}]$ nodes i and j remain at most k hops apart, then node i has sufficient time to empty its message queue and these messages will be successfully delivered to node j while it is still within communication range.

Hence, let $t' > t$ be the first time that node i and j become separated by more than k hops. This means at time t' either node i or node j left a region. If node i left a region then it sent a *leave* message at time $t' - \delta_{LD}$ and immediately set link (i, j) to Dn. Therefore $t' > t + \delta_{LD}$ and the theorem holds. Otherwise, node j left a region and sent a *leave* message at time $t' - \delta_{LD}$. This message was then processed by node i setting link (i, j) to Dn before time $t' - \delta_{LD} + F^+_{rcv} + F^+_{ack}$. Therefore $t' > t + \delta_{LD} - F^+_{rcv} - F^+_{ack}$ and the theorem holds. □

4. UNIFORM NEIGHBOR DISCOVERY

The weak progress condition for basic neighbor discovery requires nodes to be "stable" in the regions. In particular, nodes that are constantly changing regions won't have any neighbors, even if at every instant they remain arbitrarily

close to other nodes and move arbitrarily slowly. This limits the usefulness of the basic neighbor discovery protocol in settings where the nodes are in constant motion across boundaries.

However, the uniform progress condition does not suffer from these problems. Loosely speaking, uniform progress guarantees a communication link between any pair of nodes that remain sufficiently close together, without requiring the nodes to stay in the same regions. Instead of designing another protocol tailored to satisfy the uniform progress properties, we describe a construction that uses as a black-box any neighbor discovery protocol (that satisfies the weak progress condition) to build a uniform neighbor discovery service.

Let $\mathcal{U}$ be a region partitioning scheme. We define a region mapping function $X_{\mathcal{U}} : V \times \mathbb{R}^+ \to 2^{\mathcal{U}}$, that given a node $v \in V$ and a time $t \in \mathbb{R}^+$ returns the set of regions where node v was at time t. Except when a node is traveling through the border of neighboring regions, the mapping $X_{\mathcal{U}}$ returns a singleton set with one region. Node $v \in V$ is *jittering* in partition $\mathcal{U}$ at time t, if there exist constants $a, b \in \mathbb{R}^+$ and $t_1, t_2 \in [t-a, t+b]$ such that $X_{\mathcal{U}}(v, t_1) \cap X_{\mathcal{U}}(v, t_2) = \varnothing$. A node is *stable* in partition $\mathcal{U}$ at time t, if it is not jittering in $\mathcal{U}$ at time t.

Observe that by requiring the nodes to stay in fixed regions throughout a time interval, the weak progress condition implicitly required the nodes to be stable. In contrast, the uniform progress condition does not require nodes to be stable to guarantee a link. Instead, it allows nodes to keep changing regions as long as they remain k hops apart at every instant. To implement a protocol that satisfies uniform progress we will assume there exists some constant $c \in \mathbb{R}$ that bounds the maximum speed of the nodes. Since in practical deployments motion speed is bounded, and the communication speed is orders of magnitude faster than the physical speed of the nodes, we do not expect this assumption to be a limitation.

Figure 4 a) shows the trajectory of a node which is constantly changing regions, and thus jittering. This trajectory can be scaled down to obtain a trajectory where the node moves arbitrarily slowly while still jittering. Therefore it is clear that imposing a speed limit on the nodes does not solve the problem.

However, observe that in Figure 4 a) the node is jittering with respect to the grid partition defined by the solid lines, but if this partition were displaced to match the dashed lines, the same motion would become stable. We will exploit this observation by running several neighbor discovery algorithms in parallel using displaced region partitioning schemes and composing their results to guarantee uniform progress.

4.1 Stability and Partition Displacement

In this section we describe a set of displaced partitions that, assuming reasonable speed limits, guarantee a single node is stable in at least one partition. For ease of exposition we focus on regular grid partitions. In a regular grid tiling of the plane, the boundaries between different regions can be described by the union of two ordered sets of lines where: 1.) lines in the same set are parallel and uniformly spaced (the distance between neighboring lines is the same), and 2.) lines from different sets are perpendicular to each other. When a node crosses a region boundary of an axis-aligned grid partition, it will often be useful to make a distinction as to whether it crossed a horizontal or vertical boundary line.

Let U_0 be an axis-aligned grid partition where ℓ is the side length of a grid square. We consider a family of m regular grid partitions $\mathbf{U}_m = \{\mathcal{U}_0, \ldots, \mathcal{U}_{m-1}\}$. To simplify things further, let $\mathbf{U}_m$ be the family of m partitions where partition U_i is a copy of U_0 displaced by $i \cdot \frac{\ell}{m}$ in the x- and y-axis.

By definition, if a node is jittering in a partition at time t, then during the interval $[t-a, t+b]$ it crossed at least one region boundary of that partition. In particular this implies that during the interval $[t-a, t+b]$ either it crossed one horizontal boundary line or one vertical boundary line.

For the family of two partitions $\mathbf{U}_2$ any horizontal boundary line of partition U_0 intersects with any vertical boundary line of partition U_1 at a single point. Hence, for any speed limit $c > 0$, it is always possible to define a motion whose speed is bounded by c while jittering on both partitions (cf. Figure 4 b). Since any two non-parallel lines intersect at a single point, the same argument can be made for any set of two grid partitions, even when allowing arbitrary scaling, rotation and displacement of such partitions. This is summarized by the next theorem.

Theorem 1. *Consider any set of two regular grid partitions and any speed limit $c > 0$. There exists a motion that respects c while jittering in both partitions.*

However assuming a node respects a reasonable speed limit, three partitions are sufficient to guarantee the stability of a single node in a partition.

Lemma 1. *Consider any set of three axis-aligned grid partitions, where x is the minimum distance between two parallel line boundaries that belong to distinct partitions. If a node respects a speed limit of $x/(a+b)$ during the interval $[t-a, t+b]$, then it is stable in at least one partition at time t.*

Lemma 1 is tight, and one can show the following result.

Lemma 2. *Consider any set of three axis-aligned grid partitions, where x is the minimum distance between two parallel line boundaries that belong to distinct partitions. For any $\varepsilon > 0$ there exists a motion that respects a speed limit of $x/(a+b) + \varepsilon$ in the interval $[t-a, t+b]$ which jitters in all three partitions at time t.*

When considering sets of displaced identical grids, the distance between two parallel line boundaries is maximized by the set $\mathbf{U}_3$. As a corollary to Lemma 1, we have the following result (which is tight by Lemma 2).

Theorem 2. *If a node respects a speed limit of $\ell/(3(a+b))$ (where ℓ is the side length of a grid square) during the interval $[t-a, t+b]$, there exists a partition $\mathcal{U}_i \in \mathbf{U}_3$ (for which $x = \ell/3$) with respect to which the node is stable at time t.*

Proof. By definition, if a node jitters at time t in a partition, it must cross at least one region boundary of the partition during the interval $[t-a, t+b]$. Hence to jitter on all partitions at time t it must cross at least one region boundary of each partition in the interval $[t-a, t+b]$. At least two

of these boundaries are either vertical or horizontal boundary lines. In either case, the shortest distance between two parallel boundary lines that belong to different partitions is at least x.

Therefore, to jitter at time t the node must travel more than x during the interval $[t-a, t+b]$, which requires the node to attain a speed strictly larger than $x/(a+b)$. □

Figure 4 c) describes a jitter pattern for $\mathbf{U}_3$, which depicts the fact that it is necessary for the node to travel at least a distance of $\ell/3 = x$ to jitter in all three grids.

However, this result is not enough to guarantee uniform progress. The progress condition deals with edges (a pair of nodes) and not with single nodes. Moreover, it can be shown as a consequence of Theorem 1 that no set of three or four partitions is sufficient to guarantee a pair of nodes will be stable in the same partition.

Corollary 1. *Consider any set of four regular grid partitions and any speed limit $c > 0$. It is possible for two nodes to respect the speed limit of c, while not being stable in the same partition.*

Fortunately, it turns out that a set of five partitions are sufficient to guarantee that for any pair of nodes, there exists a partition in which they are both stable (assuming reasonable speed limits). First, we prove the following generalization of Lemma 1.

Lemma 3. *Consider any set of $m \geq 3$ axis-aligned grid partitions, where x is the minimum distance between two parallel line boundaries that belong to distinct partitions. If a node respects a speed limit of $x/(a+b)$ during the interval $[t-a, t+b]$, then it is stable in at least $m-2$ partitions at time t.*

Proof. Consider a node that respects a speed limit of $x/(a+b)$ during the interval $[t-a, t+b]$.

We proceed by induction on m. For $m = 3$ the base case holds by Lemma 1. Consider a set P of size $|P| = m$ of axis-aligned grid partitions, and let $P' \subset P$ be any subset of size $|P'| = m-1$. By inductive hypothesis the node is stable in a set $Q \subset P' \subset P$ of size $|Q| = m-3$ at time t. Moreover, since $|P \setminus Q| = 3$ then by Lemma 1 there exists at least one partition $\mathcal{U} \in P \setminus Q$ where the node is stable at time t. Therefore it follows the node is stable in the set $Q \cup \{U\}$ of size $m-2$, which concludes the proof. □

Therefore, if the nodes are sufficiently slow, the number of partitions where the nodes are guaranteed stability grows at the same rate as the number of partitions. Hence, it is not surprising that given enough axis-aligned grid partitions we can guarantee any pair of nodes that are sufficiently slow will be stable in a non-empty subset of partitions.

Lemma 4. *Consider any set P of size $|P| = m \geq 5$ of axis-aligned grid partitions, where x is the minimum distance between two parallel line boundaries that belong to distinct partitions. If two nodes respect a speed limit of $x/(a+b)$ during the interval $[t-a, t+b]$, there exists a subset $Q \subset P$ of size $|Q| \geq m-4$ where both nodes are stable at time t.*

Hence in particular $\mathbf{U}_5$ is enough to guarantee that any pair of nodes that respect a sufficiently low speed limit will be stable in at least one partition $\mathcal{U}_i \in \mathbf{U}_5$.

Theorem 3. *If two nodes respect a speed limit of $\ell/5(a+b)$ during the interval $[t-a, t+b]$, there exists a partition $\mathcal{U}_i \in \mathbf{U}_5$ (for which $x = \ell/5$) with respect to which both nodes are stable at time t.*

Proof. Consider nodes i and j that respect a speed limit of $x/(a+b)$ during the interval $[t-a, t+b]$.

Let P be any set of size $|P| = m \geq 5$ of axis-aligned grid partitions. By Lemma 3 at time t node i is stable in some subset $Q_i \subset P$ of size $|Q_i| = m-2$, and node j is stable in some subset $Q_j \subset P$ of size $|Q_j| = m-2$.

Hence, both nodes are stable at time t in the set $Q = Q_i \cap Q_j$. Moreover, by De Morgan's laws $Q_i \cap Q_j = (Q_i^c \cup Q_j^c)^c = (P \setminus Q_i \cup P \setminus Q_j)^c = P \setminus (P \setminus Q_i \cup P \setminus Q_j)$. Finally $|Q| = |P| - |P \setminus Q_i \cup P \setminus Q_j| \geq |P| - |P \setminus Q_i| - |P \setminus Q_j| = m-4$. □

As before, Theorem 3 is tight, since it is possible for two nodes to not be stable in any partition of $\mathbf{U}_5$ with speed limit of $\ell/5(a+b) + \varepsilon$ for any $\varepsilon > 0$. Therefore the family $\mathbf{U}_5$ is sufficient to guarantee that any pair of nodes will be stable in some partition, as long as their speed is bounded by $\ell/5(a+b)$. Moreover, we also argued that regardless of the speed limit imposed, no smaller family of regular grid partitions can guarantee the stability of both nodes in the same partition. We also showed that at least when considering identical partitions, no other arrangement of partitions has a less demanding speed bound.

4.2 Uniform Neighbor Discovery Protocol

Here we exploit the stability properties of the displaced partitions described in the previous subsection to construct a *uniform neighbor discovery protocol* (UNDP) which guarantees uniform progress. Specifically we execute an instance of a neighbor discovery protocol with the basic progress guarantees in each partition, and describe how to combine the outputs to guarantee uniform progress. Figure 5 shows the detailed interactions between different components.

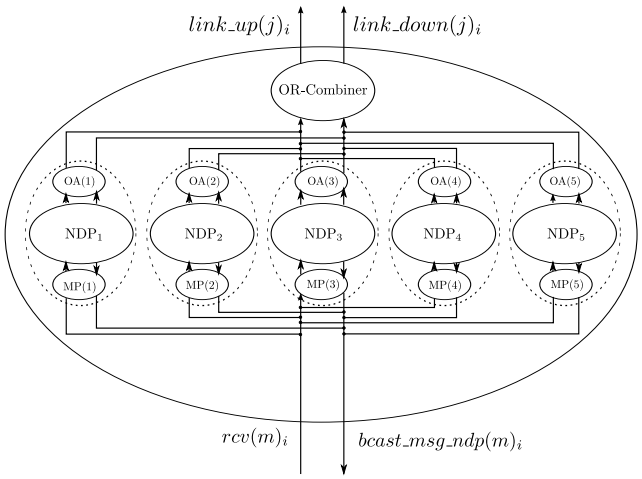
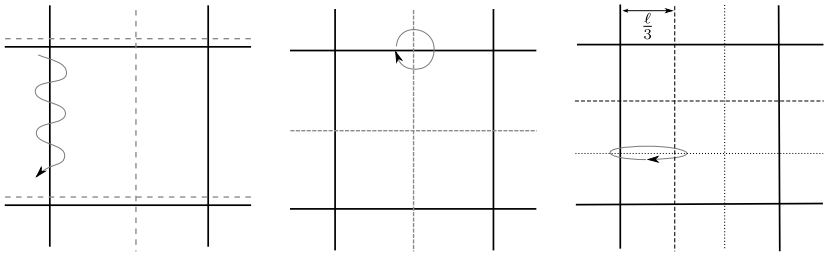


Figure 5: Uniform Neighbor Discovery block diagram.

In particular we consider the regular grid partition family $\mathbf{U}_5$, and in each partition $\mathcal{U}_i \in \mathbf{U}_5$ we execute an instance of of a neighbor discovery protocol NDP_i with basic progress guarantees. Letting k' be the parameter received by each NDP_i instance to determine the maximum hop distance between two neighbors, we set $k' = k+1$ where k will be the



(a) Jittering in a single partition (b) Jittering in $\mathbf{U}_2$. (c) Jittering in $\mathbf{U}_3$.

Figure 4: Jittering trajectories

maximum hop distance between two neighbors in the uniform neighbor discovery protocol. We will show later that this is required in order to guarantee uniform progress.

The neighbor discovery protocol black-box needs to be slightly augmented for it to run correctly in parallel. In a *message processing* (MP) phase an instance identifier is attached to every message sent and removed after routing the message to the correct instance. Similarly, an *output adapter* (OA) phase translates the $link\_up(j)$ and $link\_down(j)$ output actions of instance NDP_n to include an instance identifier (i.e. they are translated to the output actions $link\_up(j,n)$ and $link\_down(j,n)$ respectively).

Let ANDP_i represent the instances of NDP_i augmented to run in parallel. If we represent the state of each directed link as a boolean variable (where $Up = 1$ and $Dn = 0$), we want the state of a link in the uniform neighbor discovery to be the OR of the state of the basic neighbor discovery instances. Specifically, we use an OR-Combiner automaton that outputs a $link\_up(j)$ action the first time there exists **some** n such that ANDP_n considers node j a neighbor. Similarly, the OR-Combiner outputs a $link\_down(j)$ action the first time that for **every** n instance ANDP_n determines that node j should no longer be a neighbor.

It remains to prove that the UNDP algorithm described implements the reliable neighbor discovery layer with uniform progress. We start by showing that UNDP satisfies the well-formedness condition, which follows from the definition of the OR-Combiner.

Theorem 4. *UNDP satisfies the well-formedness condition.*

Proof. The OR-Combiner outputs $link\_up(j)$ if $j \notin S$ and immediately executes $S := S \cup \{j\}$. Similarly the OR-Combiner outputs $link\_down(j)$ if $j \in S$ and immediately executes $S := S \setminus \{j\}$. Moreover, no other action modifies the set S.

Hence it follows that $link\_up(j)$ and $link\_down(j)$ events alternate at each node. □

The fact that the uniform neighbor discovery protocol satisfies the validity and reliability properties follows from the fact the basic neighbor discovery protocol also satisfies these properties.

Theorem 5. *UNDP satisfies the validity and reliability conditions.*

Proof. By construction if the link (u, v) with respect to node u in UNDP at time t, then the link (u, v) is Up according to u in at least one neighbor discovery instance NDP_i at time t.

Since each neighbor discovery instance satisfies both the validity and safety conditions, then by the former the distance between u and v at time t is less than r_{min}, and by the latter a message sent by node u at time t is received by node v. □

To guarantee a communication link, the uniform progress condition stipulates the nodes should remain at most k hops apart during some time interval. Since the number of hops between two nodes is measured with respect to a partition, it seems pertinent to first define which of the partitions should be used to measure the hops between two nodes. However, this is a moot point since the guarantees hold as long as the nodes remain k hops apart with respect to any partition (the partition can be different for every time instant). To show this, we first need an auxiliary result concerning hop count in displaced grid partitions.

Lemma 5. *Consider a set of identical but displaced grid partitions and a pair of points. The hop distance between the two points varies by at most one hop between partitions.*

Proof. Fix two grid partitions A and B with side length ℓ, and two points p and q.

We decompose the hop distance and Euclidean distance between points in two components. Specifically, let $\|p-q\|_x^U$ be the number of vertical boundary lines between p and q in U, and $\|p-q\|_y^U$ be the number of horizontal boundary lines between p and q in U. Similarly, let $\|p-q\|_x$ and $\|p-q\|_y$ be the vertical and horizontal Euclidean distance between p and q.

By the neighbor structure of grid partitions it follows that $\|p-q\|^U = \max(\|p-q\|_x^U, \|p-q\|_y^U)$. Without loss of generality assume $\|p-q\|^B \geq \|p-q\|^A$, hence we need only to show $\|p-q\|^B \leq \|p-q\|^A + 1$.

Let $\|p-q\|_x^A = s$ and $\|p-q\|_y^A = t$. Hence the components of the Euclidean distance between p and q are bounded by $\|p-q\|_x \leq \ell(s+1)$ and $\|p-q\|_y \leq \ell(t+1)$. Moreover, regardless of the partition displacement, no more than $s+1$ vertical boundary lines (resp. $t+1$ horizontal boundary lines) need to be crossed to travel a distance of $\ell(s+1)$ (resp. $\ell(t+1)$). Hence, $\|p-q\|^B = \max(\|p-q\|_x^B, \|p-q\|_y^B) \leq \max(s+1, t+1)$
$= \max(s,t)+1 = \max(\|p-q\|_x^A, \|p-q\|_y^A)+1 = \|p-q\|^A+1$

□

Finally we can show that UNDP satisfies the stronger uniform progress property.

Theorem 6. *UNDP satisfies the uniform progress condition.*

PROOF. By assumption each instance NDP_i satisfies the basic progress condition with some fixed $a, b \in \mathbb{R}^+$ for nodes that are $k' = k + 1$ hops away in partition $\mathcal{U}_i$.

Consider two nodes u and v that respect a speed limit of $c = \ell/5(a + b)$ and remain k hops apart in at least one partition during the time interval $[t - a, t + b]$.

By Theorem 3 there exists at least one partition $\mathcal{U}_j \in \mathbf{U}_5$ where both u and v are stable. Since during the interval $[t - a, t + b]$ nodes u and v remained k hops apart in some partition, then by Lemma 5 they remained at most $k+1$ hops apart in all partitions. This is because the hop distance between two points varies by at most one hop between different partitions.

In particular, they also remained $k + 1 = k'$ hops apart in the partition $\mathcal{U}_i$ where they are both stable. Specifically, both u and v remained in regions $R_u \in \mathcal{U}_i$ and $R_v \in \mathcal{U}_i$ during the interval $[t - a, t + b]$, where R_u and R_v are $k + 1$ hops apart.

Since NDP_i satisfies basic progress, then the link (u, v) was Up with respect to both u and v at time t in the instance NDP_i. Finally, since the state of a link in UNDP is the OR of the state in all instances, the theorem follows. □

5. DISCUSSION

There is a collection of algorithms for mobile ad hoc networks which assume a neighbor discovery layer. In some cases, these algorithms can be used without modification with the reliable neighbor discovery service given in this paper.

For example, the work in [15] presents several algorithms to continually circulate a token through all nodes of a mobile ad hoc network. The circulating token can be used to provide total order for message delivery in group communication services. To obtain neighbor information, they rely on a hello protocol, and to ensure messages are eventually delivered they rely on TCP. Observe that neither of these mechanisms provides formal guarantees. However, these unmodified algorithms can be implemented on top of a reliable neighbor discovery service as the one presented here, which does provide formal guarantees.

A leader election algorithm for dynamic graphs is presented in [10], which assumes an underlying layer guarantees reliable neighbor discovery. The leader election algorithm guarantees that after topology changes cease, eventually there is a unique leader in every connected component. The algorithm is based on the temporally ordered routing algorithm (TORA) given in [16]. However, it is not immediate that our reliable neighbor discovery layer can be used, since the authors in [10] impose an extra synchronization condition as part of the neighbor discovery specification. Although this synchronization condition is not satisfied by our specification, it turns out this condition is not required for the algorithm to work. With little work, it is possible to show the same results hold when using the reliable neighbor discovery layer described in the original paper.

This does not mean that our neighbor discovery service can be used for all applications. For example [22] presents a mutual exclusion algorithm that assumes a neighbor discovery service which provides perfectly synchronized and timely information, and in fact, does not work with the service defined in this paper. Although our specification does not meet these requirements, it is unclear if such strong assumptions are justified in truly mobile networks.

As future work, we would like to extend the service presented to handle the presence of faulty nodes. Another interesting thread would be to compare the cost of this service against another which dispenses with the MAC layer by implementing directly on the physical layer.

6. REFERENCES

[1] Bar-Yehuda, R., Goldreich, O., Itai, A,: 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), pp. 104-126, (1992)

[2] Broch, J., Maltz, D.A., Johnson, D.B., Jetcheva, J.: A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols. In: ACM/IEEE Intl Conf. Mobile Computing and Networking, pp. 85-97, (1998)

[3] Bruschi, D., Pinto, M.D.: Lower Bounds for the Broadcast Problem in Mobile Radio Networks. Distributed Computing, 10(3), pp. 129-135, (1997)

[4] Calinescu, G.: Computing 2-Hop Neighborhoods in Ad Hoc Wireless Networks. Lecture Notes in Computer Science, Vol. 2865, pp. 175-186, (2003)

[5] Chandra, R., Fetzer, C., Hogstedt, K.: A Mesh-Based Robust Topology Discovery Algorithm for Hybrid Wireless Networks. In: Informatics, (2002)

[6] Cornejo, A., Lynch, N., Viqar, S., Welch, J.L.: Neighbor Discovery in Mobile Ad Hoc Networks Using an Abstract MAC Layer. In: Allerton'09, (2009)

[7] Kaynar, D., Lynch, N., Segala, R., Vaandrager, F.: The Theory of Timed I/O Automata. Synthesis Lectures on Computer Science. Morgan Claypool Publishers, (2006)

[8] Dutta, P., Culler, D.: Practical Asynchronous Neighbor Discovery and Rendezvous for Mobile Sensing Applications. In: SenSys'08, (2008)

[9] Dyo, V., Mascolo, C.: Efficient Node Discovery in Mobile Wireless Sensor Networks. In: DCOSS'08, (2008)

[10] Ingram, R., Shields, P., Walter, J., Welch, J.: An Asynchronous Leader Election Algorithm for Dynamic Networks. Technical Report, Department of Computer Science and Engineering, Texas A&M University, (2009)

[11] Krishnamurthy, S., Chandrasekaran, R., Mittal, N., Venkatesan, S.: Brief Announcement: Synchronous Distributed Algorithms for Node Discovery and Configuration in Multi-channel Cognitive Radio Networks. In: The Symposium on Distributed Computing (DISC), (2006)

[12] Khabbazian, M., Kowalski, D., Kuhn, F., Lynch, N.: The Cost of Global Broadcast using Abstract MAC Layers. Technical Report MIT-CSAIL-TR-2010-005, MIT CSAIL, (2010)

[13] Kuhn, F., Lynch, N., Newport, C.: The Abstract MAC Layer. In: DISC'09 23rd International Symposium on Distributed Computing, (2009)

[14] Liu, D.: Protecting Neighbor Discovery Against Node Compromises in Sensor Networks. In: ICDCS IEEE

International Conference on Distributed Computing Systems, (2009)

[15] Malpani, N., Chen, Y., Vaidya, N., Welch, J.: Distributed Token Circulation in Mobile Ad Hoc Networks. IEEE Transactions on Mobile Computing, Vol. 4, No. 2, (2005)

[16] Park, V., Corson, M.S.: A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks. In: INFOCOM'97, pp. 1405-1413, (1997)

[17] Perkins, C., Royer, E.: Ad-hoc On-Demand Distance Vector Routing. In; Workshop on Mobile Computing Systems and Applications (WMCSA'99), pp.90-100, (1999)

[18] Prakash, R., Schiper, A., Mohsin, M., Cavin, D., Sasson, Y.: A Lower Bound for Broadcasting in Mobile Ad Hoc Networks. EPFL Technical Report. IC/2004/37, (2004)

[19] Rao, A., Ratnasamy, S., Papadimitriou, C., Shenker, S., Stoica, I.: Geographic Routing without Location Information. In: Mobicom The Ninth Annual International Conference on Mobile Computing and Networking. pp. 96-108, (2003)

[20] Vasudevan, S., Kurose, J. Towsley, D.: On Neighbor Discovery in Wireless Networks with Directional Antennas. In: INFOCOM, (2005)

[21] Viqar, S., Welch J.L.: Deterministic Collision Free Communication Despite Continuous Motion. In: Algosensors'09, (2009)

[22] Walter, J., Welch, J., Vaidya, N.: A Mutual Exclusion Algorithm for Ad Hoc Mobile Networks. Wireless Networks, (2001)

APPENDIX: TIOA Code

We describe the algorithms using the TIOA formalism [7].

For simplicity we define the function $hops : \{R_u\}_{u\in U} \times \{R_u\}_{u\in U} \rightarrow \mathbb{N}$ that receives two regions and returns the number of hops between them. If one of the regions is null it returns ∞. Similarly, we define the function $getregion : \mathbb{R}^2 \rightarrow 2^{\mathcal{U}}$ that maps any point in the deployment space to a set of regions that contain such point. When queried in a region boundary it returns the set of regions that share the boundary, otherwise it returns a singleton set with the current region. For the neighbor discovery protocol we assume a *TIOA trajectory* that stops time whenever a precondition is enabled. However since formally there is no first time when a node enter or leaves a region (left-open intervals) we define a TIOA trajectory for the *enter_region* action as:

$$\exists u : Region\ (u \neq val(region)) \wedge$$
$$curreg \notin getregion(traj_{now}) \wedge u \in getregion(traj_{now}) \wedge$$
$$curreg \in getregion(traj_{now-\varepsilon}) \wedge u \in getregion(traj_{now-\varepsilon})$$

where $\varepsilon > 0$ is a small constant describing the slack, and depending on the motion of the agents with respect to the size of the regions. A similar predicate is assumed for the *leave_region* action.

Algorithm 1 Neighbor Discovery Protocol

```
automaton NDP(i:ℕ, traj:Traj, F+_ack:ℝ, L:ℝ, k:ℕ,δ_LU:ℝ,δ_LD:ℝ)
  states
    active : Bool := false;
    sendbuffer : Seq[M] := ∅;
    recvbuffer : Seq[M] := ∅;
    eventqueue : Seq[Ev] := ∅;
    S : Set[ℕ] := ∅;
    regs : Map[ℕ, Region] := empty;
    curreg : Null[Region] := nil;
    newreg : Null[Region] := nil;
    jointrigger : Real := −1;
    now : ℝ := 0;

  transitions
    output bcast(m, i)
      pre m = head(sendbuffer);
      eff sendbuffer := tail(sendbuffer);

    input rcv(m, i)
      eff recvbuffer := recvbuffer ⊢ m;

    internal enter_region(i)
      pre eventqueue = ∅ ∧ getregion(traj[now]) ≠ val(curreg);
      eff curreg := embed(getregion(traj[now]));
        if ∀t : ℝ(t ≥ now ∧ t ≤ now + δ_LU + L + δ_LD
            ⇒ getregion(traj[t]) = val(curreg)) then
          sendbuffer := sendbuffer ⊢ [[join, val(curreg), nil], i];
          active := true;

    internal leave_region(i)
      pre eventqueue = ∅ ∧ active
        ∧getregion(traj[now + δ_LD]) ≠ val(curreg);
      eff newreg := embed(getregion(traj[now + δ_LD]));
        active := false;
        if ∃t : ℝ(t ≥ now + δ_LD ∧
            t ≤ now + δ_LD + δ_LU + L + δ_LD
            ⇒ getregion(traj[t]) ≠ val(newreg)) then
          newreg := nil;
        sendbuffer := sendbuffer ⊢
              [[leave, val(curreg), newreg], i];
        for j : ℕ in S
          if hops(regs[j], val(newreg)) > k then
            eventqueue := eventqueue ⊢ [down, j, regs[j]];

    internal process_message(m, i)
      pre eventqueue = ∅ ∧ m = head(recvbuffer)
        ∧getregion(traj[now]) = val(curreg);
      eff recvbuffer := tail(recvbuffer);
        if m.sender ∈ S then
          regs := update(regs, m.sender, m.msg.reg);
        if hops(m.msg.reg, val(curreg)) ≤ k then
          if m.msg.type = join ∧ m.sender ∉ S∧
              (∀t : ℝ(t ≥ now ∧ t ≤ now + δ_LU + δ_LD
              ⇒ getregion(traj[t]) = val(curreg))) then
            if jointrigger = −1 then
              jointrigger := now + F+_ack;
            eventqueue := eventqueue ⊢ [up, m.sender, m.msg.reg];
          if m.msg.type = leave ∧ m.sender ∈ S∧
              hops(val(m.msg.dest), val(curreg)) > k then
            eventqueue := eventqueue ⊢
              [down, m.sender, regs[m.sender]];
          if m.msg.type = join_reply ∧ m.sender ∉ S
              ∧active then
            eventqueue := eventqueue ⊢ [up, m.sender, m.msg.reg];

    internal send_join_reply(i)
      pre eventqueue = ∅ ∧ jointrigger = now;
      eff jointrigger := −1;
        sendbuffer := sendbuffer ⊢
              [[join_reply, val(curreg), nil], i];

    output link_down(j, i)
      pre ∃reg : Region(head(eventqueue) = [down, j, reg]);
      eff S := S − j;
        regs := remove(regs, j);
        eventqueue := tail(eventqueue);

    output link_up(j, i)
      pre ∃reg : Region(head(eventqueue) = [up, j, reg]);
      eff S := S ∪ j;
        regs := update(regs, j, head(eventqueue).reg);
        eventqueue := tail(eventqueue);
```