COMMUNICATION ALGORITHMS FOR WIRELESS AD HOC NETWORKS

A Dissertation

by

SAIRA VIQAR

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

August 2012

Major Subject: Computer Science

COMMUNICATION ALGORITHMS FOR WIRELESS AD HOC NETWORKS

A Dissertation

by

SAIRA VIQAR

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

August 2012

Major Subject: Computer Science

ABSTRACT

Communication Algorithms for Wireless Ad Hoc Networks. (August 2012)

Saira Viqar, B.S, NUST Pakistan; M.S., NUCES Pakistan

Chair of Advisory Committee: Dr. Jennifer L. Welch

In this dissertation we present deterministic algorithms for reliable and efficient communication in ad hoc networks. In the first part of this dissertation we give a specification for a reliable neighbor discovery layer for mobile ad hoc networks. We present two different algorithms that implement this layer with varying progress guarantees. In the second part of this dissertation we give an algorithm which allows nodes in a mobile wireless ad hoc network to communicate reliably and at the same time maintain local neighborhood information. In the last part of this dissertation we look at the distributed trigger counting problem in the wireless ad hoc network setting. We present a deterministic algorithm for this problem which is communication efficient in terms of the the maximum number of messages received by any processor in the system.

DEDICATION

For my parents

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to Jennifer Welch, for being an excellent advisor and for providing support and guidance at each and every stage of my research. Her intelligence and insight into problems always motivated and inspired me in my work. I am forever indebted to her, not only for all the training and mentorship she provided, but also for her incredible kindness and patience.

A special thanks to Nancy Lynch for supporting my visits to MIT and for providing supervision and guidance. I also thank Alejandro Cornejo for his ideas and his participation in this research. My thanks to Jianer Chen, Andrew Jiang, and Alex Sprintson for taking the time to be on my committee and also for showing genuine interest in my work. I would also like to acknowledge the anonymous referees for their helpful comments.

I am indebted to my colleague and friend Hyun Chul Chung for his constant encouragement, and lively discussions, as well as his honest feedback and advice. I thank all the other members of my research team: Keerthi Deconda, Khushboo Bhatia, Gautam Roy, Erica Wang, Josef Widder, and Srikanth Sastry, for the interesting conversations and good times.

Lastly, I thank Zaid, Saba, Nasir, and Shahnawaz, for being my best friends, and my parents for all their love and also for all the sacrifices they have made to support me.

This work has been funded by the Fulbright scholarship, the IAP scholarship at Texas A&M, NSF grants 0500265 and 0964696, and Texas Higher Education Coordinating Board grants ARP 00512-0007-2006, and NHARP 00512-0130-2007.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

1. INTRODUCTION

One of the most interesting and important applications of distributed algorithms is in the field of wireless ad hoc networks. A wireless ad hoc network consists of autonomous computing nodes which communicate with each other through wireless transmissions. The nodes do not have access to a centralized communication infrastructure and are generally unaware of the network topology. The communication medium is shared and hence, only one node in a local neighborhood may transmit a message at a particular time. If multiple neighboring nodes send messages simultaneously then a collision might occur at the receiving node, disrupting the message transmission. Thus nodes have to coordinate their activities in a distributed fashion, in order to facilitate basic communication tasks such as propagating a single message throughout the network. In case the nodes are mobile, even keeping track of the local neighborhood topology is an ongoing process. Furthermore, nodes in wireless ad hoc networks are also resource-constrained, hence developing algorithms that are communication efficient is essential not only for reducing contention but also for conserving power.

In this dissertation we focus on developing deterministic algorithms for reliable and efficient communication in ad hoc networks. This work covers the following areas:

1. Reliable neighbor discovery with an abstract MAC layer
2. Neighbor knowledge and deterministic collision free communication.
3. Communication efficiency for distributed trigger counting.

This thesis follows the style of *Distributed Computing*.

The first problem we consider is keeping track of the local neighborhood topology as nodes continuously move in and out of each other's transmission and interference range. We assume that nodes are mobile but there is a bound on the maximum speed of the nodes. We focus on deterministic algorithms and use the simple unit disk graph (UDG) model. We consider the problem using two different approaches. The first is a modular approach in which the neighbor discovery problem is seen as a separate layer built upon the medium access layer (or MAC layer, which handles contention among the nodes so that messages from nodes may be received by neighbors without collisions). In the second approach the two layers are integrated together. The modular approach simplifies the design and verification of the algorithm, and allows more fine-grained analysis. However, in the integrated approach the neighbor discovery protocol may benefit from feed back from the medium access protocol and vice versa.

In the first part of this dissertation we define a reliable neighbor discovery layer for mobile ad hoc networks and present two algorithms (which have appeared in [15]), that implement this layer as a service with varying progress guarantees. Our algorithms are implemented atop an abstract MAC layer [36], which deals with the lower level details of collision detection and contention. We first describe a basic region-based neighbor discovery protocol with weak progress guarantees. 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.

The second part of this dissertation gives a solution where neighbor discovery and medium access are integrated into one layer. The algorithm (published in [48]), allows nodes in a mobile wireless ad hoc network to communicate reliably and at the same time maintain local neighborhood information. It is assumed that nodes are located on a two-dimensional plane and may be in continuous motion. In our solution we tile the plane with hexagons. Each hexagon is assigned a color from a finite set of colors. Two hexagons of the same color are located sufficiently far apart so that nodes in these two hexagons cannot interfere with each other's broadcasts. Based on this partitioning we develop a periodic deterministic schedule for mobile nodes to broadcast. This schedule guarantees collision avoidance. Broadcast slots are tied to geographic locations instead of nodes and the schedule for a node changes dynamically as it moves from tile to tile. The schedule allows nodes to maintain information about their local neighborhood. This information in turn is used to keep the schedule collision-free. We demonstrate the correctness of the algorithm, and discuss how the periodic schedule can be adapted for different scenarios.

In the last part of this dissertation we look at the distributed trigger counting problem. Suppose that there are n processors forming a clique, and external events cause triggers at each processor. An alert is to be raised for the user when the total number of triggers reaches a certain value w. We present a deterministic algorithm which is communication efficient in terms of the the maximum number of messages received by any processor in the system, i.e., the $MaxRcvLoad$, as compared to previous deterministic algorithms. We also give a lower bound for the $MaxRcvLoad$. This problem has many applications in ad hoc networks.

In the remainder of this section we provide an overview of the three different topics covered. We provide the motivation for these solutions and also outline our contributions.

1.1 Reliable Neighbor Discovery with an Abstract MAC Layer

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?

We define 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 [36] (with implementation details provided in [31]). 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. Many existing user level algorithms assume a neighbor discovery service which provides guarantees message delivery between neighbors. For example, the leader election algorithm of [27], the token circulation algorithm of [38], and the mutual exclusion algorithm of [50], all require an underlying neighbor discovery service. These problems are important primitives in distributed computing. In addition to these, even the most basic of tasks in mobile ad-hoc networks, such as routing [6, 40, 41] or broadcasting [7, 44] also require accurate and up-to-date knowledge about neighbor nodes. For example, [45] implements coordinate based routing by assuming nodes know the location of their two-hop neighbors. Similarly, [40] describes a routing algorithm for multi-hop wireless network that assumes one-hop neighbor information.

Contributions. The main contributions of this work 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.

1.2 Neighbor Knowledge and Collision Avoidance

In this part of our work we deal with the interrelated problems of neighborhood knowledge and coordinating the transmissions of mobile nodes so that reliable communication can take place. The problem is complicated by the fact that the nodes may be in continuous motion and hence the local neighborhood topology never stabilizes.

Motivation. Previous solutions to the problem adopt a probabilistic approach, including the hello protocols (cf. [6]) and reservation based MAC protocols (cf. [5, 29]). All of these protocols experience some probability of error, due to collisions in the wireless communication caused by two or more nodes broadcasting at the same time and thus disrupting the receipt of the message. Many applications can tolerate such errors. However, for some real time, mission critical applications, even a small probability of error might have severe penalties. We present a deterministic collision-free protocol which guarantees reliable communication despite the inherent drawbacks of a wireless ad hoc environment where nodes may be continuously in motion. Our protocol can be used to build a reliable communication infrastructure to meet the requirements of such mission critical applications.

Such a reliable communication infrastructure is of particular importance in applications for vehicular ad hoc networks (VANETs). These applications ensure the safety of drivers by warning them about collisions with other vehicles (cf. [39]) or advising drivers about adverse traffic conditions (e.g., rain, snow and fog). A protocol with deterministic guarantees is essential under such conditions since human life is at stake. It can also be used to relay information from the anti-skid systems and fog-probing radars already present in vehicles to police cars, ambulances, and snowplows (cf. [49]). Our system model which consists of nodes moving arbitrarily on the plane with bounded speed is in accordance with the motion of vehicles on highways, as well as on parallel roads and intersections in urban areas. In addition to VANETs our protocol also has applications in the area of robotic sensor networks [42] used for rescue and reconnaissance missions.

We also address the issue of deterministically maintaining up-to-date information about the local neighborhood of a node. The maintenance of this neighborhood knowledge is a part of our proposed solution and is interleaved with the collision-free schedule. It is also a significant problem in its own right–information about nearby nodes is required for numerous tasks in a mobile ad hoc network. For instance, neighborhood knowledge is needed for routing (cf. [6, 40, 41]), broadcasting (cf. [4, 7, 44]), distributed token circulation (cf. [38]), etc.

Contributions. The main contributions of this research are as follows

1. We develop a reliable communication scheme for mobile nodes which is collision-free.

2. We develop a deterministic technique for mobile nodes to maintain neighborhood knowledge as they move in and out of each others' broadcast range.

3. We discuss a technique for initial discovery of nodes already present in communication range when a node starts up.

The first two parts of our scheme mentioned above are interdependent on each other. Thus in our scheme, it is necessary for nodes to possess local neighborhood knowledge to transmit messages in a collision-free way. Since nodes can transmit without having collisions, they can maintain information about their local neighborhood in a timely and efficient way.

1.3 Communication Efficiency for Distributed Trigger Counting in Sensor Networks

The third part of this dissertation focuses on the communication efficiency of the distributed trigger counting problem. In this problem, introduced by [10], we assume that there are n processors, and triggers may be received by these processors due to external events. An alert is to be raised for the user when the total number of triggers reaches a certain value w which is specified by the user. Each processor may receive a different number of triggers. The order in which different processors receive triggers is not known in advance.

We assume an asynchronous computation model where the network topology is a clique. Message delivery is guaranteed. Furthermore, processors and links do not fail.

Motivation. Such an algorithm has applications in the field of sensor networks and distributed monitoring. In sensor networks sensors may be deployed to count the number of vehicles and raise an alert when a certain threshold is exceeded. They can similarly be deployed to count the number of sightings of a particular species of wildlife.

Contributions. The main contributions of this section are as follows:

1. We develop a deterministic algorithm for the trigger counting problem.

2. Our algorithm is communication efficient in terms of the $MaxRcvLoad$, defined as the maximum number of messages received by any processor in the system. The $MaxRcvLoad$ of our algorithm is $O(\log n \log w + \sqrt{n \log n \log w})$, compared to the best previous deterministic algorithms with a $MaxRcvLoad$ of $O(n \log w)$ [21].

3. We also give a lower bound of $\Omega(\log w)$ for $MaxRcvLoad$ for the case where the network graph is a tree. There are no previous lower bounds for the $MaxRcvLoad$.

1.4 Organization

The remainder of this dissertation is organized as follows. In Section 2 we give an overview of the related work for all three topics covered in this dissertation. In Section 3 of this dissertation we present the reliable neighbor discovery algorithms (which have appeared in [15]). In Section 4 we give our algorithm for neighbor discovery and medium access (published in [48]). In Section 5 we present our algorithm for the distributed trigger counting problem. In Section 6 we summarize our contributions and discuss future work.

2. RELATED WORK

There has been a lot of related work for all three problems addressed in this thesis. In this section we give an overview of the related results for all three problems.

2.1 Reliable Neighbor Discovery with an Abstract MAC Layer

There has been a lot of previous work related to neighbor discovery. For example in hello protocols [6], 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 [8] a deterministic algorithm for computing two-hop neighbors in static networks is presented. In [37] a technique is presented for secure neighbor discovery for static networks. Similarly, [35] presents a deterministic protocol for neighbor discovery in static cognitive radio networks. Lastly, [47] considers neighbor discovery in static networks with directional antennas.

A topology discovery algorithm for mobile nodes is given in [11]; however, it is assumed that few nodes move and that their speed is severely constrained. An asynchronous neighbor discovery and rendezvous protocol is presented in [16]. 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 [17]. 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 [48] 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 [14]. 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. The work presented here is an extension of the work presented in [15].

So far we have referred to three different layers: the user layer, the neighbor discovery layer, and the MAC layer. We have already discussed in detail the related work that concerns the neighbor discovery layer, but there is a lot of related work for various communication tasks in the other layers. For example, the authors in [33] and [34] deal with conflict resolution for multiple-access channels, which can be used as building blocks of a MAC layer. Another problem is broadcasting or one-to-many communication, in which a message from a source node is to be delivered to all nodes in the network, over multiple hops. Broadcast algorithms [1, 13] are typically implemented on top of both a neighbor discovery layer and a MAC layer. A comparison of the modular approach (separate neighbor discovery, MAC, and broadcast layer) versus an approach where the neighbor discovery layer and MAC layer (and perhaps the broadcast layer) are merged is still an open problem.

2.2 Neighbor Knowledge and Collision Avoidance

Much of the previous work on collision-free broadcasting in wireless ad hoc networks assumes static nodes. For example, Gandhi *et al.* consider the problem of collision-free broadcasting in wireless ad hoc networks with static nodes in [20]. The authors focus on how to minimize latency and retransmissions in such a network, and show that their algorithm is $O(1)$ of optimal in terms of both. In their algorithm they construct a broadcast tree and then use it to schedule transmissions such that all nodes receive a message in a collision-free manner. However, they suggest that for dynamic network topologies, construction and maintenance of broadcast trees is not efficient. Prabh *et al.* also present a distributed transmission scheduling algorithm for hexagonal wireless ad hoc networks in which nodes remain static [43]. The algorithm provides network-wide conflict-free packet transmission and gives a guarantee on transmission latency. They also give a clock synchronization algorithm for scheduling based on overheard messages sent by neighbors. They assume that there is a base station or sink node at the center of the network. Their focus is on convergecast or many to one communication. They also mention that their assumed topology of nodes is an oversimplification of real topologies. The network topology is cluster-based, and CDMA is used for intra-cluster communication. It is assumed that cluster heads maintain the transmission schedule of nodes in their own cluster. The authors suggest that CDMA is not scalable for multi-hop transmission, and hence a conflict-free schedule is given for inter-cluster communication.

Certain protocols which handle node mobility rely on the presence of centralized infrastructure. For example Arumugam *et al.* give a self-stabilizing, deterministic TDMA algorithm for sensor networks [2]. Their system architecture has three layers: (1) the token circulation layer, (2) the TDMA layer, and (3) the application layer. The TDMA layer implements a distance-2 coloring algorithm which is used to compute TDMA slots. The addition and removal of sensors is allowed, however, they assume the presence of a base station which maintains a spanning tree of the network and is responsible for token circulation. There is no discussion of how the algorithm would behave if the nodes were in continuous motion. Like our work they also assume that time synchronization is present during token circulation. Local neighborhood knowledge is also assumed.

In [32] the authors assume that the sensors are located exactly at the points of a regular lattice. For mobile sensors they suggest that the lattice points should be spaced finely enough so that just one sensor is within the Voronoi region of a single lattice point. However they do not consider the case of sensors crossing the boundaries of Voronoi regions while they transmit. Furthermore, making the lattice points closer together would lead to a highly inefficient schedule with a very large number of transmission slots. We have used the concept of tiling the plane in order to get a deterministic schedule, however, we have also addressed the issues that arise when fast moving nodes cross the boundaries of tiles.

In [3] Baldoni *et al.* consider a model in which nodes can move arbitrarily on the plane with a bound on the speed. They show that using conventional assumptions of node connectivity, it is impossible to carry out geocasting (transmitting information to nodes within a specific geographical area). A stronger version of connectivity, in which nodes remain neighbors for a certain interval of time, is needed to solve the problem of geocast.They present bounds for the speed of nodes in relation to the speed of information propagation for mobile ad hoc networks, as well as bounds for the number of rounds required for reliable message delivery. However, they do not give a constructive solution to the problem of collision-free communication among mobile nodes. In [28] Ioannidou presents a model for mobile ad hoc networks based on tiling the plane with hexagons. This model is then used to implement dynamic quorum systems. The hexagons are assumed to be surrounded by circles called camases, which represent bounds on how far nodes can travel within a particular interval of time. However, the authors assume that there is no interference in the network, that is, collision avoidance is performed by a lower layer of the network.

2.3 Communication Efficiency for Distributed Trigger Counting in Sensor Networks

The trigger counting problem has been investigated previously. In [10] a randomized algorithm $LayeredRand$, is presented with $MaxRcvLoad$ equal to $O(\log n \log w)$ with high probability. The authors in [10] look at the problem in an asynchronous message passing system. They consider a clique with point-to-point links. In their solution it is assumed that nodes are divided into layers. Processors in each layer aggregate a certain number of triggers and send this information to some processor in the upper layer chosen uniformly at random.

A randomized algorithm is also presented in [9] which has $MaxRcvLoad$ equal to $O(\log n + \log w)$ with high probability. This algorithm is similar to the above mentioned $LayeredRand$ algorithm however, it is more complicated and the number of rounds required for termination is not guaranteed as in $LayeredRand$.

The authors of [21] give a deterministic algorithm for which the $MaxRcvLoad$ is $O(n \log w)$. In [21] it is also shown that any deterministic algorithm for the trigger counting problem must have message complexity $\Omega(n \log(w/n))$. However, there are no bounds for the $MaxRcvLoad$.

In [19] the authors study the trigger counting problem. However, they call it the *threshold detection* problem. The authors give a lower bound of $\Omega(\log w)$ for the average message complexity of any randomized protocol with constant failure probability which solves the threshold detection problem. They also give a randomized protocol and then convert this protocol into a deterministic protocol in a model called the transmissions model. In this model if a node transmits a message, this message is delivered to all its neighbors in the network. In this model, for the deterministic protocol, the maximum number of transmissions made by any node is $O(\log^2 w \log^2 n)$. In contrast our protocol is for the point-to-point model and we consider communication efficiency in terms of the maximum number of messages received by any node.

3. RELIABLE NEIGHBOR DISCOVERY FOR MANETS [1]

3.1 Introduction

In mobile ad hoc networks (MANETs), the underlying communication graph changes over time. However 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? In this section we define a reliable neighbor discovery layer which establishes links over which message delivery is guaranteed. User layer algorithms can then use this neighbor discovery layer to solve application problems. We then present two algorithms that implement a reliable neighbor discovery layer with different progress guarantees.

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 [36] (with implementation details provided in [31]). This modular approach makes the algorithm easier to design, understand and verify. Moreover, it allows us to focus on the challenge of dealing with arbitrary mobility patterns while trying to maximize the time that the links remain up while guaranteeing all links are "reliable".

We first describe a basic region-based neighbor discovery protocol which relies on sending notification messages when nodes enter and exit regions to set up the

[1]Part of this section is reprinted with permission from Cornejo, A., Viqar, S., Welch, J.L.: Reliable neighbor discovery for mobile ad hoc networks. In: Proceedings of the DIALM-POMC Joint Workshop on Foundations of Mobile Computing, pp. 63–72, Copyright 2010 ACM, Inc. http://doi.acm.org/10.1145/1860684.1860699

communication links. The correctness of this algorithm hinges on 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 handle nodes crossing the region boundaries, we present a second protocol (the uniform neighbor discovery protocol) which runs multiple instances of the basic region-based neighbor discovery protocol, each of them using a different region partition. We then describe how by composing the output of each of the instances appropriately, we can guarantee a reliable neighbor discovery protocol with stronger progress guarantees. We show that this protocol can be used with region partitions that are either regular square tilings, or regular hexagonal tilings.

We also describe an additional property of a neighbor discovery layer called coordination. Depending on the level of coordination, a communication link between two nodes may be established at the same time at both endpoints or it may come up at different times at the two endpoints. We show that the basic neighbor discovery protocol provides a higher level of coordination as compared to the uniform neighbor discovery protocol. We also discuss the impact of coordination on different applications.

3.2 System Model

The Timed I/O Automata (TIOA) modeling formalism [30] 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 environment and communicate using a local broadcast primitive.

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. A pair of regions with a nonempty intersection are said to be 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. Note that this is not an unrealistic assumption for VANETs (Vehicular Ad Hoc Networks) or other outdoor mobile networks such as rescue robots. CarTel [23] is one example of a mobile system where each vehicle is equipped with a GPS. For indoor environments mobile nodes may use localization schemes using beacons or receivers which locate the mobile device either by transmitting periodic signals, or by listening to the transmissions of the mobile device. Such a scheme is presented in [46].

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. Due to the motion planning algorithm, the speed and trajectory of the node are predetermined. Furthermore, information about the entire trajectory is not required. The only information required is that a node is going to cross a region boundary in the near future.

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. Even if the one-hop message delay is of the order of a hundred milliseconds, vehicles traveling at 200 km per hour can only cover 5.55 meters ≤ 6 meters in this time interval.

There are five components in the system: the *network layer*, the *abstract MAC layer*, the *MAC Broker layer*, the *neighbor discovery layer*, and the *user layer* (see Figure 3.1, and Table 3.1).

3.2.1 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.

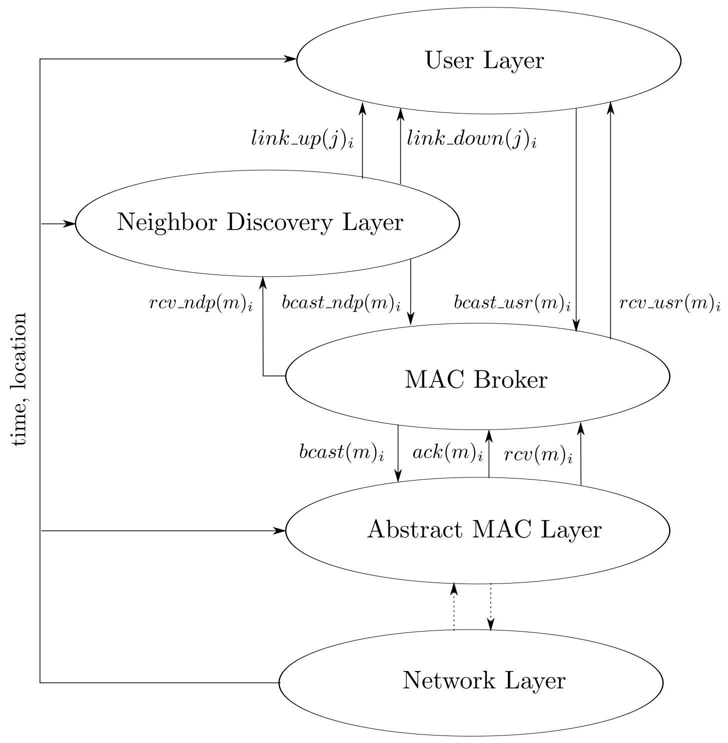


Fig. 3.1.: MANET system block diagram.

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.

3.2.2 The Abstract MAC Layer

We present a slight simplification of the MAC layer specification [36] by ignoring the functionality to abort messages in transit. The abstract MAC layer provides reliable local broadcast with timing guarantees. It also provides acknowledgement

Table 3.1: Interface actions for the MANET system.

Automaton	Action	Description
Abstract MAC Layer	*bcast(m)$_i$*	Reliable local broadcast of message *m* at node *i*.
	rcv(m)$_i$	Reception of message *m* at node *i*.
	ack(m)$_i$	Acknowledgement for message *m* at node *i*.
MAC Broker	*bcast_usr(m)$_i$*	Local reliable broadcast of an application message *m* at node *i*.
	rcv_usr(m)$_i$	Reception of an application message *m* at node *i*.
	bcast_ndp(m)$_i$	Local reliable broadcast of neighbor discovery message *m* at node *i*.
	rcv_ndp(m)$_i$	Receiving neighbor discovery message *m*.
Neighbor Discovery Layer	*link_up(j)$_i$*	A link is up for node *j* at node *i*.
	link_down(j)$_i$	A link is down for node *j* at node *i*.

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 $rcv(m)_i$. 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 acknowledgement. These time bounds are constant and we assume that they are available to algorithms implemented on top of the abstract MAC layer. These time bounds take into account the maximum possible amount of contention, as defined by the node degrees that occur in the dynamic communication graph (G_{comm}) induced by the motion of the nodes.

Note that these time bounds are only for one-hop message delays. In case the total number of nodes in the network is not known, the message delay over multiple hops may be unbounded. These one-hop message delays are related to the maximum contention in the network, which can be bounded if there is a bound on the maximum degree of a node. Hence, it is not an unreasonable assumption to have bounded one-hop message delays. In addition, if such bounds are provided by the MAC layer only with a high probability then, a protocol implemented on top of such a MAC layer can provide correctness guarantees with high probability.

The cost of implementing this abstract MAC layer exactly as described in [36] might be prohibitively large. However, it is possible to provide similar guarantees with a high probability (see [31]).

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 acknowledgements, and no receives after acknowledgements. Finally, every $bcast(m)_i$ causes an $ack(m)_i$.

3.2.3 The MAC Broker Layer

As its name suggests, this layer acts as a broker between the MAC layer and both the user and neighbor discovery layers. It provides the following three guarantees:

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 (described in Subsection 2.4).
3) Routing: Received messages are routed correctly to either the neighbor discovery or the user layer.

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 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 a flag is attached to the message before pushing it in the queue. When receiving a message through the input action $rcv(m)_i$ this flag is removed and used to trigger either a $rcv\_usr(m')_i$ or a $rcv\_ndp(m')_i$ action.

3.2.4 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 $action_i^j(t) \in \{link\_up(j)_i, link\_down(j)_i\}$ be the most recent link event for link (i,j) at node i at time t. If $action_i^j(t) = link\_up(j)_i$ 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 ((a, b, k)-Weak Progress)**.** *There exist constants $a, b \in \mathbb{R}^+$ and $k \in \mathbb{N}$, 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 (which implies a distance of at most r_{min} between i and j), 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 3.4). 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.

Definition 4 ((a, b, k)-Uniform Progress)**.** *There exist constants $a, b \in \mathbb{R}^+$ and $k \in \mathbb{N}$, 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 (which implies a distance of at most r_{min} between i and j), 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 (k-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}).*

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.*

3.2.5 The User Layer

The user layer automaton is a composition of separate (and non-interacting) automata for the users $\{1, \ldots, n\}$. A user learns about the state of its neighbors through the *link_up* and *link_down* output actions of the neighbor discovery automaton. Similarly it broadcasts and receives messages through the MAC broker automaton using the *bcast_usr* and *rcv_user* actions.

3.3 Basic Neighbor Discovery Protocol

Here we describe the basic neighbor discovery protocol (referred to as BNDP), which satisfies the reliable neighbor discovery layer specifications with weak progress. 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.

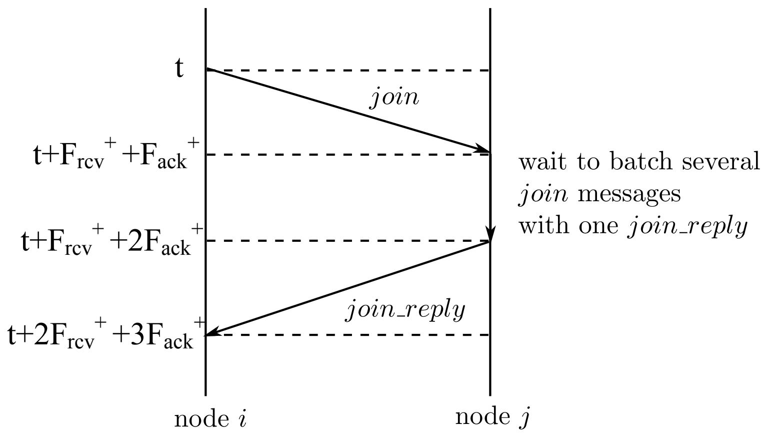
When a node i enters a new region and determines that it is going to remain there for sufficiently long, it broadcasts a *join* message. The recipients of the *join* message will start setting up the corresponding link to i if they have not already done so. The *join* message also serves as a request 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 *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 a 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 then 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 $\delta_{LU} = 2F^{+}_{rcv} + 3F^{+}_{ack}$ (see Figure 3.2). 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,



t
$t+F_{rcv}^{+}+F_{ack}^{+}$
$t+F_{rcv}^{+}+2F_{ack}^{+}$
$t+2F_{rcv}^{+}+3F_{ack}^{+}$
join
wait to batch several *join* messages with one *join_reply*
join_reply
node i
node j

Fig. 3.2.: Time required for setting up a link.

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 would potentially require a *join_reply* message), *join_reply* messages are buffered in intervals of F_{ack}^{+} so that multiple *join* messages can be answered using a single *join_reply* message.

Consider the following scenario. Suppose that node i is in region R_i of the network. Now suppose that $n-1$ nodes move into region R_i and send *join* messages. Using a naive strategy will result in node i sending $n-1$ *join_replies*. This would result in overflow in the Neighbor Discovery Protocol message queue in the MAC Broker layer. Therefore we have node i wait for F_{ack}^{+} time and collect the *join* messages and responds with one *join_reply* message. This 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. Therefore $\delta_{LU}=2F_{rcv}^{+}+3F_{ack}^{+}$.

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 $\delta_{LD} = 2F^+_{rcv} + (q+1)F^+_{ack}$ (see Figure 3.3), where q is the maximum size of the queue. 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.

Other relevant details of the algorithm which we have not yet mentioned, are that it keeps track of the set of neighbors using a set S, which is both checked before either the *link_down* or *link_up* actions are executed, and updated after executing them. Also to avoid conflicts a node always discards any message it receives from a node which is more than k hops away.

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.

For BNDP the number of messages sent depends on the number of times nodes cross region boundaries. This is because every time a node crosses a boundary two messages (a *leave* and a *join*) are sent. Other nodes in the neighborhood send *join_reply* messages in response. The number of *join_reply* messages corresponding to one boundary crossing is bounded by the maximum degree of G_{comm}. As described

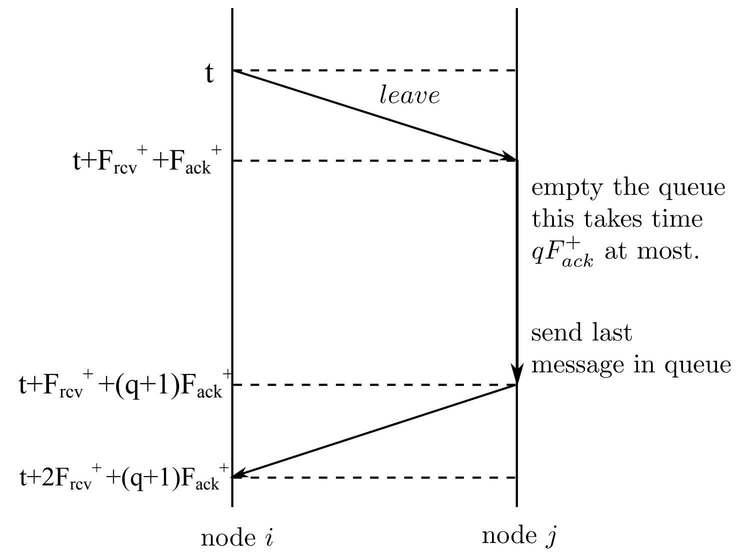


Fig. 3.3.: Time required for taking down a link.

above, the timing of these messages depends on the delay bounds provided by the abstract MAC layer and the maximum queue size q of the MAC broker layer.

The detailed TIOA code for the protocol is given in the appendix. In the next section we will describe a protocol that allows such nodes to maintain communication links.

3.3.1 Correctness Proof.

In this subsection, we show that the basic neighbor discovery protocol described satisfies the well-formedness, weak progress, validity, and reliability defined in Section 3.2. Specifically, it satisfies the (a, b, k)-weak progress condition with constants $a = \delta_{LU}$ and $b = \delta_{LD} + L$ where $L \geq 0$ is a user-provided parameter.

Theorem 1. *BNDP satisfies the well-formedness condition.*

Proof. 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. □

Theorem 2. *BNDP satisfies the (a, b, k)-weak progress condition.*

Proof. 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 simultaneously). 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. □

Theorem 3. *BNDP satisfies the k-validity condition.*

Proof. 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 messages from nodes which are more than k hops away are always ignored, then for (j, i) to be Up at time t it must be that node i sent a *join* (or *join_reply*) message when it has k hops away from j and this message was received by j before time t. Let $t' < t$ be the last time that node i sent a *join* (or *join_reply*) message when it was at most k hops away from j and which was received by node j before time t.

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

If node j sent the *leave* message, then it would have set link (j, i) to Dn immediately. On the other hand, if node i sent the *leave* message, then it sent it at time $t'' - \delta LD$ and it will be received by node j at time $t'' - \delta_{LD} + F^+_{rcv} + F^+_{ack} < t''$, at which point node j would set the link (j, i) to Dn. In either case since by assumption the last *join* (or *join_reply*) message received by j from i while they were k hops apart was received at time $t' < t''$, this implies that the link (j, i) remains Dn at time t – a contradiction. □

Theorem 4. *BNDP satisfies the reliability condition.*

Proof. 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. We will show that this is the case.

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. However since by assumption at time t the link is Up then $t' > t + \delta_{LD}$ 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. $\square$

3.4 Uniform Neighbor Discovery

The weak progress condition only requires that links should be formed between relatively "stable" nodes. In other words, links are only required between a pair of nodes which do not cross any region boundaries and either remain inside the same region or remain in two regions that are close. In contrast, uniform progress requires links between nodes which stay close to each other for sufficiently long intervals of time, irrespective of whether they cross region boundaries.

We showed in Section 3.3, Theorem 2, that the basic neighbor discovery protocol guarantees weak progress. However, it does not guarantee uniform progress. Hence the basic neighbor discovery protocol does not guarantee links between nodes which stay close together during arbitrarily long intervals of time, but keep crossing boundaries. This can be restrictive in environments where nodes frequently cross region boundaries. In this section we present a uniform neighbor discovery protocol which guarantees uniform progress, and allows nodes which stay close for a sufficient time to form links, even if they cross region boundaries during that time.

Before we describe in detail how to implement the uniform neighbor discovery protocol, let us define some properties related to the motion of nodes, which are used to guarantee uniform progress. Note that in the following definition a and b correspond to the same constants given in the definition of (a, b, k)-weak progress.

Definition 7. *Suppose that $\mathcal{Z}$ is a region partitioning scheme. We say a node $v \in V$ is* stable *in partition $\mathcal{Z}$ at time t, if $\forall t \in [t-a, t+b]$ v stays in the same region with respect to $\mathcal{Z}$ and does not cross any region boundary which belongs to $\mathcal{Z}$.*

Definition 8. *We say a node is* jittering *in partition $\mathcal{Z}$ at time t, if it is not stable in $\mathcal{Z}$ at time t.*

Note that weak progress requires that nodes stay in one region for a certain interval of time in order to guarantee a link. Hence, weak progress requires only *stable* nodes to form links. Uniform progress on the other hand, requires not only stable nodes to form links, but also *jittering* nodes that are sufficiently close for some period of time.

From now on we assume that there exists some constant $c \in \mathbb{R}$ that bounds the maximum speed of the nodes. Since in real deployments motion speed is always 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.

In Figure 3.4 a) the node is jittering in the solid line partition, but not in the dashed line partition where it is stable. This is a key observation related to the stability property of nodes which we use in order to guarantee uniform progress.

3.4.1 Stability and Displaced Grid Partitions

In this section we discuss the motion of nodes with respect to regular partitions of the plane, and how these partitions can help in boosting the progress guarantees provided by BNDP. We only consider regular grid partitions or square tilings of the plane. Other types of partitions such as hexagonal tilings of the plane are discussed in Section 3.4.4.

Regular grid partitions can be thought of as consisting of two set of lines. Lines in the same set are parallel to each other and uniformly spaced. Lines from different sets are perpendicular to each other and intersect at only one point. We consider only grid partitions which are aligned with the x- and y-axis. Hence, one set of lines can be referred to as horizontal lines, while the other set can be referred to as vertical lines.

Definition 9. *Let $\mathcal{Z}_0$ be a grid partition where ℓ is the distance between two parallel lines. We define the set of w grid partitions $\mathbf{Z}_w = \{\mathcal{Z}_0, \ldots, \mathcal{Z}_{w-1}\}$ as consisting of identical axis-aligned partitions displaced from each other by a distance of $i \cdot \frac{\ell}{w}$ along both the horizontal and vertical axes.*

If a node is jittering with respect to a partition, it means that it has crossed at least one boundary line of that partition during time interval $[t-a, t+b]$ (where a and b refer to the constants in the definition of (a, b, k)-weak progress).

We now show a result concerning the motion of nodes with respect to $\mathbf{Z}_2$.

Theorem 5. *For the set of regular grid partitions $\mathbf{Z}_2$ there exists a node motion that respects a speed limit of c, where $c > 0$, while jittering in both partitions.*

Proof Sketch. For the set $\mathbf{Z}_2$ any horizontal line of one partition intersects with any vertical line of the second partition at a single point. Therefore, it is possible to define a motion such that the node speed never exceeds c and the node is jittering on both partitions (see Figure 3.4 b)). $\square$

In the following we show an important result concerning a set of three or more displaced grid partitions. We show that for the case of three or more partitions, a node can jitter in at most two partitions at time t. Again a and b here refer to the same constants given in the definition of (a, b, k)-weak progress.

Lemma 1. *For any set of three or more regular, axis-aligned grid partitions, such that the minimum distance between two parallel lines is x, a node which has a maximum speed of $x/(a+b)$ during the interval $[t-a, t+b]$, jitters in at most two partitions at time t.*

Proof. Let us suppose in contradiction that a node respects a speed limit of $x/(a+b)$ during the interval $[t - a, t + b]$ and jitters with respect to three or more partitions at time t. This implies that it crosses one region boundary of three or more distinct partitions during the time interval $[t-a, t+b]$. Since at least two of these boundaries are parallel to each other and the distance between them is x, the node must have a speed which is strictly greater than $x/(a + b)$. This is a contradiction. $\square$

The above lemma concerns the stability of a single node. However, for our problem we have to consider links formed between pairs of nodes. Hence, it is important to investigate the stability properties of a pair of nodes with respect to displaced partitions. Lemma 1 leads to the following corollary, which shows that even four displaced partitions are not sufficient for two nodes to be stable in the same partition.

Corollary 1. *Consider any set of four regular, axis-aligned 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.*

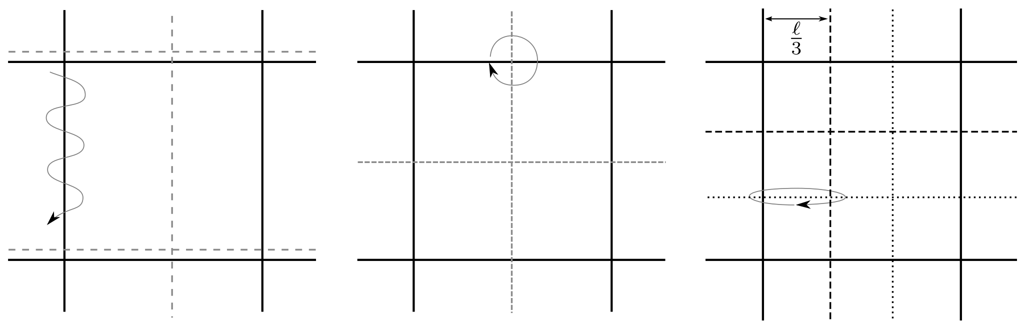
Proof. Consider two nodes i and j. Suppose that node i is jittering in subset A of the partitions at time t, where $|A| \leq 2$ by Lemma 1, and node j is jittering in subset B of the partitions at time t, where $|B| \leq 2$ by Lemma 1. It may be that $|A| + |B| = 4$ and $A \cap B = \emptyset$. In this case there is no partition in which both nodes are stable at the same time. $\square$

Finally we show that a set of five identical displaced partitions is sufficient to guarantee the stability property for a pair of nodes.

Lemma 2. *For any set of five or more regular, axis-aligned grid partitions, such that the minimum distance between two parallel lines is x, and a pair of nodes i and j which have a maximum speed of $x/(a + b)$ during the interval $[t - a, t + b]$, there exists a partition where both i and j are stable at time t.*

Proof. Consider two nodes i and j. Suppose that node i is jittering in subset A of the partitions at time t, where $|A| \leq 2$ by Lemma 1, and node j is jittering in subset B of the partitions at time t, where $|B| \leq 2$ by Lemma 1. Since, $|A| + |B| \leq 4$, at least one partition is not in $A \cup B$, and there exists at least one partition at time t, where both nodes i and j are stable. $\square$

This leads us to the following Theorem, which shows that we can use the set of partitions $\mathbf{Z}_5$, as defined earlier, in order to help achieve the desired stability property for a pair of nodes



(a) Jittering in a single partition

(b) Jittering in $\mathbf{Z}_2$.

(c) Jittering in $\mathbf{Z}_3$.

Fig. 3.4.: Jittering trajectories

Theorem 6. *If two nodes respect a speed limit of $\ell/5(a+b)$ during the interval $[t-a, t+b]$, there exists a partition $\mathcal{Z}_i \in \mathbf{Z}_5$ (for which $x = \ell/5$) with respect to which both nodes are stable at time t.*

3.4.2 Uniform Neighbor Discovery Protocol

In this section we present the *uniform neighbor discovery protocol* (UNDP) which guarantees uniform progress by using the stability properties of nodes with respect to a set of five displaced partitions as discussed in the previous subsection.

The main idea used to implement UNDP is to simultaneously execute five instances of BNDP, each associated with one of five identical, displaced partitions. In particular we use the set $\mathbf{Z}_5$ of identical partitions as defined in the previous subsection and run a separate instance of BNDP for each of the five partitions. We then *compose* the output of these five instances to get the desired uniform progress guarantee. Figure 3.5 shows the detailed interactions between different components.

We show that this technique of composing the output of different BNDP instances provides stronger guarantees, however, these guarantees apply to a smaller

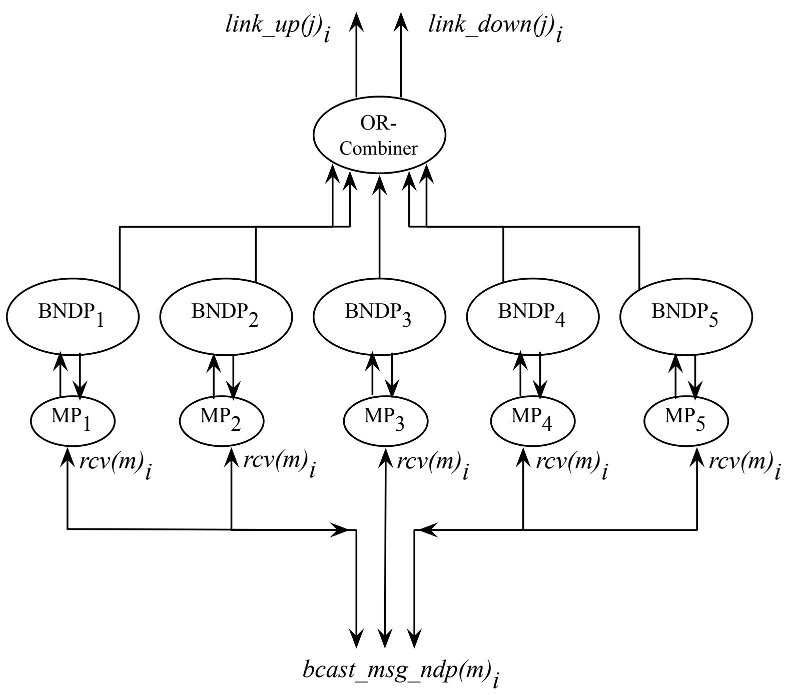


Fig. 3.5.: Uniform neighbor discovery block diagram. Each BNDP instance has a message processing (MP) automaton associated with it.

neighborhood area as compared to BNDP. This means that if we set the maximum hop distance as k' for each instance of BNDP, then after composition, the final UNDP output will guarantee uniform progress for nodes which have $k = k' - 1$ as the maximum hop distance between them. We discuss the trade-off between progress and hop distance in detail at the end of this section.

All messages sent and received by each BNDP instance must be routed to the correct instance of BNDP at the other node. For this purpose we have a message processing (MP) automaton with each BNDP instance which attaches an id to each message sent and removes the ids of received messages. The id associates each message with a particular partition among the set of five partitions and hence, with a particular instance of BNDP.

In order to compose the outputs of the five BNDP instance we have an OR-Combiner automaton. The OR-Combiner receives the output of each of the five instances and outputs a $link\_up(j)$ if there exists any BNDP instance for which node j is a neighbor (i.e. the first time that a $link\_up(j)$ happens for any BNDP instance). The OR-Combiner outputs a $link\_down(j)$ if there exists no BNDP instance for which j is a neighbor.

3.4.3 Proof of Correctness for the Uniform Neighbor Discovery Protocol

Theorem 7. *UNDP satisfies the well-formedness condition.*

Proof. UNDP satisfies well-formedness since the output of the OR-Combiner is composed of the outputs of the BNDP instances and BNDP satisfies the well-formedness condition. □

Theorem 8. *UNDP satisfies the k-validity and reliability conditions.*

Proof. The output of the OR-Combiner is composed of the outputs of the BNDP instances. A link between two nodes i and j is Up at time t, if it is Up in at least one of the BNDP instances. Since BNDP satisfies validity and safety, the distance between i and j is less than r_{min}, and a message sent by node i at time t is received by node j. □

The (a, b, k)-uniform progress condition states that nodes which remain at most k hops apart during some time interval should be guaranteed a communication link. Since the number of hops between two nodes is measured with respect to a single partition,

we pick one of the five partitions arbitrarily and define the number of hops with respect to it.

However, the algorithm actually provides stronger connectivity guarantees, since the link between two nodes remains Up as long as the nodes remain k hops apart with respect to any partition (the partition can be different for every time instant).

Before we show that UNDP guarantees uniform progress, let us define some terms.

Definition 10. *Suppose we have two points p and q on the plane. If we draw a straight line segment U which connects p and q, $hop_y^Z(p, q)$ is the number of vertical lines (parallel to the y-axis) belonging to partition Z which intersect U.*

Definition 11. *Suppose we have two points p and q on the plane. If we draw a straight line segment U which connects p and q, $hop_x^Z(p, q)$ is the number of horizontal lines (parallel to the x-axis) belonging to partition Z which intersect U.*

Definition 12. *Let $hop^Z(p, q)$ be the number of hops between two points p and q with respect to partition Z in the region graph of partition Z as defined in Section 3.2.*

From the definition of hops with respect to grid partitions we have:

$$hop^Z(p, q) = max(hop_x^Z(p, q), hop_y^Z(p, q))$$

Lemma 3. *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. Suppose that A and B are two partitions which are identical but displaced. Now suppose, without loss of generality that for two points p and q $hop^B(p, q) \geq hop^A(p, q)$. Now suppose that $hop_x^A(p, q) = x_a$ and $hop_y^A(p, q) = y_a$. Since the partitions are identical but displaced from each other by some distance, there may be only one more horizontal line from partition B between points p and q, than from partition A. The same is true of vertical lines. Hence we have the following:

$$hop_x^B(p,q) \leq x_a + 1$$
$$hop_y^B(p,q) \leq y_a + 1$$

However we know that:

$$hop^B(p,q) = max(hop_x^B(p,q), hop_y^B(p,q))$$

Hence we have:

$$\begin{aligned} hop^B(p,q) &\leq max(x_a + 1, y_a + 1) \\ &= hop^A(p,q) + 1 \end{aligned}$$

From this we have:

$$hop^B(p,q) \leq hop^A(p,q) + 1$$

□

We can now prove that UNDP satisfies the uniform progress guarantee.

Theorem 9. *UNDP satisfies the (a,b,k)-uniform progress condition.*

Proof. Consider two nodes i and j which remain k hops apart with respect to one out of the five partitions during time interval $[t-a, t+b]$ and have a maximum speed of $\ell/5(a+b)$ during this time interval.

These nodes are stable in at least one partition due to Theorem 6. Since these nodes are k hops apart in one partition, they are $k' = k+1$ hops apart in all five partitions due to Lemma 3. This means these nodes are $k' = k+1$ hops apart in the partition in which they are stable. We also know that the BNDP instance associated with this partition guarantees weak progress. This means that the link (i, j) remains *Up* with respect to this partition during time interval $[t-a, t+b]$. This further implies that the link remains *Up* in the output of the OR-Combiner. Hence, uniform progress is guaranteed. $\square$

Note that the stronger progress guarantees associated with UNDP apply to a smaller neighborhood area ($k = k' - 1$ hops) as compared to BNDP (k' hops). If we increase the value of k' and fit a larger number of regions inside the circle of radius r_{min}, we can decrease the difference between the area covered by $k = k' - 1$ hops and k' hops. However, since we now have more regions in the same area, a mobile node may have to cross more region boundaries while following a certain motion pattern. This means that more messages would be sent, since every time a node crosses a boundary two messages (*leave* and *join*) are sent by the crossing node and other nodes in the neighborhood send *join_reply* messages. These *join_reply* messages are bounded by the maximum degree of G_{comm}.

In addition to this, the value of parameter k cannot be increased beyond a certain value. As k increases, x, which is the minimum distance between two parallel line boundaries that belong to distinct partitions, becomes smaller. This means that nodes traveling with a certain speed can now cross two boundaries that belong to distinct partitions, in a smaller interval of time. However, nodes must respect a speed limit of $x/(a+b)$. The parameters a and b depend on the queue size q, and the delay bounds provided by the abstract MAC layer: F^{+}_{rcv}, and F^{+}_{ack}. These delay bounds depend on the implementation of the abstract MAC layer and are independent of the neighbor discovery layer. Reducing q which is the size of the queue means placing stricter requirements on the user layer to prevent overflow. Hence, it may not be possible to reduce the values of a and b beyond a certain point. Therefore, in order to tolerate fast node speeds, k should not be increased beyond a certain threshold.

3.4.4 Stability and Displaced Hexagonal Partitions

In this section we will prove a simple result which allows us to translate the main results we showed for regular grid partitions for a particular family of hexagonal partitions. The most important consequence is that there is a family of 5 regular hexagonal partitions which is sufficient to guarantee the stability of any pair of nodes in at least one partition (assuming reasonable speed limits). Note that we do not characterize the optimal placement of these partitions.

Hexagons, like squares, form a regular tiling of the plane and provide a close approximation to the circular wireless broadcast range of mobile nodes. In a hexagonal tiling of the plane the boundaries between different regions can be described by the union of three sets of line segments. Line segments from the same set are parallel or on the same line and line segments from different sets may intersect with each other.

Let $\mathcal{H}_0$ be a hexagonal partition where h is the distance between any two parallel sides of a hexagon. We consider a family of p hexagonal partitions $\mathbf{H}_p = \{\mathcal{H}_0, \ldots, \mathcal{H}_{p-1}\}$, where partition $\mathcal{H}_i$ is a copy of $\mathcal{H}_0$ displaced by $i \cdot \frac{h}{p}$. This displacement is along any axis which is perpendicular to any two parallel sides of a hexagon (see Figure 3.6). For the purpose of this discussion, we will assume the displacement is along the horizontal axis.

It will be convenient to group the boundaries of each partition into three groups of parallel line segments, and prove some properties about these groups. We summarize these properties in the next lemma.

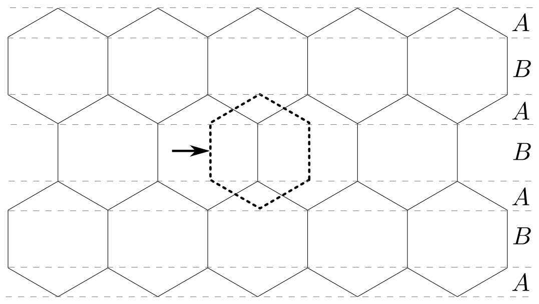


Fig. 3.6.: Hexagonal partitions.

Lemma 4. *The region boundaries of the family* $\mathbf{H}_p$ *can be separated into three sets of line segments* R*,* S *and* T *(where segments from the same set are either parallel or lie on the same line), such that:*

The intersections between two region boundaries that belong to distinct region partitions involve only one line segment from the set S *and one line segment from* T *(and never a line from* R*).*

Proof. First, let us cover the space by two disjoint sets of horizontal strips

There is a single set of parallel line segments (specifically vertical line segments) in the strips of type B, let R denote the set of these line segments. Since the region partitions in $\mathbf{H}_p$ are displaced horizontally, it should be evident that no two line segments in R intersect.

In contrast, in the strips of type A there are two sets of parallel line segments, which we group into the sets S and T, each of them containing parallel line segments.

As before, since the region partitions in $\mathbf{H}_p$ are displaced horizontally (and neither the lines in S or T are horizontal), it should be evident that no two line segments in S (and no two line segments in R) intersect.

Moreover, for a line segment $x \in R$ and $y \in S$ (or $y \in T$) it is also never the case that x and y intersect if they belong to different region partitions. In other words, if the partition boundaries of two regions A and B (of two different partitions) intersect, it is always the case that they intersect at the point where a line segment $x \in S$ and a line segment $y \in T$ intersect. □

The key property that allowed us to show the main results of Section 3.4.1 was that a regular grid partition consists of two sets of lines, and lines from the same set never intersect, while lines from different sets always intersect. On the other hand, with $\mathbf{H}_p$ we now have three sets of parallel line segments, R, S and T. However, from the previous result we know that intersections of two different partition never involve lines from the set R, and they always involve one line from S and one line from T.

This allows us to obtain analogous versions of Theorems 5, and 6 for hexagonal partitions, if a node respects a speed limit of $x/(a+b)$ (where x is the minimum distance between two parallel lines that belong to distinct partitions) during the interval $[t-a, t+b]$.

3.5 Coordination

In this section we discuss coordination between the endpoints of a link in the context of neighbor discovery. In an asynchronous network, each endpoint may have a different perspective on the status of a particular communication link. Depending on the level of coordination between the two endpoints, the link may be established at the same time at both endpoints or it may come up at different times at the two endpoints. The underlying neighbor discovery protocol may control link status notifications to provide different levels of coordination, according to the requirements of different applications. We define three different types of coordination conditions for neighbor discovery:

1. Coordinated Neighbor Discovery
2. Loosely Coordinated Neighbor Discovery.
3. Uncoordinated Neighbor Discovery.

Each of these specifications is defined by the level of coordination between the two endpoints of a communication link. In coordinated neighbor discovery, the link comes Up and goes Dn at the same time at the two endpoints. Coordinated neighbor discovery is defined by the following requirement:

Definition 13 (Coordinated Neighbor Discovery)**.**

A link_up at one endpoint occurs at time t if and only if a link_up at the other endpoint occurs at time t. There is an analogous condition for link_downs.

This specification is restrictive and may be costly to implement. However, since the events are perfectly coordinated at the two endpoints, it may be beneficial under certain circumstances.

In loosely coordinated neighbor discovery, we relax this requirement. The link can come Up at different times at the two endpoints. The same is true for links going Dn. However, whenever the link goes Up (resp. Dn) at one endpoint, there is a corresponding Up (resp. Dn) event at the other endpoint before a Dn (resp. Up) event occurs at either endpoint. Loosely coordinated neighbor discovery is defined by the following requirement.

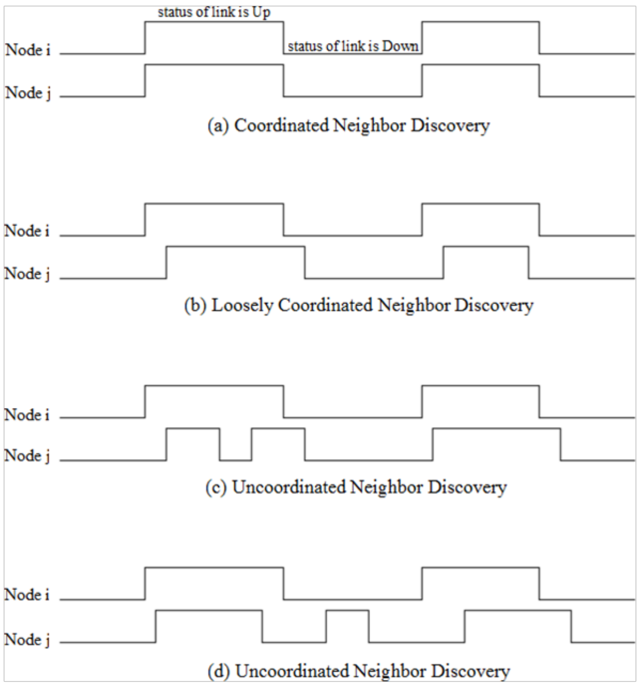

status of link is Up
status of link is Down
Node i
Node j
(a) Coordinated Neighbor Discovery
Node i
Node j
(b) Loosely Coordinated Neighbor Discovery
Node i
Node j
(c) Uncoordinated Neighbor Discovery
Node i
Node j
(d) Uncoordinated Neighbor Discovery

Fig. 3.7.: Different levels of coordination in neighbor discovery.

Definition 14 (Loosely Coordinated Neighbor Discovery). *If a link_up$(j)_i$ (resp., link_down$(j)_i$) occurs at node i at time t_1, then a link_up$(i)_j$ (resp., link_down$(i)_j$) must occur at node j at some time $t_2 > t_1$ before a link_down (resp., link_up) can occur at either node after time t_1.*

This can also be stated in terms of the following conditions:

1. If link (i, j) is *Up* during time interval $[t_1, t_2]$ then link (j, i) cannot go through the states *Up*, *Dn*, *Up* during time interval $[t_1, t_2]$.

2. If link (i, j) is *Dn* during time interval $[t_1, t_2]$ then link(j, i) cannot go through the states *Dn*, *Up*, *Dn* during time interval $[t_1, t_2]$.

Intuitively if there is a *link_up* at one endpoint then there is a corresponding *link_up* at the other end before there is a *link_down* at either end and vice versa.

In uncoordinated neighbor discovery there is no coordination required between the two endpoints of the link. The status of the link can go Up or Dn arbitrarily at the two nodes between which the link is present. Note that even without any coordination, message delivery is guaranteed. That is, if the link is up at node i and it sends a message to node j then this message will be delivered to node j even if the link is down at node j due to *Reliability*. These different levels of coordination are shown in Figure 3.7.

3.5.1 Coordination and BNDP

We now prove that BNDP guarantees loose coordination.

Theorem 10. *BNDP satisfies the loose coordination condition.*

Proof. If link (i, j) is Up during time interval $[t_1, t_2]$ then link (j, i) cannot go through the states Up, Dn, Up during time interval $[t_1, t_2]$. To prove this fix nodes i and j where the directed edges (i, j) and (j, i) are both in the Up. Suppose the edge (j, i) switches to the Dn state at time $t_1 \leq t \leq t_2$ while the edge (i, j) remains Up. The state change $Up \rightarrow Dn$ of edge (j, i) was caused by either a *leave_region* action or by the reception of a *leave* message. It suffices to show that in either case the edge (j, i) can't switch back to Up before the edge (i, j) switches to the Dn state.

If node j executed *leave_region* at time t it will broadcast a *leave* message. Thus by time at most $t + F^{+}_{rcv}$ node i would have received the *leave* message and switched to the Dn state as well, and node j cannot go to state Up before $t + \delta_{LD} > t + F^{+}_{rcv}$.

If node j processed a *leave* message at time t, the message was sent by node i at time t' (where $t \geq t' \geq t - F^{+}_{rcv}$). This means node i became inactive at t' and switched to Dn state. However, link (j, i) can only switch back to Up when node

j receives a *join* or *join_reply* message from node i. Since nodes only send these messages when *active*, this will not happen until $t' + \delta_{LD}$ at the earliest.

If link (i, j) is Dn during time interval $[t_1, t_2]$ then link(j, i) cannot go through the states Dn, Up, Dn during time interval $[t_1, t_2]$. To show this fix nodes i and j where the directed edges (i, j) and (j, i) are both in the Dn state. Suppose the edge (j, i) switches to the Up state at time $t_1 \leq t \leq t_2$ while the edge (i, j) remains Dn. The state change $Dn \rightarrow Up$ of edge (j, i) was triggered by the reception of a *join* or a *join_reply* message from node i. It suffices to show that in either case the edge (j, i) can't switch back to Dn before the edge (i, j) switches to the Up state.

If node j processed a *join* message at time t it was sent by node i at time t' (where $t \geq t' \geq t - F^+_{rcv}$). By time t'' (where $t + F^+_{ack} + F^+_{rcv} \geq t'' \geq t$) node i receives the corresponding *join_reply* message and switches to the Up state (note that $t + F^+_{ack}$ time is required for the batch processing of *join* messages). Node j can only switch back to Dn by either executing a *leave_region* or receiving a *leave*. However, since it sent a *join_reply* at time $t + F^+_{ack}$ it will execute *leave_region* at time $t + \delta_{LU} \geq t + F^+_{ack} + F^+_{rcv}$ at the earliest. Moreover it cannot receive the *leave* message before it is sent by node i, and this won't happen until $t' + \delta_{LU} + L \geq t + F^+_{ack} + F^+_{rcv}$ at the earliest, since node i sent a *join* at time t'.

If node j processed a *join_reply* message at time t it was sent by node i at time t' (where $t \geq t' \geq t - F^+_{rcv}$). Therefore at time t' node i was active and in the Up state, the earliest it can switch to a Dn state is at time $t' + \delta_{LU} \geq t$, but this contradicts the assumption that (i, j) was Dn. $\square$

3.5.2 Coordination and UNDP

Now we show that although UNDP satisfies uniform progress, it does not guarantee even loose coordination. We show this by means of a counter-example, where we assume that $k = 2$. Consider the motion pattern shown in Figure 3.8. The time line for links between nodes i and j is shown in Figure 3.9. Suppose that at time t both nodes i and j are neighbors in one grid partition only (the dotted partition in Figure 3.8) as shown by their original positions in Figure 3.8. Let us call this partition x. Then in a small interval of time the following events occur. First node j crosses the boundary of a another grid partition (the dashed partition in Figure 3.8). Let us call this partition y. Node j sends a *join* message for partition y. Then node j does a $link\_down(i)_j$ for partition x and sends a leave message for partition x, since it is about to cross the boundary of partition x. Node i then crosses the boundary of grid partition y.

After this node i receives the *join* message sent by node j and does a $link\_up(j)_i$ for partition y. Node i then receives the *leave* message sent by node j and does a $link\_down(j)_i$ for partition x. Node i sends a *join_reply* in response to the earlier *join* message. Node j receives this and does a $link\_up(i)_j$ for partition y.

The resulting link status for partitions x and y is shown. The link between nodes i and j remains down in all the other three grid partitions since they are more than k hops apart (for $k = 2$). Combining the outputs of partitions x and y causes a violation of loose coordination.

3.5.3 Impact on Applications

In this section we discuss the impact of coordination on different applications. We consider two different application layer algorithms which assume neighbor discovery

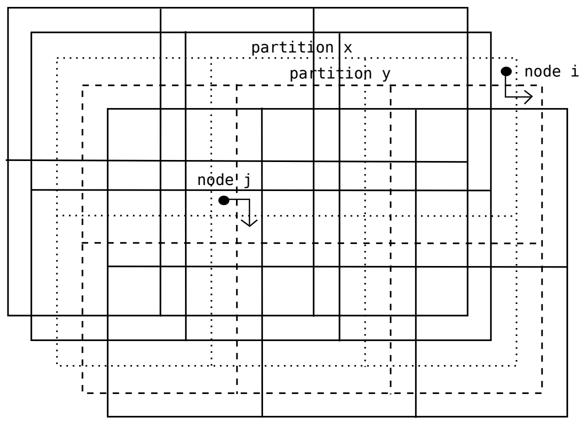


Fig. 3.8.: Counter example for loose coordination in UNDP.

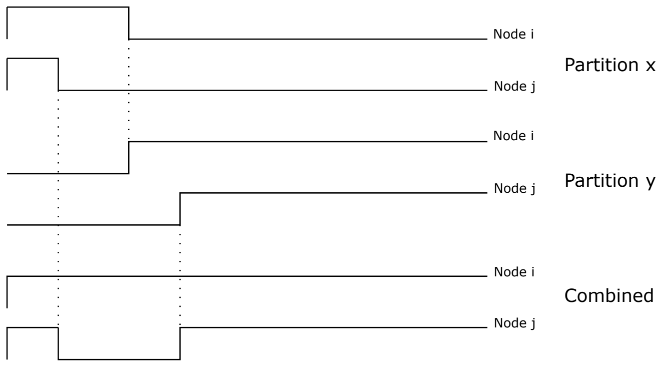


Fig. 3.9.: Time line.

layers with different levels of coordination. In one case it may be shown that although loose coordination is assumed, no coordination is actually required. Hence, it may be beneficial when designing algorithms for mobile ad hoc networks to consider how much coordination is sufficient and necessary.

The first algorithm is a mutual exclusion algorithm presented in [50] which assumes coordinated neighbor discovery. However, closer analysis reveals that coordinated neighbor discovery is not necessary for the correctness of the algorithm. In fact, loose coordination is sufficient. But without loose coordination, mismatched *link_ups* and *link_downs* for the link between two nodes can cause starvation of one of the nodes. Hence, the algorithm may be used with Basic-NDP, at the expense of uniform progress.

The second algorithm is a leader election algorithm presented in [27]. Here the authors assume loose coordination so that nodes may transition appropriately from learning about each other to being neighbors. However, no coordination is actually required, since there is no information lost if mismatched *link_ups* and *link_downs* occur at the two endpoints of a link. The algorithm can be used with UNDP and benefit from uniform progress.

There are also other algorithms which can be used with UNDP without being modified. For instance, the token circulation algorithms presented in [38] assume an uncoordinated neighbor discovery layer.

In conclusion, Basic-NDP provides loose coordination, however, only weak progress is guaranteed. In contrast UNDP provides stronger progress, however, there is no guarantee even of loose coordination between endpoints, which may be useful for certain applications.

4. NEIGHBOR KNOWLEDGE AND COLLISION AVOIDANCE[1]

4.1 Introduction

In this section we focus on the problem of *maintaining* information about neighboring nodes and learning about neighbors as they enter communication range. We present a deterministic solution for nodes in a mobile wireless ad hoc network to communicate reliably and maintain local neighborhood information. The nodes are located on a two-dimensional plane and may be in continuous motion. In our solution we tile the plane with hexagons. Each hexagon is assigned a color from a finite set of colors. Two hexagons of the same color are located sufficiently far apart so that nodes in these two hexagons cannot interfere with each other's broadcasts. Based on this partitioning we develop a periodic deterministic schedule for mobile nodes to broadcast. This schedule guarantees collision avoidance. Broadcast slots are tied to geographic locations instead of nodes and the schedule for a node changes dynamically as it moves from tile to tile. The schedule allows nodes to maintain information about their local neighborhood. This information in turn is used to keep the schedule collision-free. We demonstrate the correctness of our algorithm, and discuss how the periodic schedule can be adapted for different scenarios. The periodic schedule, however, does not address the problem of initial neighbor discovery at start-up. We give a separate algorithm for this problem of initially discovering nodes present within communication range at start-up.

Our work is inspired by that of Ellen *et al.* [18], in which a collision-free schedule is presented for nodes that are restricted to moving along a one-dimensional line

[1]Part of this section is reprinted with kind permission from Springer Science and Business Media, from Viqar, S., Welch, J.L.: Deterministic collision free communication despite continuous motion. In: ALGOSENSORS, pp. 218–229, Copyright Springer-Verlag Berlin Heidelberg 2009. http://dx.doi.org/10.1007/978-3-642-05434-1_22.

(such as along a highway). Their work is particularly relevant for Vehicular Ad Hoc Networks (VANETs) since vehicles often move along highways laid out in straight lines. In that work, the line is partitioned into equal-size segments. The segments are partitioned into a finite number of classes (or colors) in a way that guarantees that simultaneous broadcasts by two nodes in different segments with the same color do not interfere. A schedule is developed that accommodates the node mobility; this schedule is then used to facilitate a scheme to ensure that each node learns about other nodes before they get too close to each other. Their results, however, are not applicable to two-dimensional VANETs with multiple lanes, parallel roads and intersections since the nodes are restricted to one dimension. In this work we no longer restrict nodes to moving on a one-dimensional straight line and consider the general case of nodes moving arbitrarily on a two-dimensional plane. Hence our work is not only applicable to two-dimensional VANETs but also any other type of two-dimensional MANETs.

In our solution we use a combination of Space Division Multiplexing (SDM) and Time Division Multiplexing (TDMA). For the SDM part of our algorithm we consider the plane to be tiled with hexagons. We partition the hexagons into a finite set of colors such that nodes in different hexagons of the same color cannot interfere with each other's broadcasts. The partitioning is shown in a simple example in Figure 4.1. The partitioning allows nodes in different geographic locations to broadcast simultaneously without collisions. We also take into consideration the fact that nodes may be in motion as they broadcast. Based on this partitioning, we develop an efficient periodic deterministic schedule for mobile nodes to transmit which guarantees collision avoidance. The schedule is based on TDMA and ensures that nodes in different colored hexagons never broadcast at the same time. The schedule ensures that every node learns about other nodes before they have entered

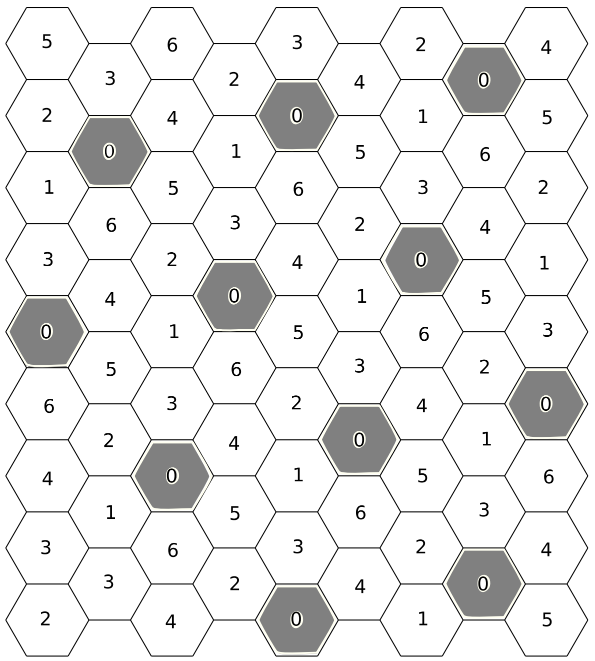


Fig. 4.1.: An example of a partitioning. Hexagons are partitioned into seven classes. Hexagons of the same color or partition broadcast at the same time.

a certain distance inside its broadcast radius. A preliminary version of this work appears in [48].

4.2 Definitions

We consider a set of n nodes which move on a two-dimensional plane. Each mobile node has a unique identifier from a set I. This set is bounded in size. The mobile nodes may fail at any time. We only consider crash failures. For each node there is a trajectory function which specifies the motion of the node by giving the location of the node on the plane at every time. We assume that a node's trajectory remains constant for a certain fixed interval of time (this interval is defined in the next section). The maximum speed of the nodes has an upper bound given by σ.

Communication between nodes takes place through wireless broadcast. The transmission radius of the wireless network is given by R and the interference radius is given by R'. If we consider two nodes p and p' such that p broadcasts and p' remains within distance R of p during the broadcast, then the message sent by p will *arrive* at p'. If there is no other transmitting node within the interference radius R' of p' during the broadcast slot of p the message will be *received* by p' and p' successfully learns the contents of the message.

Each node has access to the current time (through GPS etc.). Hence, its location at a particular time can be determined from its trajectory function. Notice that the presence of a GPS device is a realistic assumption for vehicular ad hoc networks. Nodes begin transmitting at fixed intervals of time. A *broadcast slot* is the time it takes for a node to complete its transmission so that its message arrives at all nodes in broadcast range.

We assume there exists an upper bound on the number of nodes per unit area. This upper bound on the density of nodes is realistic since nodes cannot be infinitely small in size.

4.2.1 Problem Definition

The aim of this work is to provide a deterministic collision-free schedule for mobile nodes such that every node gets infinitely many opportunities to broadcast. This schedule can serve as the Medium Access Control (MAC) layer for mobile ad hoc networks where nodes may be in continuous motion for long periods of time.

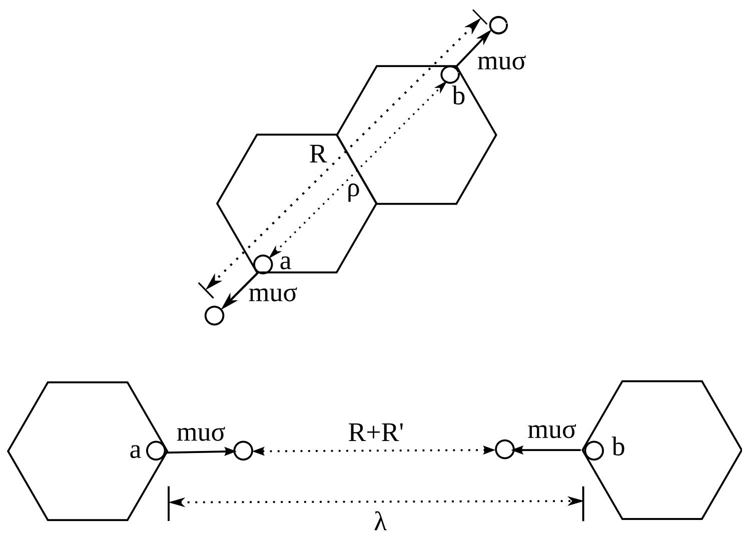


Fig. 4.2.: If a and b are in adjacent hexagons at the beginning of a phase they can receive each other's broadcasts since $\rho + 2mu\sigma \leq R$. On the other hand if a and b are in hexagons allocated the same slot they can broadcast without collision since $\lambda - 2mu\sigma \geq R + R'$.

4.3 Algorithm Overview

In our solution we assume that the plane is tiled with hexagons. Our choice of hexagons is based on two factors. Hexagons can form a regular tiling of the plane, and they give a good approximation of the circular broadcast range of wireless nodes.

In our algorithm (see Algorithm 1) mobile nodes are dynamically scheduled to broadcast depending on the geographic location of the tile they occupy at a particular instant of time. The size of these hexagonal tiles depends on the broadcast radius R of the mobile nodes. Roughly we require that R spans a little more than two tiles. This ensures that nodes in adjacent hexagons are within each others' broadcast radius.

A set of m contiguous hexagonal tiles are grouped together to form a *supertile*. Each tile in a supertile is assigned a different color. These supertiles also tile the plane. Corresponding hexagons which lie at the same position in two different supertiles share the same color and are scheduled to broadcast simultaneously. By carefully selecting the number of tiles m in a supertile and its shape, we ensure that tiles of the same color in adjacent supertiles are located far enough apart, so that nodes in these tiles can broadcast simultaneously without causing a collision at any receiving node. Note that collision-freedom is ensured despite the fact that the nodes maybe in continuous motion while they broadcast.

The choice of m depends on the actual values of the broadcast radius R, the interference radius R', and the upper bound σ on the maximum speed of the nodes. Roughly, the supertile should be large enough so that nodes in tiles assigned the same color remain more than $R + R'$ apart, even if they are moving straight toward each other. We assume that the tiling of the plane and the assignment of colors to tiles is predetermined and known to all the mobile nodes in advance. Tiling the plane in this way is a form of space division multiplexing (SDM) since the mobile nodes are separated in space to prevent interference. In addition to this, we perform Time Division Multiplexing inside the hexagons and the supertiles. A fixed number of broadcast slots (given by u) are grouped together to form a *round*. Each round corresponds to one hexagon, and is the time allocated to all the nodes in one hexagon to schedule their broadcasts. In order to cover all the hexagons in one supertile we then require m rounds; one for each color. We define this as one *phase*. The length of one phase is then equal to mu broadcast slots. The assignment of the m rounds in a phase to different colored tiles forms an ordering of the colors with respect to time. Note that the ordering of colors can change from phase to phase depending on the *schedule*. In Section 4.5 we discuss different types of schedules.

Slots are allocated to mobile nodes only at the beginning of every phase. Hence at the start of every phase a mobile node determines the color of the hexagonal tile it is located in. It can then determine which one of the m rounds in that phase it should broadcast in. Furthermore, at the start of every phase, a mobile node possesses knowledge about all other nodes present in its own tile (owing to the maintenance of local neighborhood knowledge). Specifically it is aware of the identifiers of these nodes. Based on the rank of its own identifier in this set of identifiers, it can select one of the u slots in its round, to carry out its broadcast. Hence, at the start of a phase, every node in every tile knows exactly which slot to broadcast in. Note that we assume that the maximum number of nodes that can occupy a tile at any instant is bounded by $v < u$. This allows us to have fixed length rounds and phases. The first v slots of a round are used by the nodes to perform broadcasts. The remaining $u - v$ slots can be used by other protocols or applications.

As mentioned earlier, the size of the tiles and supertiles is also influenced by σ, the upper bound on node speed, because σ determines the maximum distance that a node can travel in one phase. Since the length of a phase is mu slots, this distance is given by $mu\sigma$. We require that R should be larger than the diameter of two tiles by at least $2mu\sigma$. We also require that tiles of the same color be separated at least by $R + R'$ in addition to $2mu\sigma$. Suppose that a node moves out of its tile before its turn to broadcast in a phase. These constraints will ensure that its broadcast still reaches its neighbors, without causing a collision. In essence, the broadcasts of all the nodes in one supertile are separated in time and cannot interfere. Only nodes present in all tiles of the same color throughout the plane broadcast simultaneously. However, these nodes are always sufficiently separated and cannot interfere with each others' broadcasts.

The pseudocode for the algorithm is given below. The function $clock()$ returns the current time, and $location()$ returns the current location of the node. The function $findColor()$ takes as argument the location of the node. It uses the fixed division of the plane into tiles and supertiles to determine which color tile the node is located in. A node includes its trajectory function in its broadcast packet so that neighbors can calculate the location of the node at the beginning of the next phase. We assume that a node's trajectory remains unchanged for the duration of at least one phase. This is to ensure that neighbors can have up to date information about a node's trajectory until it broadcasts new information.

Algorithm 1 Code for node p

```
$id$ {node $p$'s id}
$trajectory$ {$p$'s trajectory}
$N$ {set of "neighboring" nodes; initially contains all nodes within $p$'s own and adjacent hexagons; each entry $q$ consists of $q.id$ and $q.trajectory$}
S {set of nodes that might become neighbors; initially empty; candidates are collected during each phase}
**when** $receive\ a\ message(id, trajectory)\ from\ node\ q$
    $S := S \cup < id, trajectory >$
**when** $clock() = \pi mu$, for some integer $\pi$ {the beginning of phase $\pi$}
    $N := \emptyset$ {N is the set of neighbors of a node}
    $loc := location()$ {$(x, y)$ coordinates, based on $p$'s trajectory and current time}
    $hex := findHex(loc)$ {calculate hexagon containing loc}
    $color := findColor(hex)$ {calculate color of hexagon(expressed as an integer)}
    $\forall y \in S$ {update neighbor set}
        **if** $(y.trajectory(\pi mu) \in Hex(loc))$ {determine neighbors}
            $N := N \cup y$ {N is the set of neighbors}
    $i := getRank(N)$ {get rank of p's id in set N; smallest id in N has rank 1, etc.}
    $slot := color + (i - 1)$ {this is the slot to broadcast in}
    $S := \emptyset$
**when** $clock()\ mod\ mu = slot$ {time to broadcast}
    $broadcast(id, trajectory)$
```

4.4 Analysis

4.4.1 Collision Avoidance

We require the following constraint (C1) for collision avoidance (see Figure 4.2).

C1. Let the minimum distance between simultaneously transmitting hexagons be λ . Then we require $\lambda - 2mu\sigma \geq R + R'$.

The following lemma shows that under constraint (C1) the algorithm ensures collision avoidance. Consider two nodes present right at the boundary of different hexagons of the same color at the beginning of a phase. They are separated only by the minimum possible distance (λ). Even if they are moving directly toward each other throughout the current phase, their broadcasts will not cause a collision. Hence, a node can cross the boundary of its original hexagon during a phase safely (without causing collisions). This property is maintained from phase to phase.

Lemma 1. *If (C1) holds then every broadcast that arrives at a node is received.*

Proof. Suppose that this is not the case. Then there are three distinct processors p, q and r and a broadcast slot j such that p and r broadcast during slot j and q is within distance R of p throughout slot j, and q is within distance R' of r at some time during slot j. Thus at some time during slot j, processors p and r are at most distance $R + R'$ apart. Let j' be the first slot in the phase that contains slot j. At the beginning of slot j', processors p and r are more than distance λ apart. This is because they have been assigned the same broadcast slot in the same round and hence, they are in hexagons which transmit at the same time. By assumption such hexagons are separated by distance λ. From the beginning of slot j' until the end of slot j is $(j - j' + 1)u \leq mu$ units of time. During this period of time they can each travel distance at most $mu\sigma$, and so they are more than distance $\lambda - 2mu\sigma \geq R + R'$ apart during slot j. This is a contradiction. □

4.4.2 Maintenance of Neighborhood Knowledge

We assume that at start-up all nodes have information about nodes in their own hexagon and in adjacent hexagons, that is all nodes know the trajectory function of nodes that are within their own hexagon or in adjacent hexagons. This is stated in assumption (A1).

A1. At the beginning of phase 0, every node knows the id and trajectory of every other node in its own hexagon and in adjacent hexagons.

Later in Section 4.7 we show how to relax this assumption.

We introduce constraint (C2) in order to ensure the maintenance of neighborhood knowledge (see Figure 4.2). Lemma 2 shows that nodes maintain knowledge about nodes in their own and adjacent hexagons.

C2. Let the distance between the farthest points on the boundary of adjacent hexagons be equal to ρ. We require that $\rho + 2mu\sigma \leq R$.

Lemma 2. *If assumption (A1) and constraints (C1) and (C2) hold, then at the beginning of each phase π ($\pi \geq 0$) of Algorithm 1 every node knows about every node that is in its own or an adjacent hexagon.*

Proof. By induction on π.

Basis : $\pi = 0$; By assumption (A1) and constraint (C2) the lemma holds for phase 0.

Induction: let us assume the lemma holds for π and prove it for $\pi + 1$.

Consider two nodes p and q that are not in adjacent hexagons at the beginning of phase π. p is in hexagon $h1$ and q is in hexagon $h2$ at the beginning of phase π. Suppose that at the beginning of phase $\pi + 1$ they move into adjacent hexagons. We need to prove that p learns about q by the end of phase π. Similar arguments hold for q learning about p. Since p and q can only cover distance $mu\sigma$ in phase π, at worst they must be distance $\rho + 2mu\sigma$ apart, at the beginning of phase π, in order to become neighbors in phase $\pi + 1$.

By constraint (C2) q will remain within distance R of node p till the end of phase π, since it can only cover distance $mu\sigma$ during phase π. Hence, by Lemma 1 p will receive its broadcast and learn about node q before the end of phase π. $\square$

4.5 Schedules

A *schedule* defines the order in which the rounds are allocated to different colored tiles in a supertile. A schedule can span multiple phases, and each phase can have a different ordering of the m colors. We define a schedule to be *periodic* if the sequence of colors repeats after a fixed number of phases. For a particular execution of the algorithm the schedule for all supertiles is the same and known *a priori*. In this section we present a general framework for schedules in terms of liveness, fairness, and directional bias; these terms are defined subsequently. We also discuss the advantages of particular schedules through examples.

A particular execution of the algorithm is defined to be *safe* if no collisions occur during the entire execution. The constraints that we have discussed so far ensure safety. In particular the distribution of colors on the plane avoids inter-tile collisions, whereas neighborhood knowledge together with the TDMA performed in each round prevents intra-tile collisions. In order to ensure *liveness* during an execution we require the following condition:

- Every color present in a supertile is allocated at least one round in the schedule.

Furthermore in order to ensure *fairness* we require the following:

- Each color is allocated the same number of rounds in the schedule.

A schedule is defined to have *directional bias* if it favors the propagation of information in one particular direction. We start by considering schedules with a time

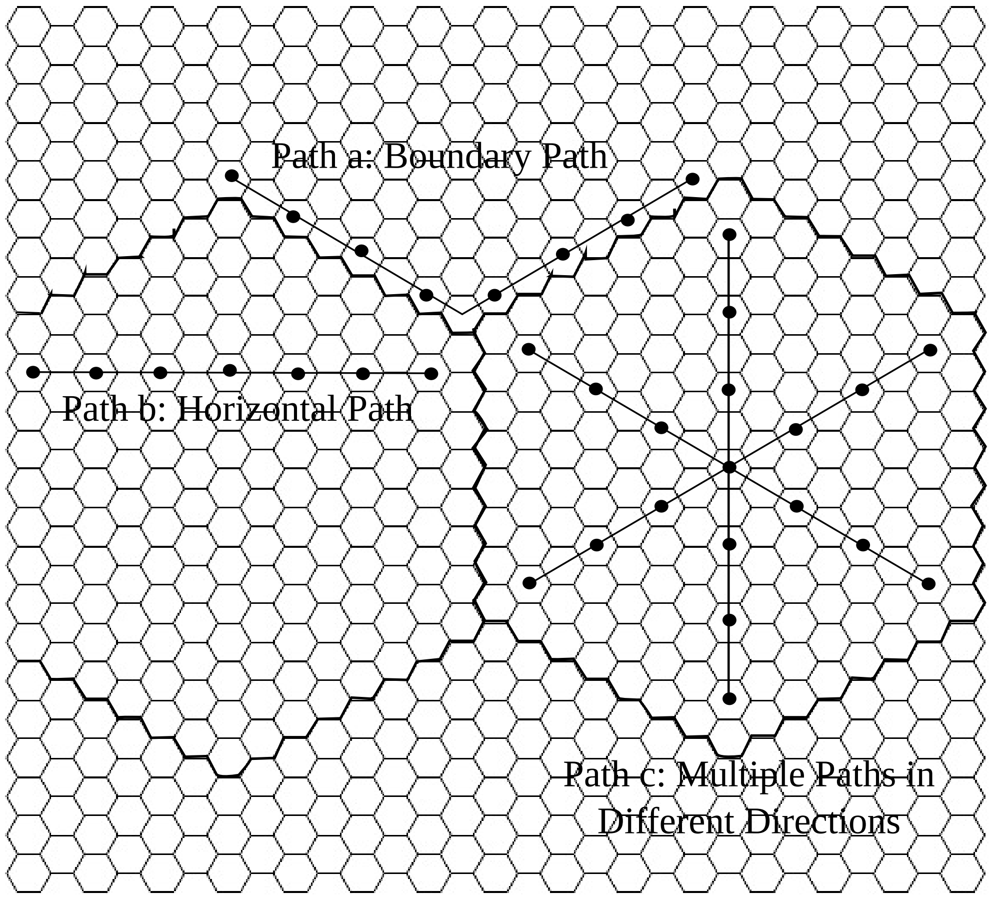


Fig. 4.3.: Information should flow on all paths without directional bias.

period equal to one phase. In such schedules each color is allocated exactly one round during one phase and the sequence of colors is the same for every phase. However, such schedules can suffer from directional bias. The following example illustrates this. Suppose that we have a schedule in which slots are allocated from left to right and top to bottom in a supertile in one phase. This schedule is biased in favor of propagating information rightwards. A similar argument holds for information traveling downwards. Consider the horizontal *path b* shown in Figure 4.3. Suppose that information has to travel on this path from one node to another in the rightward direction. The information will propagate at a rate of one supertile per phase. Now suppose that the information has to travel leftwards. This will occur at a rate of only one hop per phase which is considerably slower. So the schedule is biased in favor of the information traveling rightwards and downwards. This shows that a schedule which is optimal for a certain path may perform poorly for some other path.

We can modify the schedule depending on the requirements of the application. For example for VANETs with parallel roads we can have alternating left to right and right to left allocation of rounds along each road, depending how the road traverses a supertile. Instead of delving into the requirements of particular applications, however, we construct a generalized schedule which is neutral in terms of directional bias.

In order to avoid the above mentioned directional bias we consider cases where the time period is more than one phase. Hence, the order in which rounds are allocated to colors can change from phase to phase. We seek a schedule that is efficient and that is not biased toward speeding up information propagation in one direction at the expense of another direction. From the above examples it is clear that we should alternate between all directions in order to have an unbiased schedule. Consider the paths in example c shown in Figure 4.3. The schedule should favor propagation along all six paths from the center equally. The most obvious way this can be achieved is if we allocate slots in a circular fashion. We can allocate slots in a supertile in concentric circles, starting from the center of the supertile going toward its boundaries, in one phase. This enables the propagation of information from the center of the supertile toward its boundaries. In the next phase we can allocate slots starting at the boundaries of the supertile, going toward the center. This will facilitate the propagation of information toward the center of the supertile.

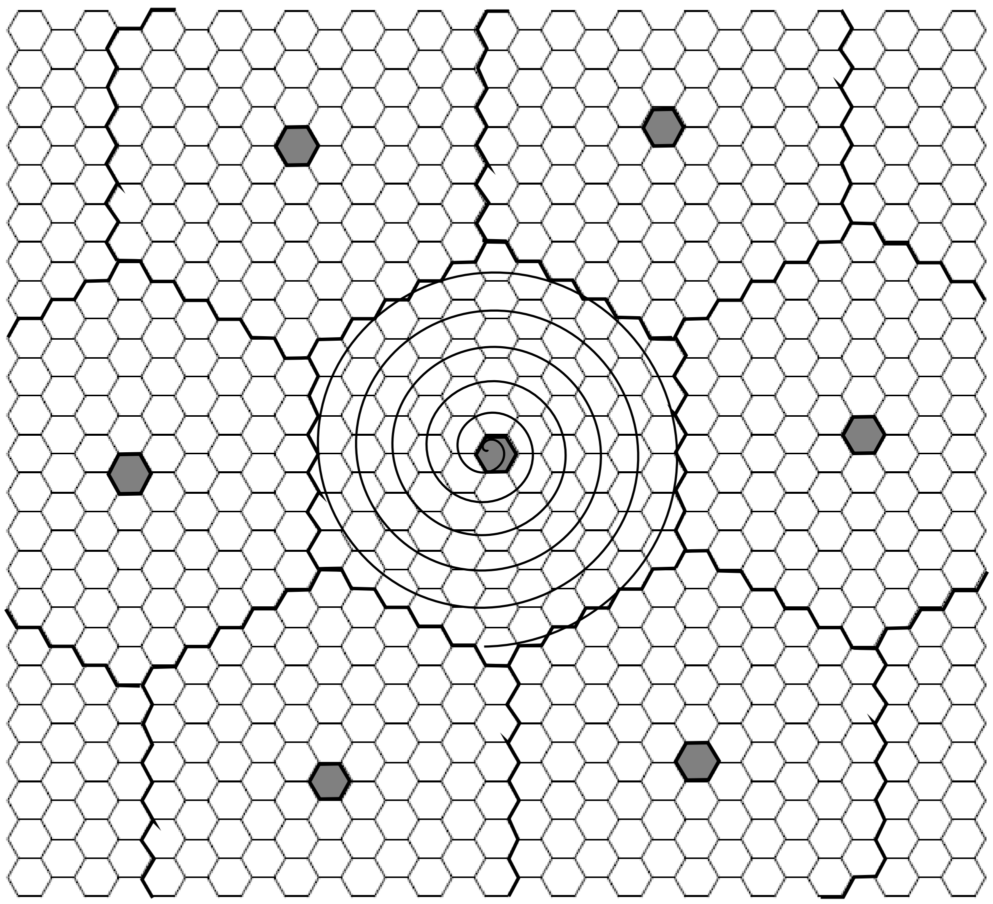


Fig. 4.4.: An example of the network division into regions where m=91.

Finally the phases alternate between anticlockwise phases and clockwise phases. The motivation for this is demonstrated by *path a* in Figure 4.3. The clockwise phase will allow efficient propagation from right to left whereas the anticlockwise phase will allow efficient propagation from left to right. The resulting schedule consists of a allocation of broadcast slots in concentric circles in which the hexagons take turns in one phase going in a clockwise manner outwards and in the next phase going anticlockwise inwards. The next two phases follow an anticlockwise outwards pattern and a clockwise inwards pattern. The periodicity of this final schedule is four phases. This schedule is depicted in Figure 4.4.

4.5.1 An Example of the Network Tiling

In this section we show that the schedule developed above is practically feasible, under the given constraints. We use the default values for R (the broadcast radius) and R'(the interference radius) given in the IEEE 802.11 standard [24] which are 250 meters and 550 meters respectively. Suppose we take the duration of a phase to be equal to 100 milliseconds, and σ to be equal to 200 km per hour. This is a reasonable assumption for VANETs. Using these values, we can show that the maximum distance that a node can travel in one phase is 5.55 meters $\leq$ 6 meters, which we take as the value of $mu\sigma$. We want to maximize the size of the hexagons so that we can have fewer hexagons in between two hexagons which are allocated slots at the same time. This will allow us to minimize m and have fewer broadcast slots in one phase. Hence, taking $\rho + 2mu\sigma = 250$ meters we calculate the side of a hexagon (given by s) to be equal to 65.8 meters. From constraint (C1) we know that $\lambda \geq R + R' + 2mu\sigma = 250 + 550 + 12 = 812$ meters. The distance between two hexagons of the same color has to be greater than or equal to $\lambda = 812$ meters. To calculate the distance in terms of s the side of a hexagon, we divide by $s = 65.8$, hence $\lceil 812/65.8 \rceil = 13$. Therefore the distance between hexagons of the same color should be greater than or equal to 13 times the side of a hexagon. An example of such a tiling of the network is given in Figure 4.4. Here each region consists of 91 hexagons, hence $m = 91$. This tiling is formed with regions which consist of concentric rings of hexagons and may not be optimal in terms of m for the given values of R and R'. However, the shape of the regions is relatively regular, which

Schedules	Average Number of Rounds
Spiral	87.64828
Left to Right	293.16553
Random	275.25516

Fig. 4.5.: Average number of rounds for information propagation on paths between all node pairs on the boundary tiles of a supertile.

makes it easier to formulate a schedule and perform analysis. Note that using the final schedule given in Section 4.5 each node gets to transmit once during a phase.

4.6 Simulation Results

We performed a simulation for the comparison of the speed of information propagation for different schedules. We considered information propagation on paths between all node pairs on the boundary tiles of one supertile (of the same size as shown in Figure 4.4). In order to compare different schedules we assumed that there is at least one correct node in each hexagon at all times and nodes remain static. We considered three different schedules: random, left to right and spiral (as described in Section 4.5).

The results show that the spiral schedule has on average lower propagation delay in terms of rounds (see Figure 4.5). Thus information can propagate faster (on average) across supertiles using this schedule.

The results show that the maximum propagation delay is 125 rounds for the spiral schedule, whereas for the other schedules it goes as high as 850 rounds, as shown in Figure 4.6 . For some paths the random and let to right schedule perform better than the spiral schedule, however, on average the spiral schedule requires fewer number of rounds.

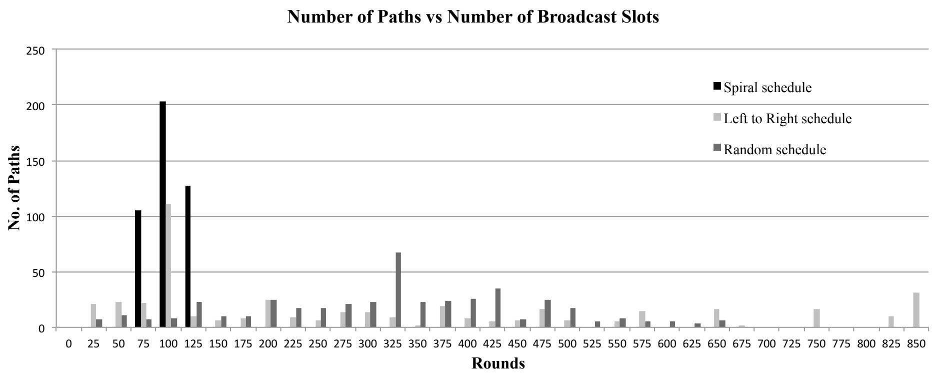


Fig. 4.6.: Comparison of information propagation on paths between all node pairs on the boundary tiles of a supertile.

4.7 Initializing Nodes at Start-Up

Assumption (A1) about initial neighbor knowledge, which is required for Algorithm 1, is quite restrictive. It states that at the beginning of phase 0, every node knows the id and trajectory of every other node in its own hexagon and in adjacent hexagons. We define the terms *neighbors*, and *neighborhood* with respect to a node p to mean other nodes in its own hexagon and in adjacent hexagons at a particular instant of time. In this section we give an initialization algorithm for gaining initial knowledge about neighbors, so that assumption (A1) can be relaxed. This initialization algorithm runs during an initialization phase (see Figure 4.7). After this phase is over, nodes start executing Algorithm 1, for neighbor knowledge *maintenance*. Instead of assumption (A1) we now require the following constraints on nodes:

A2. For simplicity we assume that nodes start up at the same time. A discussion on how to deal with nodes that wake up at different times is given later.

A3. We assume that nodes remain within their original hexagon during the initialization phase.

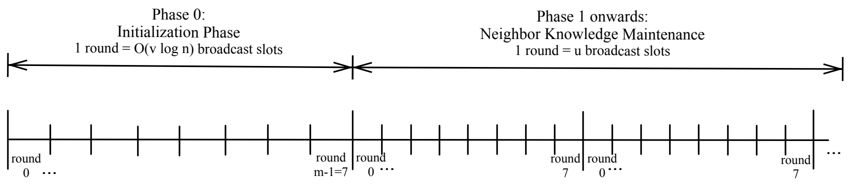


Fig. 4.7.: Initialization phase and neighbor knowledge maintenance (assuming $m = 8$).

Throughout this section, for simplicity we consider initial knowledge to include only the ids of nodes, even though assumption (A1) refers to both ids and trajectories. We assume that the trajectory function of a node may be expressed succinctly and it may be appended to the id when a node sends a message.

Our initialization algorithm is based on a previous deterministic algorithm for gossiping (all-to-all communication), given in [22]. This previous algorithm from [22] is complicated and deals with arbitrary network topologies. We modify this algorithm and make it simpler and more efficient, since we deal only with neighboring nodes forming a clique, instead of arbitrary topologies.

The time complexity of the algorithm given in [22] is $O(v \log^2 n \log^2 v)$, where v is the total number of nodes present in the network and n is the size of the name space of the ids. The model of wireless broadcast given in [22] assumes that the actual number of nodes that participate in the protocol is much smaller than the name space of node identifiers. This assumption fits in with our model since there can be at most v nodes in one hexagon at one instant of time and v is much smaller than the size of the name space n.

The efficiency of the algorithm given in [22] algorithm lies in the fact that the worst case time complexity is polylogarithmic in n rather than linear in n. Our modifications for the case of a clique result in a more efficient $O(v \log n)$ time initialization algorithm.

4.7.1 Definitions

Our initialization algorithm is based on using *selective families* (a variant of superimposed codes [25]). These *selective families* are families of subsets of $\{1, 2, ..., n\}$. Before we describe the initialization algorithm in detail, here are some definitions:

Definition 15. *A set S* hits *a set X if and only if $|S \cap X| = 1$.*

Definition 16. *A set S* avoids *a set X if and only if $S \cap X = \emptyset$.*

Definition 17. *Given positive integers n and v where $v < n$, a family $\mathcal{S}$ of subsets of $\{1, 2, ..., n\}$ is a v-*selector *over $\{1, 2, ..., n\}$, if for any two disjoint sets X, $Y \subseteq \{1, 2, ..., n\}$ with $v/2 \leq |X| \leq v$ and $|Y| \leq v$, there exists a set in $\mathcal{S}$ which hits X and avoids Y.*

It has been shown in [12] that for each pair of positive integers n and v where $v < n$, there exists a v-selector over $\{1, 2, ..., n\}$, of size $O(v \log n)$ (i.e. $|\mathcal{S}| = O(v \log n)$, where $\mathcal{S}$ is the family of subsets which forms the v-selector).

The authors in [22] show the following (Lemma 1 in [22]) :

Lemma 3. *Let $\mathcal{S}$ be a v-selector over $\{1, 2, ..., n\}$ and let V be any subset of $\{1, 2, ..., n\}$ such that $v \leq |V| \leq 2v$. Let Y be the set of all elements $y \in V$ such that there exists a set Z in $\mathcal{S}$ which hits V on y, i.e., $Z \cap V = \{y\}$. Then Y contains more than half of the elements of V.*

Selectors may be used to provide some guarantees on how many nodes in a neighborhood can transmit without message collisions. In the context of radio broadcasts, the subsets of a selector can be thought of as forming a transmission schedule for the nodes. The subsets can be arranged in an arbitrary order which is fixed and known to all the nodes. Suppose this order is given by $(s_1, ..., s_m)$ where each s_i is a set of node ids. Then for $i = 1, 2, ..., m$, at step i, all the nodes with ids in s_i transmit their message.

Lemma 3 may be applied to our problem as follows. There can be no more than v nodes in one hexagon at one time. Let V be the set of ids of the nodes in a hexagon; we know that $|V| \leq v$. Let V' be any subset of the n ids such that $|V'| = v$ and V is a subset of V'. The v-selector then contains more than $|V'|/2$ sets each of which contain exactly one id from V'. If S is one of the sets that contains exactly one id from V', say x, then when the schedule instructs all nodes with ids in S to broadcast, the node with id x, if present in the hexagon, transmits alone. In this way, if there are initially $v = |V| \leq 2v$ nodes in the hexagon, then a v-selector can be used, and more than half of the nodes in the hexagon will be able to transmit alone. However, nodes that transmit alone cannot detect that they have done so. These nodes must be informed that they should stop participating in the schedule in order to allow other nodes the chance to transmit their ids. If there are fewer than v nodes in the hexagon initially then the guarantees of Lemma 3, regarding more than half the nodes transmitting alone, do not hold. In this case the size of the selector must be adjusted to match the number of nodes present.

4.7.2 An $O(v \log n)$ Time Algorithm

In this section we describe our $O(v \log n)$ time initialization algorithm which allows nodes to gain initial knowledge about the ids of nodes within their neighborhood. For simplicity we assume that v is a power of two.

Algorithm 2 Initialization Algorithm: code for node p

```
$id$  {node $p$'s id}
$id\_list$ {list of node ids received so far and $p$'s own id}
$leader := \perp$  {the id of the leader node in an iteration}
$total\_slots$  {the total number of broadcast slots which is $O(v \log n)$}
$passive := false$  {flag to stop broadcasting id, initially false}
$\mathcal{S}^0, \mathcal{S}^1, ..., \mathcal{S}^{\log_2 v}$ {each $\mathcal{S}^i$ is a $v/2^i$-selector of size $O((v/2^i) \log n)$, $S^i_j$ is the $jth$ set in $\mathcal{S}^i$}

1: for $i = 0$ to $\log_2 v$ do {there are $\log_2 v + 1$ iterations}
2:     $leader := \perp$
3:     for $j = 1$ to $|S^i|$ do
4:         if $id \in S^i_j$ & $passive = false$ then
5:             $broadcast(\langle id, leader \rangle)$
6:         else {either not scheduled to broadcast or $passive = true$, so listen}
7:             if $receive\ a\ message(\langle id_q, leader_q \rangle)\ from\ node\ q$ then
8:                 $id\_list := id\_list \cup \{id_q\}$
9:                 if $leader = \perp$ then {this is the first message received during current iteration}
10:                    if $id = leader_q$ then {if other nodes have set $leader$ to $p$}
11:                        $leader := id$ {$p$ sets itself as the leader}
12:                    else
13:                        $leader := id_q$ {$leader$ now contains the id of sender of the message}
14:    if $id = leader$ then
15:        $broadcast(id\_list)$ {the leader sends all ids at the end of this iteration}
16:        $passive := true$ {if leader then it becomes passive}
17:    else if $receive\ a\ message(id\_list_q)\ from\ node\ q$ then
18:        $id\_list := id\_list \cup id\_list_q$
19:        if $id \in id\_list_q$ then
20:            $passive := true$ {if a node receives its own id then it becomes passive}
```

The initialization algorithm runs in the the initialization phase, consisting of m rounds. Each round consists of $O(v \log n)$ broadcast slots (see Figure 4.7). Nodes in each hexagon run their own instance of the initialization algorithm, separated in time from nodes in neighboring hexagons. Each hexagon in a supertile has a separate round assigned to it during the initialization phase, for this purpose (see Figure 4.7). In this way there is no interference between the transmissions of nodes in different hexagons. If a node p is located in a hexagon of color c_i, where $0 \leq i < m$ during the initialization phase, then it runs the code given in Algorithm 2 during round i of the initialization phase. During other rounds of the initialization phase, node p listens to the transmissions of nodes in other hexagons, and collects their ids. If a node q in an adjacent hexagon transmits without collision during a round $j \neq i$ then p successfully receives its message and gets to know its id. This is because $\rho + 2mu\sigma \leq R$ due to (C1), where ρ is the distance between the farthest points on the boundary of adjacent hexagons.

We now describe an iterative procedure that is carried out during each round. This procedure requires $\log_2 v + 1$ iterations. In the first iteration we use a v-selector to schedule node transmissions. In each subsequent iteration, a smaller and smaller selector is used to schedule node transmissions. Specifically, in each iteration i, where $0 \leq i \leq \log_2 v$, a $v/2^i$-selector is used (lines 3 and 4 of Algorithm 2).

Let *active* nodes be defined as those nodes for which *passive* is false in Algorithm 2. These are nodes which have not yet transmitted their ids successfully without collision, or they have done so but do not know about it (since nodes cannot detect whether their own transmissions are received by other nodes). During each iteration, some active nodes may be able to transmit alone without collision due to the properties of the selector used to schedule transmissions. Specifically, if during iteration

i there are x active nodes such that $v/2^i \leq x \leq v/2^{i-1}$, then more than half of these nodes will transmit without collision due to Lemma 3.

Let *passive* nodes be defined as those nodes for which *passive* is true in Algorithm 2. If a node gets to broadcast without collision during iteration i, and it detects that it has done so by receiving confirmation from other nodes, it becomes a passive node and does not participate in subsequent iterations. The number of active nodes is reduced as the iterations progress. We show that eventually, after $\log_2 v$ iterations, the number of active nodes is reduced to one or less. After this point in time, just one additional iteration is required after which all nodes in the hexagon have transmitted their ids successfully and gained knowledge about each other's ids.

Since transmitting nodes cannot detect whether their own transmissions are received by other nodes, in order to inform these nodes to stop participating we employ a leader election mechanism. The leader node informs all nodes whose ids have been transmitted successfully in the current iteration to stop participating so they can transition from active to passive. It does so by transmitting their ids in a designated leader slot (lines 14-20 of Algorithm 2). These nodes, upon receiving their ids from the leader, change from active to passive and do not perform any more transmissions. After the leader node performs its transmission, the remaining active nodes run a selector of smaller size. This process is carried out iteratively.

Leader election is carried out in each iteration simply by electing the first node whose id is successfully received in that iteration. Suppose that the id of this node is w_0. For all nodes other than w_0, this id is received in the slot in which w_0 transmits alone during the current iteration. Nodes other than w_0 then include w_0's id in the messages they transmit. For w_0 itself, it receives its own id the second time a node transmits alone during the current iteration. Hence, a node which receives its own id in the first message that it receives successfully during the current iteration, knows

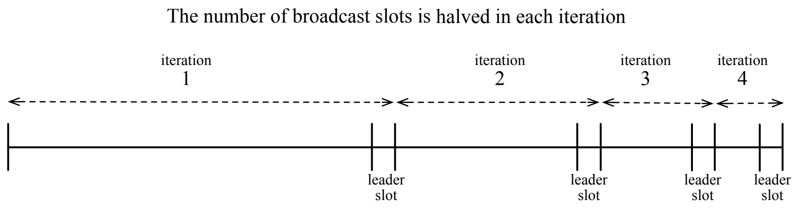


Fig. 4.8.: One round of the initialization phase; total number of broadcast slots is $O(v \log n)$.

that it is the leader. Node w_0 is then allocated a broadcast slot at the end of the current iteration of the schedule, where it sends all the ids it has collected (see Figure 4.8).

The algorithm described above is non-constructive because of the use of selectors; however, in [26] the authors present an explicit construction of v-selectors of size $O(v \log^{O(1)} n)$. Using this explicit construction, our algorithm can be made constructive with a slowdown of $O(\log^{O(1)} n)$.

We now show that by the end of Algorithm 2, the *id_list* of a node contains the ids of all nodes in its own hexagon.

Lemma 4. *In each iteration of the initialization algorithm if two or more nodes transmit without collisions then one node is elected as the leader, otherwise no node is elected as the leader.*

Proof. The *leader* variable is reset at the beginning of each iteration of the algorithm (line 2 of Algorithm 2). The id in the first message received successfully by a node at the start of an iteration is assigned to the *leader* variable (line 9 of Algorithm 2). For a particular node p there are two possibilities. Suppose node p was not the first node to transmit without message collision in iteration i. In this case it will receive the message of the first node to transmit successfully and store its id in the *leader* variable (line 13 of Algorithm 2). After this no change will be made to *leader* during iteration i. On the other hand suppose node p was the first to transmit successfully during iteration i. Then the second node to transmit successfully will include p's id in its message (line 5 of Algorithm 2). p will then receive its own id and set itself as the leader for iteration i (line 11 of Algorithm 2).

If node p is the only node to transmit successfully during the current iteration, then it will not receive any messages, and no leader is elected.

If no node transmits successfully during the current iteration, then no messages will be received at any node, and no leader is elected. $\square$

Lemma 5. *In iteration i of the initialization algorithm if $x \geq 2$ nodes transmit without collisions then we have the following:*

1. *For all nodes p present in the hexagon, id_list_p contains the ids of all these x nodes.*

2. *These x nodes become passive after iteration i.*

Proof. In iteration i of the initialization protocol if two or more nodes transmit without collision then by Lemma 4 one of these nodes is elected leader during that iteration. At the end of iteration i the leader successfully receives the ids of all nodes that transmit without collisions. During the leader slot after iteration i, the leader transmits the entire list of ids it has learnt so far (line 15 of Algorithm 2). Since the leader elected during iteration i alone transmits during the leader slot after iteration i, all nodes in the hexagon successfully receive the entire list of ids transmitted by the leader. The leader then becomes passive (line 16 of Algorithm 2).

If a node finds its own id in this list, it becomes passive (refer to line 20 of Algorithm 2). Hence, all x nodes that transmitted successfully during iteration i become passive. $\square$

Lemma 6. *If a node p becomes passive then $p \in id\_list_x$ for all nodes x present in the hexagon occupied by p.*

Proof. A node p becomes passive by setting *passive* to true in line 16 or line 20 of Algorithm 2. Suppose this occurs in iteration i. This means node p was either the leader in iteration i, or it received id_list_q and p's id was in id_list_q. If p was the leader in iteration i this means it received a message earlier during iteration i from another node containing p's id (line 7 of Algorithm 2). In both cases p received a message from another node containing its own id. This means at least one other node, say q, in p's hexagon has p's id in its *id_list*. This in turn means that p transmitted alone in the hexagon at some previous point in time. Since all the nodes in a hexagon form a clique, if p transmitted alone and q received its message then all other nodes in the hexagon also received p's message during the same transmission. Hence, for all nodes x in the hexagon $p \in id\_list_x$. $\square$

Lemma 7. *If only one node remains active at the beginning of the last iteration then it transmits alone during the last iteration and all other nodes in the hexagon learn its id.*

Proof. In the last iteration a $(v/2^{\log_2 v} = 1)$-selector is used. If only one active node is left during this iteration, then due to Lemma 3, it transmits its id alone. All other nodes learn its id, even though this node itself does not receive confirmation of this. $\square$

Theorem 1. *If there are v or fewer nodes in a hexagon initially then at the end of the initialization algorithm, for all nodes p in the hexagon, id_list_p contains the ids of all nodes present in the hexagon.*

Proof. If a node p is alone in the hexagon, then id_list_p is initialized so that it contains the id of p.

Now we are left with the case where there are originally $x \geq 2$ nodes initially in the hexagon. In this case if only one active node remains at the beginning of the last iteration, this last remaining active node may or may not transition to a passive state, however, due to Lemma 7, other nodes learn its id. Note that all other nodes have already transitioned to being passive, hence, by Lemma 6 all nodes in the hexagon know each other's ids.

Now suppose there are two or more active nodes at the beginning of the last iteration. We will show that this leads to a contradiction using an inductive argument. Let $active^i_{beg}$, and $active^i_{end}$ be the number of active nodes at the beginning and end of iteration i, respectively. Note that:

$$active^i_{end} = active^{i+1}_{beg} \tag{4.1}$$

Claim 1. *If there are two or more active nodes at the beginning of the last iteration, then for all rounds $0 \leq i \leq \log_2 v$ we have $active^i_{beg} \geq v/2^{i-1}$.*

Claim 1 implies that for $i = 0$:

$$active^0_{beg} \geq v/2^{0-1} = 2v$$

This is a contradiction, since there cannot be more than v nodes in a hexagon initially.

We now prove Claim 1. The proof is by induction on i.

(*Basis:* $i = \log_2 v$)

By assumption, at the beginning of the last iteration there are at least two active nodes:

$$active_{beg}^{\log_2 v} \geq 2$$

Hence, $active_{beg}^{i} \geq v/2^{i-1}$ is true since

$$v/2^{\log_2 v-1} \geq 2$$

(*Inductive Case*)

Assuming $active_{beg}^{i} \geq v/2^{i-1}$ we show that $active_{beg}^{i-1} \geq v/2^{i-2}$. We know from (1) that:

$$active_{beg}^{i} = active_{end}^{i-1}$$

Hence, we then have by assumption:

$$active_{end}^{i-1} \geq v/2^{i-1} \tag{4.2}$$

Suppose in contradiction $v/2^{i-2} > active_{beg}^{i-1}$. Then by (2) we have the following:

$$v/2^{i-2} > active_{beg}^{i-1} \geq v/2^{i-1}$$

In this case, by Lemma 3 more than half the nodes active at the beginning of iteration $i-1$ transmit alone during iteration $i-1$. We also know that there are two or more active nodes present in every iteration, hence, $active_{beg}^{i-1} \geq active_{end}^{i-1} \geq 2$. Hence, due to Lemma 3, two or more nodes transmit alone and, due to Lemma 5 all these nodes become passive by the end of iteration $i-1$. Due to Lemma 3 the number of active nodes at the end of iteration $i-1$ is less than half of the number of active node at the beginning of iteration $i-1$. We then have:

$$active_{beg}^{i-1} \geq 2(active_{end}^{i-1}) \tag{4.3}$$

From (1) and (3):

$$active_{beg}^{i-1} = active_{end}^{i-2} \geq 2(active_{end}^{i-1}) \tag{4.4}$$

From (4) and (2):

$$2(active_{end}^{i-1}) \geq 2v/2^{i-1} \geq v/2^{i-2} \tag{4.5}$$

Hence, from (5) and (3):

$$active_{beg}^{i-1} \geq v/2^{i-2}$$

(End of proof of Claim 1.) □

We now show that at the end of the entire initialization phase nodes know about all nodes in their own and adjacent hexagons. This is because by the end of the initialization phase nodes in all hexagons get their turn to run Algorithm 2 during the assigned round. During other rounds nodes collect the ids of nodes from adjacent hexagons. Due to constraint (C1), which states that $\rho + 2mu\sigma \leq R$, nodes receive transmissions from nodes in adjacent hexagons. Hence, nodes learn the ids of all nodes in adjacent hexagons.

Theorem 2. *At the end of the initialization phase, for all nodes p, id_list_p contains the ids of all nodes present in p's adjacent hexagons.*

Proof. At the end of the initialization phase nodes in all hexagons have completed the initialization algorithm. Only nodes in hexagons of the same color run the algorithm simultaneously. Hence, there are no message collisions between nodes from different hexagons. Since $\rho + 2mu\sigma \leq R$ due to (C1), where ρ is the distance between the farthest points on the boundary of adjacent hexagons, nodes receive the transmissions of nodes in adjacent hexagons. Due to this, and assumption (A3), for all nodes p, id_list_p contains the ids of all nodes present in p's own and adjacent hexagons. $\square$

Note that a node p may receive a message from a node q which is within distance R, but not in p's own hexagon or in an adjacent hexagon, during the initialization phase. As a result p includes q's id in id_list_p. However, since p also knows q's trajectory information, it can determine whether q is its *neighbor* or not, at the start of the next phase.

Theorem 3. *The initialization algorithm requires $O(v \log n)$ broadcast slots.*

Proof. In the first iteration of the initialization algorithm, a v-selector with length $O(v \log n)$ is run. In each subsequent iteration, a smaller and smaller selector is used to schedule node transmissions. Specifically, in each iteration i, a $v/2^i$-selector of length $O(v/2^i \log n)$ is used. The length of the selector decreases by exactly half in each iteration. Hence, the total number of broadcast slots used for the $\log_2 v + 1$ iterations is $O(v \log n)$. □

4.7.3 Nodes Starting Up at Different Times

In the above discussion we assumed that all nodes start up at the same time and remain within their original hexagon during the initialization phase. One way in which to accommodate nodes starting up at different times is to run the initialization algorithm repeatedly. In order not to make the phases too long, the initialization algorithm may only be run as part of the lth phase (and each round in the lth phase has an extra $O(v \log n)$ broadcast slots). The value of l can be adjusted according to the number of new nodes which wake up as time passes. Thus l can be made larger if fewer nodes are expected to join in. This means that the initialization algorithm is run after a long period of time. However, new nodes have to remain in the same hexagon for at least the time that they participate in the initialization algorithm. Furthermore, nodes cannot join in if they start up while the initialization algorithm is running. In this case nodes must wait for the current instance of the algorithm to finish and participate in the next instance.

4.7.4 Other collision detection models

In our model we assume that nodes do not possess collision detection capabilities. Hence, nodes cannot distinguish between message collision and silence. Furthermore, while a node is transmitting it has no way of knowing whether its message was successfully received by its neigbors or not. This is a very weak model. The initialization algorithm described above also works correctly for stronger models where it is assumed that nodes possess collision detection capabilities (that is, nodes can distinguish between silence and message collision).

However, our algorithm cannot tolerate spurious messages (that is, messages may be received despite message collisions, but the content of the original message from the sender may be altered due to a collision).

The initialization algorithm also cannot handle the case where due to a message collision some nodes within broadcast range of the sender do not receive the message, while other nodes are able to receive the message successfully. Also, if there is uncertainty associated with the communication link, meaning even if a node transmits alone it is not guaranteed that its neighboring nodes will receive its message, then the algorithm will not work.

5. A DETERMINISTIC ALGORITHM FOR TRIGGER COUNTING

5.1 Introduction

In the distributed trigger counting problem, introduced by [10], it is assumed that there are n processors in an asynchronous system with point-to-point links, and triggers may be received by these processors due to external events. An alert is to be raised for the user when the total number of triggers reaches a certain value w which is specified by the user. Each processor may receive a different number of triggers. The number of triggers received by each processor and the order in which different processors receive triggers is not known in advance. We present a deterministic algorithm for this problem which is communication efficient in terms of the $MaxRcvLoad$, which is defined as the maximum number of messages received by any processor in the system. For applications in sensor networks, a low $MaxRcvLoad$ may lead to energy conservation and low levels of congestion in the network. The $MaxRcvLoad$ of our algorithm is $O(\log n \log w + \sqrt{n \log n \log w})$, compared to the best previous deterministic algorithms in the same model [21], which have a worst case $MaxRcvLoad$ of $O(n \log w)$.

System Model. In this work we assume an asynchronous computation model. We consider the situation where the network topology is a clique. This is a first step toward more general multi-hop network topologies. Message delivery is guaranteed and there are no spurious messages. Furthermore, processors and links do not fail. The links are assumed to be FIFO.

Table 5.1 gives a summary of the previous results related to the trigger counting problem as discussed in Section 2.

Table 5.1: Summary of related work.

Algorithm	Type	$MaxRcvLoad$
$Centralized$ [21]	Deterministic	$O(n \log w)$
$LayeredRand$ [10]	Randomized	$O(\log w \log w)$
$CoinRand$ [9]	Randomized	$O(\log n + \log w)$
$DeterministicTriggerCounting$	Deterministic	$O(\log n \log w + \sqrt{n \log n \log w})$

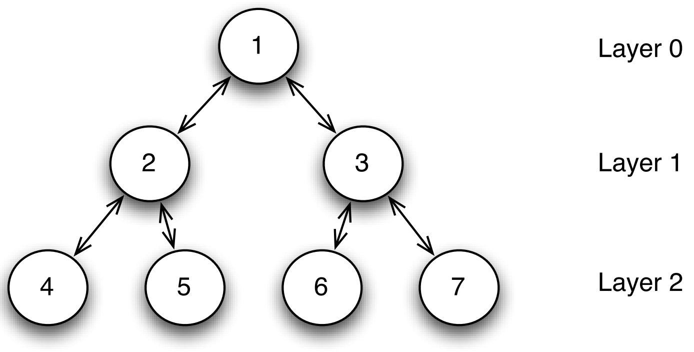


Fig. 5.1.: Nodes are divided into layers. A binary tree structure is used to transmit end-of-round messages. Each layer corresponds to one level of the tree. The nodes are numbered accordingly.

5.2 The *DeterministicTriggerCounting* Algorithm

The pseudo-code for our algorithm, called *DeterministicTriggerCounting*, is given in Algorithm 3. It is expressed in the timed I/O Automata (TIOA) modelling formalism [30]. *DeterministicTriggerCounting* is motivated by the earlier *LayeredRand* algorithm given in [10]. In that algorithm it is assumed that there is a designated root processor which collects the triggers. It is assumed that $n = 2^L - 1$, for some integer L. The processors are divided into L layers. The root is the only processor in layer 0. There are 2^ℓ processors in layer ℓ. Layer $L - 1$ is called the leaf layer.

In *LayeredRand* each processor maintains a count of the triggers which are still to be received in a local variable $\hat{w}$. For each layer ℓ a threshold value τ_ℓ is defined as follows:

$$\tau_\ell = \lceil \hat{w}/(4 \cdot 2^\ell \cdot \log_2(n+1)) \rceil$$

Each processor x at level ℓ maintains a counter C_x to keep track of the triggers received so far by itself and processors in layers below. Each time a trigger is received by non-root processor x, C_x is incremented (line 12 of Algorithm 3). .

In *LayeredRand* if $C_x \geq \tau_\ell$ then processor x chooses a processor z uniformly at random from layer $\ell - 1$, and a message called a *coin* is sent by processor x to z. C_x is updated: $C_x = C_x - \tau_\ell$. If a coin is received by processor x from a processor in a lower layer then C_x is updated as follows: $C_x = C_x + \tau_{\ell+1}$.

The *LayeredRand* algorithm works in asynchronous rounds. The local variable $\hat{w}$ is updated at the beginning of each round by the root node to indicate how many triggers remain to be collected. The root node initiates an end-of-round procedure at the end of each round. The end-of-round messages are transmitted from the root to the leaf layer along a binary tree structure overlayed on the clique (see Figure 5.1). The nodes in each layer form each successive level of the binary tree. The binary tree structure helps to keep $MaxRcvLoad$ low for the root. The value of τ_ℓ is higher for layers closer to the root. This means that coins are sent to the root less frequently as compared to lower layers.

The root initiates an end-of-round procedure when $C_{root} \geq \lceil \hat{w}/2 \rceil$. It sends *round_reset* messages to its children. These *round_reset* messages propagate down the links of the binary tree. In response, the non-root processors send a count of how many triggers have been received so far to the root. Through the end-of-round procedure the root gets the count of w', the total number of triggers received by all processors in the current round. It then sets $\hat{w} = \hat{w} - w'$ and broadcasts $\hat{w}$ to all processors down the tree. If $\hat{w} = 0$ then the root raises an alert for the user. The end-of-round procedure is described in detail at the end of this section.

The only difference between our algorithm and the *LayeredRand* algorithm is that instead of picking a processor from level $\ell - 1$ uniformly at random, in *DeterministicTriggerCounting* there is a fixed iterative order for picking layer $\ell - 1$ processors, in order to send a coin. This fixed iterative order is a permutation of layer $\ell - 1$ processor ids. Each layer ℓ processor has its own permutation of layer $\ell - 1$ processor ids.

In order to define this permutation we assume that the n processors in the network are numbered from 1 to n according to the layer which they belong to (see Figure 5.1). Then π_i is defined as the permutation of layer $\ell - 1$ processor numbers at processor i in layer ℓ. Define $\oplus$ as addition modulo $2^{\ell-1}$. Then π_i is given by:

$$(\lfloor i/2 \rfloor \oplus 0) + 2^{\ell-1}, (\lfloor i/2 \rfloor \oplus 1) + 2^{\ell-1}, (\lfloor i/2 \rfloor \oplus 2) + 2^{\ell-1}, ..., (\lfloor i/2 \rfloor \oplus 2^{\ell-1}) + 2^{\ell-1}$$

The permutations are defined such that two processors from layer ℓ share the same permutation.

Processor i from level ℓ selects all the $2^{\ell-1}$ processors in level $\ell - 1$ one by one, according to its permutation of level $\ell - 1$ processors. The variable *next* in Algorithm 3 is updated to the next id in this permutation, with wrap-around (line 23 of Algorithm 3). Each time a coin is generated at processor i, it is sent to the next processor in the permutation. See Figure 5.2 for an illustration of the permutation.

Algorithm 3 Code for processor i

```
**automaton** DeterministicTriggerCounting()
  **states**
    $round\_num : \mathbb{N} := 1, C : \mathbb{N} := 0, D : \mathbb{N} := 0$ {counters for round number and triggers}
    $\ell : \mathbb{N}$ {layer of node $i$}
    $i : \mathbb{N}$ {sequence number of processor in its layer}
    $next : \mathbb{N} := \lfloor i/2 \rfloor \bmod 2^{\ell-1} + 2^{\ell-1}$ {processor to which the next coin will be sent}
    $\hat{w} : \mathbb{N} := w$ {number of triggers left}
    $\tau : \mathbb{N} := \lceil \hat{w}/(4 \cdot 2^{\ell} \cdot \log_2(n+1)) \rceil$ {number of triggers to be collected in a coin}
    $trigger\_queue : Seq[T] := \oslash$ {used to store triggers received}
    $coin\_queue : Seq[M] := \oslash$ {used to store coins received}
    $eor\_message\_queue : Seq[M] := \oslash$ {used to store end-of-round messages received}
    $send\_queue : Seq[M] := \oslash$ {used to store messages which are to be sent}
    $suspend : Bool := false$ {indicates whether triggers and coins should be processed}
    $root : Bool$ {true if $x$ is the root}
    $leaf : Bool$ {true if $x$ is a leaf}
    $reduce\_count : \mathbb{N}$:=0 {count of reduce messages received in current round}
    $d$:=0 {local variable for storing triggers sent by child processors }

  **transitions**

  **output** $send(m, j)$ {send message $m$ to process $j$}
    **pre** $m = head(send\_queue)$;
    **eff** $send\_queue := tail(send\_queue)$;

  **input** $receive\_trigger(t)$ {a trigger is received due to an external event}
    **eff** $trigger\_queue := trigger\_queue \vdash t$;

  **input** $receive\_coin(c)$ {receive a coin from a lower layer}
    **eff** $coin\_queue := coin\_queue \vdash c$;

  **input** $receive\_eor\_message(m, j)$ {receive end-of-round message $m$ from process $j$}
    **eff** $eor\_message\_queue := eor\_message\_queue \vdash m$;

  **internal** $process\_trigger(t)$ {remove trigger from the trigger queue and add to the $C$ and $D$ counter}
    **pre** $suspend = false \wedge t = head(trigger\_queue)$;
    **eff** $C := C + 1$;
        $D := D + 1$; {$D$ is never reset}
        $trigger\_queue := tail(trigger\_queue)$;

  **internal** $process\_coin(c)$ {add the count of triggers in the received coin to $C$}
    **pre** $suspend = false \wedge c = head(coin\_queue)$;
    **eff** **if** $c.round\_num = round\_num$ **then** $C := C + \tau/2$;
        $coin\_queue := tail(coin\_queue)$;
        **endif**;

  **internal** $send\_coin()$ {a coin should be sent to the upper layer if $C$ exceeds $\tau$}
    **pre** $suspend = false \wedge \neg root \wedge C \geq \tau$;
    **eff** $send\_queue := send\_queue \vdash [[coin, i, round\_num], next]$;
        $next := (next + 1) \bmod 2^{\ell-1} + 2^{\ell-1}$; {next is reset according to $\pi_i$}
        $C := C - \tau$; {subtract the number of triggers in one coin from $C$}

  **internal** $end\_round()$ {root starts end-of-round procedure}
    **pre** $suspend = false \wedge root \wedge C \geq \hat{w}/2$; {the threshold for $C$ has been exceeded at the root}
    **eff** $send\_queue := send\_queue \vdash [[round\_reset], 2i]$;
        $send\_queue := send\_queue \vdash [[round\_reset], 2i + 1]$;
        $suspend := true$; {trigger and coin processing is suspended during the end-of-round procedure}
```

```
**internal** $process\_message(m)$ {process end-of-round messages}
  **pre** $m = head(eor\_message\_queue)$;
  **eff**
      $eor\_message\_queue := tail(eor\_message\_queue)$;
      **if** $m.type = round\_reset$ **then**
        $suspend = true$; {suspend trigger and coin processing if $round\_reset$ is received from parent}
        $send\_queue := send\_queue \vdash [[round\_reset], 2i]$;
        $send\_queue := send\_queue \vdash [[round\_reset], 2i + 1]$;
        **if** $leaf$ **then** $send\_queue := send\_queue \vdash [[reduce, round\_num, D], \lfloor i/2rfloor]$;
        **endif**; {when $round\_reset$ reaches each leaf, $D$ counter value is sent to the parent processor}
      **endif**;
      **if** $m.type = reduce \wedge m.round\_num = round\_num$ **then**
        $d := d + m.D$; {$D$ from both children added in $d$}
        $reduce\_count := reduce\_count + 1$; {count the number of reduce messages}
        **if** $reduce\_count = 2 \wedge \neg root$ **then** {received reduce message from both children}
          $reduce\_count := 0$;
          $send\_queue := send\_queue \vdash [[reduce, round\_num, D + d], \lfloor i/2rfloor]$;
        **endif**; {$reduce$ message contains sum of $D$ values of all descendants}
        **if** $reduce\_count = 2 \wedge\ root$ **then** $\hat{w} := \hat{w} - D - d$; {root receives $reduce$ }
          $reduce\_count := 0$;
          **if** $\hat{w} = 0$ **then**{if all the triggers have been received then terminate}
            $raise\_alert$;
            $terminate$;
          **else**
            $round\_num := round\_num + 1$; {root increments the round number}
            $C := 0$;
            $send\_queue := send\_queue \vdash [[inform, round\_num, \hat{w}], 2i]$;
            $send\_queue := send\_queue \vdash [[inform, round\_num, \hat{w}], 2i + 1]$;
          **endif**;
          $d := 0$;
        **endif**;
      **endif**;
      **if** $m.type = inform$ **then**{if $inform$ is received from the parent}
        $\hat{w} := m.\hat{w}$; {initialize $\hat{w}$ for the next round}
        $\tau := \lceil \hat{w}/(4 \cdot 2^{\ell} \cdot \log_2(n + 1)) \rceil$;
        $\tau/2 := \lceil \hat{w}/(4 \cdot 2^{\ell+1} \cdot \log_2(n + 1)) \rceil$;
        $C := 0$;
        $round\_num := m.round\_num$; {reset the round number}
        $send\_queue := send\_queue \vdash [[inform, round\_num, \hat{w}], 2i]$;
        $send\_queue := send\_queue \vdash [[inform, round\_num, \hat{w}], 2i + 1]$;
        $suspend := false$;
      **endif**;
```

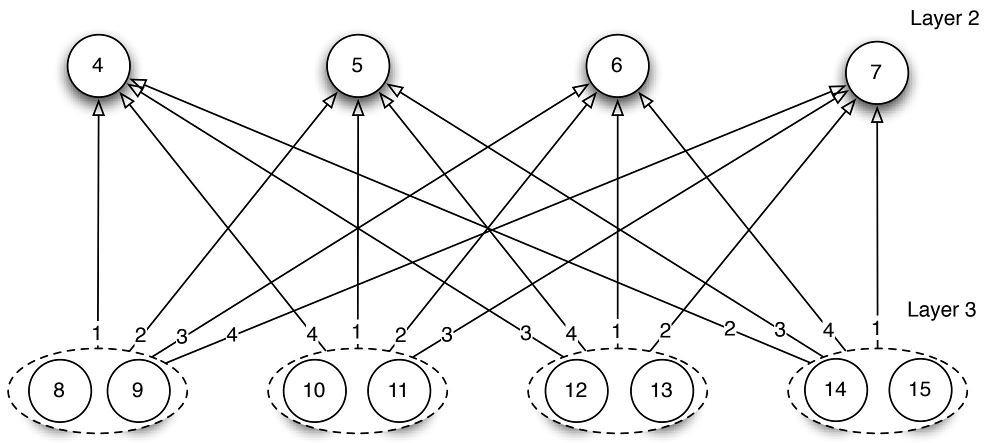


Fig. 5.2.: The ordering for sending coins from layer 2 to layer 3.

5.2.1 End-of-Round Procedure

We now describe the end-of-round procedure in detail. The root sends *round_reset* messages to its children when $C_{root} \geq \lceil \hat{w}/2 \rceil$ (line 25 of Algorithm 3). When a child processor receives the *round_reset* message from its parent processor it suspends trigger and coin processing and sends the *round_reset* message to both its children in the binary tree (line 34 of Algorithm 3). Triggers and coins are buffered in the *trigger_queue* and the *coin_queue* respectively, to be processed later when the end-of-round procedure is over.

When a leaf processor receives the *round_reset* message it sends a *reduce* message to its parent in the binary tree along with its *round_num* and its D counter (line 38 of Algorithm 3). The D counter ensures that an up-to-date value of the triggers received at each processor is sent to the root during the end-of-round procedure.

Upon receiving the *reduce* message from both its child processors in the binary tree, a non-root processor sends the sum of its own D counter value and the D

counter values of its children, to its parent processor in the binary tree (line 44 of Algorithm 3).

Upon receiving the *reduce* message from both its child processors in the binary tree, the root processor subtracts from $\hat{w}$ its own D counter value and the values received from its children (line 48 of Algorithm 3). If $\hat{w}$ is equal to zero the threshold has been reached and the root raises an alert. Otherwise the root processor sends an $inform$ message along with the incremented *round_num* value, and the new value of $\hat{w}$ to both its child processors (line 53 of Algorithm 3).

When a child processor receives the $inform$ message from its parent processor it resets its C counter, and updates its $\hat{w}$, *round_num*, and τ values (line 62 of Algorithm 3). After this it resumes trigger and coin processing (line 70 of Algorithm 3).

We assume there is an underlying point-to-point automaton with interface actions $send(m, j)$ for sending a message m to processor j, $receive\_coin(c)$ for receiving a coin from another processor, and $receive\_eor\_message(m, j)$ for receiving an end-of-round message from processor j. There is also a separate automaton which interacts with the external environment and has interface action $receive\_trigger(t)$ which represents a trigger being received from the environment.

5.3 Analysis of $DeterministicTriggerCounting$

The analysis for the correctness of the algorithm and the worst case bound on message complexity remain almost the same as those given for *LayeredRand* in [10]. This is because in our algorithm only the order of processors to which coins are sent may be different from *LayeredRand*. For completeness, in this section we give a more detailed proof of correctness for Algorithm 3 compared to the proof of correctness provided in [10].

First of all we show that whenever the root enters a new round it eventually starts the end-of-round procedure.

Lemma 5. *If an alert is never raised then round_num i for the root grows without bound.*

Proof. We will show that for every value of *round_num* i, if the root does not raise the alert (line 50 to 52 of Algorithm 3) then the root eventually executes *end_round*(). The proof is by induction on the i.

(*Basis:* $i = 1$) In this case $\hat{w} = w$.

Suppose that at time t the system has received all remaining $\hat{w}$ triggers. We now show that if the root has not already executed *end_round*() at or before time t it eventually does so after time t. This is because eventually after time t all coin messages in the system are delivered, and all $\hat{w}$ triggers are counted in the C counters of some processors (due to line 12 and 17 of Algorithm 3). At this point for all processors x we have:

$$\sum_{x=1}^{n} C_x = \hat{w}$$

Summing the C counters excluding the root we have:

$$\begin{aligned}\sum_{x=2}^{n} C_x &\le \sum_{\ell=1}^{L-1} 2^{\ell}(\tau_\ell - 1)\\ &= \sum_{\ell=1}^{L-1} 2^{\ell}(\lceil \hat{w}/(4 \cdot 2^{\ell} \cdot \log_2(n+1))\rceil - 1)\\ &\le \sum_{\ell=1}^{L-1} 2^{\ell}(\hat{w}/(4 \cdot 2^{\ell} \cdot \log_2(n+1)))\\ &\le (L-1)\hat{w}/(4\log_2(n+1))\\ &\le \hat{w}/4\end{aligned}$$

Hence, for the root, $C \ge \lceil \hat{w}/2 \rceil$, and the root executes *end_round*().

(*Inductive Case*)

Assuming the root executes *end_round*() for round $i-1$ we show that the root executes *end_round*() for round i.

Consider the point in time when the root sets *round_num* to $i > 1$ (line 54 of Algorithm 3). Let the initial value of $\hat{w}$ for round i be $\hat{w}_i$. The value of $\hat{w}_i$ is set to $\hat{w} - D - d$ immediately before this, say at time t (line 48 of Algorithm 3). First of all we show that at time t, *suspend* $=$ *true* for all processors in the system. This is because the root has executed *end_round*() for round $i-1$ and set its value of *suspend* to *true* (line 29 of Algorithm 3). Since we assume reliable communication, all processors have received the *round_reset* message propagated down from the root (line 34 of Algorithm 3) and also set *suspend* to true (line 35 of Algorithm 3). Furthermore, the root increments *round_num* to i only when all the processors including leaf processors have received *round_reset* messages, and *reduce* messages have been propagated from each child node to its parent in the binary tree, including the children of the root (line 48 of Algorithm 3).

We can see that at any processor, *process_trigger*() or *process_coin*() will not be executed until *suspend* is set to *true* again and *round_num* is set to i. This happens when $inform$ messages travel down the binary tree from the root (line 62 of Algorithm 3). Also the total number of triggers counted by the root at time t is $D + d$. For each processor, its contribution to d is the number of triggers it received before it set *suspend* to *true* (line 38 and 46 of Algorithm 3). Therefore, all remaining $\hat{w}_i$ triggers, since they have either not been processed yet or not been received yet, will be counted in round i or later rounds.

Suppose that at time $t' > t$ the system has received all remaining $\hat{w}_i$ triggers. We now show that if the root has not already executed *end_round*() for round i at or before time t' it eventually does so after time t'. This is because eventually after time t' all triggers from the *trigger_queue* are processed, since *suspend* is set to *true* when the $inform$ message is received. All coin messages in the system are delivered, and all $\hat{w}_i$ triggers are counted in the C counters of some processor. At this point for all processors x we have:

$$\sum_{x=1}^{n} C_x = \hat{w}_i$$

Summing the C counters excluding the root we have:

$$\begin{aligned}\sum_{x=2}^{n} C_x &\leq \sum_{\ell=1}^{L-1} 2^{\ell}(\tau_{\ell} - 1) \\ &= \sum_{\ell=1}^{L-1} 2^{\ell}(\lceil \hat{w}/(4 \cdot 2^{\ell} \cdot \log_2(n+1)) \rceil - 1) \\ &\leq \sum_{\ell=1}^{L-1} 2^{\ell}(\hat{w}/(4 \cdot 2^{\ell} \cdot \log_2(n+1))) \\ &\leq (L-1)\hat{w}/(4 \log_2(n+1)) \\ &\leq \hat{w}/4\end{aligned}$$

Hence, for the root, $C \geq \lceil \hat{w}_i/2 \rceil$, and the root eventually executes *end_round*().

□

Lemma 6. *The root does not raise an alert before w triggers have occurred in the system.*

Proof. For each round $\hat{w}$ is initialized at the root by subtracting the sum of the D counter of all processors, propagated to the root through *reduce* messages. D counters are never reset and count only the triggers received by a particular processor itself. Furthermore, *suspend* is set to *true* before the D count is sent to the parent of a processor. A node only sets *suspend* to $false$ after incrementing *round_num* (in line 70 of Algorithm 3). Therefore, each trigger is only counted once by the root. □

Theorem 11. *Algorithm 3 detects when w triggers have been received by the system.*

Proof. If during the end-of-round procedure the root detects that $\hat{w} = 0$, it raises an alert and then terminates (in line 50 of Algorithm 3). This together with Lemma 5 and 6 shows that Algorithm 3 correctly detects when w triggers have been received by the system. □

5.3.1 Bound on the Total Message Complexity and $MaxRcvLoad$

Here we analyze the total message complexity of Algorithm 3 and show that the $MaxRcvLoad$ is bounded by $O(\log n \log w + \sqrt{n \log n \log w})$. Fix an arbitrary execution of Algorithm 3.

Lemma 7. *The total number of rounds in the execution is at most* $\lceil \log_2 w \rceil$.

Proof. As soon as the root node detects $C \geq \hat{w}/2$, it initiates an end-of-round procedure. Thus $\hat{w}$ drops by a factor of two in each successive asynchronous round. Algorithm 3 terminates when the root detects that $\hat{w} = 0$. □

Lemma 8. *Suppose that for two processors i and j, $round\_num_i = round\_num_j$, then we have $\hat{w}_i = \hat{w}_j$.*

Proof. This is because $\hat{w}$ and *round_num* are both updated upon receiving an inform message, and the root sends only one inform message for one value of *round_num*. □

In the following lemma we give an upper bound for the total number of coins received by the layer above the leaf layer (i.e., layer $L - 2$). The leaf layer has the minimum τ value. This means that for a particular *round_num* leaf processors send coins upon receiving the least number of triggers, as compared to processors in other layers. If all triggers are received by leaf processors rather than processors in other layers, then the most number of coins messages will be sent. Hence, the upper bound for total number of coins received by layer $L - 2$ is also the upper bound for the rest of the layers.

Lemma 9. *The maximum number of coins received by all processors in layer $L - 2$ during the execution of Algorithm 3 is* $\lceil 4(2^{L-1} \log_2 n \log_2 w) \rceil$.

Proof. Consider round r (as indicated by *round_num* in Algorithm 3), and let $\hat{w} = w_i$ for a processor i from layer $L-2$ in round r. Then for all processors in layer $L-2$ and $L-1$ for which $round\_num = r$, we have $\hat{w} = w_i$ due to Lemma 8. Note that for $round\_num = r$ any processor will only process coins which have round number r (line 17 of Algorithm 3).

Suppose that $\hat{w}$ is the total number of triggers which are still to be collected when the root transitions to round number r. In the worst case all these triggers may occur before any processor in layer $L-2$ or layer $L-1$ transitions to the next round (i.e., $r+1$), and the coins associated with these triggers are received by layer $L-2$ processors while $round\_num = r$ for all layer $L-2$ processors. This is the worst case since the maximum number of coins will be received by layer $L-2$ processors for $round\_num = r$ in this case. In this case for round r the maximum number of coins received by all processors in layer $L-2$ is $\hat{w}/(\tau_{L-1}) \leq 4(2^{L-1}\log_2 n)$ (substituting the value of τ_{L-1}). Thus in total the maximum number of coins received by layer $L-2$ is $\lceil 4(2^{L-1}\log_2 n \log_2 w)\rceil$ due to Lemma 7. □

In the above analysis end-of-round messages may be ignored since a constant number of end-of-round messages (*round_reset*, *reduce*, and *inform* messages) are sent in each round along the links of the binary tree structure. The total message complexity of Algorithm 3 is determined by summing up the total number of coins sent over all layers and is given by Theorem 12. The analysis is the same as that given in Lemma 1 in [10].

Theorem 12. *The message complexity of* DeterministicTriggerCounting *is given by* $O(n \log n \log w)$.

Proof Sketch. Here we give the main idea for the proof of the above theorem. The number of coins sent in each round is:

$$\sum_{\ell=1}^{L-1} \hat{w}/\tau_\ell \leq \sum_{\ell=1}^{L-1} 4 \cdot 2^\ell \cdot \log_2 n \leq 4 \cdot (n-1) \cdot \log_2 n$$

The total number of coins sent in all rounds is then $O(n \log n \log w)$. $\square$

We analyze the *MaxRcvLoad* by determining the number of coins received by any node. It is obvious that leaf nodes do not receive any coins. In order to determine the *MaxRcvLoad* let us define the following terms.

Definition 18. *Define X as the total number of coins sent from layer $\ell+1$ to layer ℓ. We know from Lemma 9 that $X = \lceil 4(2^{\ell+1} \log_2 n \log_2 w) \rceil$.*

Definition 19. *Suppose that processor i in layer $\ell+1$ sends s_i coins out of the total X coins. Let $s_i = a_i \cdot 2^\ell + b_i$ where $a_i \geq 0$ and $0 \leq b_i < 2^\ell$ (note that b_i is the remainder after dividing s_i by 2^ℓ).*

We know that these s_i coins are sent iteratively to layer ℓ processors according to π_i, as indicated by the *next* variable in Algorithm 3. Note that *next* is not reset at the beginning of the asynchronous rounds.

Also note that $X = \sum_{i=2^{\ell+1}}^{2^{\ell+2}-1} s_i$. Each processor in layer ℓ receives either $\sum_{i=2^{\ell+1}}^{2^{\ell+2}-1} a_i$ or $\sum_{i=2^{\ell+1}}^{2^{\ell+2}-1} a_i + 1$ coins from processor i.

Definition 20. *Let the last b_i coins sent by processor i in layer $\ell+1$ be called the remainder coins of i.*

Lemma 10. *$\sum_{i=2^{\ell+1}}^{2^{\ell+2}-1} a_i = O(\log n \log w)$.*

Proof.

$$\begin{aligned}
&\sum_{i=2^{\ell+1}}^{2^{\ell+2}-1} a_i \\
=&\sum_{i=2^{\ell+1}}^{2^{\ell+2}-1} a_i \cdot 2^\ell/2^\ell \\
\leq&(1/2^\ell)\sum_{i=2^{\ell+1}}^{2^{\ell+2}-1} (a_i \cdot 2^\ell + b_i) \\
=&(1/2^\ell)\sum_{i=2^{\ell+1}}^{2^{\ell+2}-1} s_i \\
=&X/2^\ell \\
=&\ O(\log n \log w)
\end{aligned}$$

□

Before we proceed let us define the following terms.

Definition 21. *For two processors with numbers p and q, where p is from layer ℓ and q is from layer $\ell+1$, define $dist(p,q)$ as follows:*

$$\begin{aligned}
p - \lfloor q/2 \rfloor + 1 &\quad if\ p \geq \lfloor q/2 \rfloor \\
p - (2^\ell - (\lfloor q/2 \rfloor \bmod 2^\ell)) + 1 &\quad if\ p < \lfloor q/2 \rfloor
\end{aligned}$$

Note that $dist(p,q)$ is the minimum number of coins q must send in order to send a coin to p (in accordance with π_q).

Definition 22. *For a processor j from layer ℓ and a processor i from layer $\ell+1$ let $c_i^j = 1$ if $b_i \geq dist(j,i)$, and $c_i^j = 0$ otherwise.*

Definition 23. *Let $\delta_i = \lfloor i/2 \rfloor$ be the first element of π_i.*

In the following lemma we show that if we consider only the remainder coins of layer $\ell+1$ processors and c of these are received by one layer ℓ processor, then the sum of all the remainder coins of layer $\ell+1$ processors is $\Omega(c^2)$.

Lemma 11. *For all $c > 0$ if there exists a processor j for which $\sum_{i=2^{\ell+1}}^{2^{\ell+2}-1} c_i^j = c$ then $\sum_{i=2^{\ell+1}}^{2^{\ell+2}-1} b_i = \Omega(c^2)$*

Proof. Each layer $\ell+1$ processor i starts sending coins beginning with processor $\delta_i{=}\lfloor i/2 \rfloor$. It then sends coins to layer ℓ processors with increasing processor numbers and wraps around when the layer ℓ processor with the greatest number is reached. Since j receives c coins and only one coin can be received by j from each layer $\ell+1$ processor i due its last b_i coins (remainder coins), there exist c layer $\ell+1$ processors for which $b_i \geq dist(j,i)$. Order each of these c layer $\ell+1$ processors in increasing order of $dist$ with respect to j. Suppose that this ordering is given by $p_1, p_2, ..., p_c$. Then we have $dist(j,p_1) \geq 1$, and $dist(j,p_2) \geq 1$. Since only two processors in layer $\ell+1$ have the same value of offset, we have $dist(j,p_3) \geq 2$, and $dist(j,p_4) \geq 2$. In general we have $dist(j,p_i) \geq \lceil i/2 \rceil$. Since $dist(j,p_i)$ represents the least number of coins sent by p_i, we have:

$$2\sum_{i=1}^{c/2} i \leq \sum_{i=2^{\ell+1}}^{2^{\ell+2}-1} b_i$$

From this we have that:

$$2\sum_{i=1}^{c/2} i = c^2/4 + c/2 \leq \sum_{i=2^{\ell+1}}^{2^{\ell+2}-1} b_i$$

Hence, $\sum_{i=2^{\ell+1}}^{2^{\ell+2}-1} b_i$ is $\Omega(c^2)$. □

Lemma 12. *For all processors j in layer ℓ $\sum_{i=2^{\ell+1}}^{2^{\ell+2}-1} c_i^j = O(\sqrt{n \log n \log w})$*

Proof. Suppose c is the number of coins received by processor j. From Lemma 11 we have

$$\sum_{i=2^{\ell+1}}^{2^{\ell+2}-1} b_i = \Omega(c^2)$$

Hence:

$$c = O(\sqrt{\sum_{i=2^{\ell+1}}^{2^{\ell+2}-1} b_i})$$

We also have:

$$c = \sum_{i=2^{\ell+1}}^{2^{\ell+2}-1} c_i^j$$

Due to Lemma 9 we have:

$$\sum_{i=2^{\ell+1}}^{2^{\ell+2}-1} b_i \leq 4(n/2)\log_2 n \log_2 w$$

Therefore, we have:

$$c = \sum_{i=2^{\ell+1}}^{2^{\ell+2}-1} c_i^j = O(\sqrt{n \log n \log w})$$

□

In Lemma 10 we considered all coins sent by layer $\ell + 1$ processors except remainder coins, whereas, in Lemma 12 we considered only remainder coins. The *MaxRcvLoad* is given by the sum of the contribution from remainder coins and non-remainder coins. Hence, we have the following:

Theorem 13. *The MaxRcvLoad for Algorithm 3 is* $O(\log n \log w + \sqrt{n \log n \log w})$.

Proof. From Lemma 10 and 12 we have that the *MaxRcvLoad* for *DeterministicTriggerCounting* is $O(\log n \log w + \sqrt{n \log n \log w})$. A constant number of end-of-round messages are sent to each processor during the end-of-round procedure at the end of each round. Since there are $O(\log n)$ rounds, these messages do not affect the *MaxRcvLoad.* □

5.4 Lower Bound

In this section we show that for any deterministic algorithm which solves the trigger counting problem in a tree where the root has constant degree, at least one node must receive $\Omega(\log w)$ messages. The lower bound is for the case of all algorithms in which there is a designated root node, fixed in advance, which raises the alert when the threshold is reached.

First of all we give a lower bound of $\Omega(\log w)$ for the *MaxRcvLoad* when n is fixed and w grows without bound with respect to n. This follows from the lower bound of $\Omega(n \log(w/n))$ given in [21] for the total number of messages. This lower bound is stated as follows:

Theorem 14. *The total number of messages exchanged by any algorithm which solves the trigger counting problem is (as shown in [21])*

$$\begin{aligned} \Omega(n \log(w/n)) \; if \quad & w \geq n \\ \Omega(w) \; if \quad & w < n \end{aligned}$$

Lemma 13. *For any deterministic algorithm which solves the trigger counting problem where n is fixed and w grows without bound with respect to n, the $MaxRcvLoad$ is $\Omega(\log w)$.*

Proof. The total number of messages sent by any deterministic algorithm for the trigger counting problem is $\Omega(n \log(w/))$ by Theorem 14. By the pigeonhole principle, at least one of the n nodes receives $\Omega(\log(w/n))$ messages. Hence, the $MaxRcvLoad$ is $\Omega(\log(w/n)) = \Omega(\log(w))$, since n is fixed and w grows without bound with respect to n. □

We now show a lower bound for the case of a tree in which the designated root node has constant degree.

Theorem 15. *For any deterministic algorithm which solves the trigger counting problem for a tree where the designated root node has constant degree c, the $MaxRcvLoad$ is* $\Omega(\log w)$.

Proof. Suppose that there exists an algorithm A which solves the trigger counting problem for such a tree and has $MaxRcvLoad$ $o(\log w)$. We can then use algorithm A to solve the trigger counting problem for $c + 1$ nodes which form a star network, where c is the degree of the root node in the tree, using an algorithm A', which is as follows:

- In algorithm A' the node at the center of the star network simulates the behaviour of the root node in the tree.
- Other nodes in the star network simulate the behaviour of the c neighbors of the root, including message passing on the links with their descendants.
- Each time a node other than the root node in the tree sends a message to the root node of the tree in algorithm A, the node simulating it in the star network sends a message to the center node of the star in algorithm A'.
- The center node raises the alert for the user in algorithm A' when the root node in the simulated tree raises the alert in algorithm A.

Algorithm A terminates correctly when the threshold is reached and the root node in the tree raises an alert for the user. At this time no node has received greater than $o(\log w)$ messages in total on all of its incoming links. Hence, algorithm A' terminates correctly and the center node raises the alert when the threshold is reached. At this time no node in the start network has received greater than $o(\log w)$ messages in total on all of its incoming links. However, we already showed in Lemma 13 that one node must receive at least $\Omega(\log w)$ messages. This is a contradiction. □

6. CONCLUSION

In this section we summarize our results and outline the future work for the three different topics addressed in this thesis.

6.1 Reliable Neighbor Discovery with an Abstract MAC Layer

We have given a specification for a reliable neighbor discovery layer, with two different progress conditions. We have presented two different protocols, Basic-NDP which satisfies weak progress and provides loose coordination between mobile nodes and UNDP which provides uniform progress at the expense of loose coordination. We have discussed different algorithms which may be used with either Basic-NDP or UNDP, depending on the requirements.

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. Another direction would be to extend this work to three dimensions and use three dimensional partitions.

6.2 Neighbor Knowledge and Collision Avoidance

We have presented a deterministic schedule for nodes in a MANET, which avoids collisions and allows nodes to maintain information about neighboring nodes. We have also given an initialization algorithm for nodes to learn about neighbors at start-up.

It remains open to analyze the rate of information propagation for our schedule, between two mobile nodes located initially at some distance on the plane. Lower bounds on the density of nodes may have to be specified, so that there are no gaps in the flow of information.

Another interesting problem would be to analyze the problem of neighbor knowledge maintenance for clusters of nodes moving on the plane. Connectivity constraints for clusters would have to be specified. Also we would have to specify what is meant by two dynamic node clusters merging on the plane.

It would be of interest to explore the fundamental limitations of deterministic solutions through lower bounds on performance or impossibility results.

6.3 Communication Efficiency for Distributed Trigger Counting in Sensor Networks

We have presented a deterministic algorithm which achieves better $MaxRcvLoad$ as as compared to previous deterministic algorithms. We also gave a lower bound of $\Omega(\log w)$ for the $MaxRcvLoad$. Future work includes tightening the gap between upper and lower bounds. It would also be interesting to investigate whether there is a gap between the lower bounds for deterministic and randomized algorithms. Another direction would be to consider node failures, and network graphs other than cliques. Algorithms for broadcast instead of point-to-point communication may also be developed.

REFERENCES

[1] Anta, A.F., Milani, A., Mosteiro, M.A., Zaks, S.: Opportunistic information dissemination in mobile ad-hoc networks: The profit of global synchrony. In: Proceedings of the International Symposium on Distributed Computing (DISC), pp. 374–388 (2010)

[2] Arumugam, M., Kulkarni, S.: Self-stabilizing deterministic tdma for sensor networks. In: G. Chakraborty (ed.) Distributed Computing and Internet Technology, *Lecture Notes in Computer Science*, vol. 3816, pp. 69–81. Springer Berlin / Heidelberg (2005).

[3] Baldoni, R., Ioannidou, K., Milani, A.: Mobility versus the cost of geocasting in mobile ad-hoc networks. In: A. Pelc (ed.) Distributed Computing, *Lecture Notes in Computer Science*, vol. 4731, pp. 48–62. Springer Berlin / Heidelberg (2007).

[4] 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, 104–126 (1992)

[5] Bharghavan, V., Demers, A., Shenker, S., Zhang, L.: Macaw: A media access protocol for wireless lans. In: Proc. ACM SIGCOMM Conference on Communications Architectures, Protocols, and Applications, pp. 212–225 (1994)

[6] Broch, J., Maltz, D., Johnson, D., Jetcheva, J.: A performance comparison of multi-hop wireless ad hoc network routing protocols. In: Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom), pp. 85–97 (1998)

[7] Bruschi, D., Del Pinto, M.: Lower bounds for the broadcast problem in mobile radio networks. Distrib. Comput. 10, 129–135 (1997).

[8] Calinescu, G.: Computing 2-hop neighborhoods in ad hoc wireless networks. In: ADHOC-NOW, pp. 175–186 (2003)

[9] Chakaravarthy, V., Choudhury, A., Sabharwal, Y.: Improved algorithms for the distributed trigger counting problem. In: Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International, pp. 515–523 (2011).

[10] Chakaravarthy, V.T., Choudhury, A.R., Garg, V.K., Sabharwal, Y.: An efficient decentralized algorithm for the distributed trigger counting problem. In: ICDCN'11, pp. 53–64 (2011)

[11] Chandra, R., Fetzer, C., Hogstedt, K.: A mesh-based robust topology discovery algorithm for hybrid wireless networks. In: Informatics, pp. 1–15 (2002)

[12] Chrobak, M., Gasieniec, L., Rytter, W.: Fast broadcasting and gossiping in radio networks. J. Algorithms 43(2), 177–189 (2002).

[13] Clementi, A.E.F., Monti, A., Silvestri, R.: Distributed broadcast in radio networks of unknown topology. Theor. Comput. Sci. 302, 337–364 (2003).

[14] Cornejo, A., Lynch, N., Viqar, S., Welch, J.L.: A neighbor discovery service using an abstract mac layer. In: Proceedings of the 47th Annual Allerton Conference, pp. 1460–1467 (2009)

[15] Cornejo, A., Viqar, S., Welch, J.L.: Reliable neighbor discovery for mobile ad hoc networks. In: Proceedings of the DIALM-POMC Joint Workshop on Foundations of Mobile Computing, pp. 63–72 (2010)

[16] Dutta, P., Culler, D.: Practical asynchronous neighbor discovery and rendezvous for mobile sensing applications. In: Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems, SenSys '08, pp. 71–84 (2008).

[17] Dyo, V., Mascolo, C.: Efficient node discovery in mobile wireless sensor networks. In: Proceedings of the 4th IEEE International Conference on Distributed Computing in Sensor Systems, DCOSS '08, pp. 478–485. Berlin, Heidelberg (2008).

[18] Ellen, F., Subramanian, S., Welch, J.L.: Maintaining information about nearby processors in a mobile environment. In: Proceedings of the International Conference on Distributed Computing and Networking, Lecture Notes in Computer Science 4308, pp. 193–202 (2006)

[19] Emek, Y., Korman, A.: Efficient threshold detection in a distributed environment: extended abstract. In: Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, PODC '10, pp. 183–191. ACM, New York, NY, USA (2010).

[20] Gandhi, R., Parthasarathy, S., Mishra, A.: Minimizing broadcast latency and redundancy in ad hoc networks. In: Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking & Computing, MobiHoc '03, pp. 222–232. ACM, New York, NY, USA (2003).

[21] Garg, R., Garg, V.K., Sabharwal, Y.: Scalable algorithms for global snapshots in distributed systems. In: Proceedings of the 20th Annual International Conference on Supercomputing, ICS '06, pp. 269–277. ACM, New York, NY, USA (2006).

[22] Gasieniec, L., Pagourtzis, A., Potapov, I., Radzik, T.: Deterministic communication in radio networks with large labels. Algorithmica 47(1), 97–117 (2007).

[23] Hull, B., Bychkovsky, V., Zhang, Y., Chen, K., Goraczko, M., Miu, A., Shih, E., Balakrishnan, H., Madden, S.: Cartel: a distributed mobile sensor computing system. In: Proceedings of the 4th International Conference on Embedded Networked Sensor Systems, SenSys '06, pp. 125–138. ACM, New York, NY, USA (2006).

[24] IEEE 802.11, 2007 IEEE standard for information technology-telecommunications and information exchange between systems-local and metropolitan area networks-specific requirements - part 11: Wireless lan mac and phy specifications.

[25] Indyk, P.: Deterministic superimposed coding with applications to pattern matching. In: Proceedings of the 38th Annual Symposium on Foundations of Computer Science, FOCS '97, pp. 127–. IEEE Computer Society, Washington, DC, USA (1997).

[26] Indyk, P.: Explicit constructions of selectors and related combinatorial structures, with applications. In: Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '02, pp. 697–704. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2002).

[27] Ingram, R., Shields, P., Walter, J.E., Welch, J.L.: An asynchronous leader election algorithm for dynamic networks. In: Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 1–12 (2009)

[28] Ioannidou, K.: Dynamic quorum systems in mobile ad-hoc networks. Ph.D. Thesis, Department of Computer Science, University of Toronto (2006)

[29] Karn, P.: MACA - a new channel access method for packet radio. In: ARRL/CRRL Amateur Radio 9th Computer Networking Conference, pp. 134–140 (1990)

[30] Kaynar, D.K., Lynch, N., Segala, R., Vaandrager, F.: The theory of timed I/O automata. Tech. Rep. MIT-LCS-TR-917a, MIT Laboratory for Computer Science, Cambridge, MA (2004)

[31] Khabbazian, M., Kuhn, F., Kowalski, D.R., Lynch, N.: Decomposing broadcast algorithms using abstract mac layers. In: Proceedings of the 6th International Workshop on Foundations of Mobile Computing, DIALM-POMC '10, pp. 13–22 (2010).

[32] Klappenecker, A., Lee, H., Welch, J.L.: Scheduling sensors by tiling lattices. Parallel Processing Letters (to appear).

[33] Komlos, J., Greenberg, A.G.: An asymptotically nonadaptive algorithm for conflict resolution in multiple-access channels. IEEE Transactions on Information Theory IT-31(2), 302–306 (1985)

[34] Kowalski, D.R.: On selection problem in radio networks. In: Proceedings of the Twenty-Fourth Annual ACM Symposium on Principles of Distributed Computing, PODC '05, pp. 158–166 (2005).

[35] 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: Proceedings of the International Symposium on Distributed Computing (DISC), pp. 572–574 (2006)

[36] Kuhn, F., Lynch, N., Newport, C.: The abstract MAC layer. In: Proceedings of the 23rd International Symposium on Distributed Computing (DISC), pp. 48–62 (2009)

[37] Liu, D.: Protecting neighbor discovery against node compromises in sensor networks. In: Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems, ICDCS '09, pp. 579–588 (2009).

[38] Malpani, N., Vaidya, N., Chen, Y., Welch, J.L.: Distributed token circulation in mobile ad hoc networks. IEEE Transactions on Mobile Computing 4(2), 154–165 (2005)

[39] Misener, J., Sengupta, R., Krishnan, H.: Cooperative collision warning: Enabling crash avoidance with wireless technology. In: Proceedings of the 12th World Congress on Intelligent Transport Systems (2005)

[40] Park, V.D., Corson, M.S.: A highly adaptive distributed routing algorithm for mobile wireless networks. In: Proceedings of the 16th IEEE Conference on Computer Communications (INFOCOM), pp. 1405–1413 (1997)

[41] Perkins, C.E., Royer, E.M.: Ad-hoc on-demand distance vector routing. In: Proceedings of the 2nd IEEE Workshop on Mobile Computer Systems and Applications, pp. 90–100 (1999)

[42] Petriu, E., Whalen, T., Abielmona, R., Stewart, A.: Robotic sensor agents: a new generation of intelligent agents for complex environment monitoring. IEEE Magazine on Instrumentation and Measurement 7(3), 46–51 (2004)

[43] Prabh, K.S., Abdelzaher, T.F.: On scheduling and real-time capacity of hexagonal wireless sensor networks. In: Proceedings of the 19th Euromicro Conference on Real-Time Systems, ECRTS '07, pp. 136–145. IEEE Computer Society, Washington, DC, USA (2007).

[44] Prakash, R., Schiper, A., Mohsin, M., Cavin, D., Sasson, Y.: A lower bound for broadcasting in mobile ad hoc networks. Tech. Rep. IC/2004/37, EPFL (2004)

[45] Rao, A., Ratnasamy, S., Papadimitriou, C., Shenker, S., Stoica, I.: Geographic routing without location information. In: Proceedings of the 9th Annual International Conference on Mobile Computing and Networking, MobiCom '03, pp. 96–108 (2003).

[46] Smith, A., Balakrishnan, H., Goraczko, M., Priyantha, N.: Tracking moving devices with the cricket location system. In: Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services, MobiSys '04, pp. 190–202 (2004).

[47] Vasudevan, S., Kurose, J.F., Towsley, D.F.: On neighbor discovery in wireless networks with directional antennas. In: INFOCOM, pp. 2502–2512 (2005)

[48] Viqar, S., Welch, J.L.: Deterministic collision free communication despite continuous motion. In: ALGOSENSORS, pp. 218–229 (2009)

[49] Wald, L.: When the roads talk, your car can listen. The New York TImes (2008)

[50] Walter, J., Welch, J.L., Vaidya, N.: A mutual exclusion algorithm for ad hoc mobile networks. Wireless Networks 7(6), 585–600 (2001)

APPENDIX A

APPENDIX: TIOA CODE

We describe the algorithms using the TIOA formalism [30].

For simplicity we define the function *hops* : $\{R_u\}_{u\in U} \times \{R_u\}_{u\in U} \to \mathbb{N}$ 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* : $\mathbb{R}^2 \to 2^{\mathcal{U}}$ 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:

$$\begin{aligned} \exists u : \mathit{Region}\ (u \neq \mathit{val}(\mathit{region}))\wedge \\ \mathit{curreg} \notin \mathit{getregion}(\mathit{traj}_{\mathit{now}}) \wedge u \in \mathit{getregion}(\mathit{traj}_{\mathit{now}})\wedge \\ \mathit{curreg} \in \mathit{getregion}(\mathit{traj}_{\mathit{now}-\varepsilon}) \wedge u \in \mathit{getregion}(\mathit{traj}_{\mathit{now}-\varepsilon}) \end{aligned}$$

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 4 Neighbor Discovery Protocol

```
automaton NDP($i$:$\mathbb{N}$, $traj$:$Traj$, $F^{+}_{ack}$:$\mathbb{R}$, $L$:$\mathbb{R}$, $k$:$\mathbb{N}$,$\delta_{LU}$:$\mathbb{R}$,$\delta_{LD}$:$\mathbb{R}$)
  states
    $active : Bool := false;$
    $sendbuffer : Seq[M] := \oslash;$
    $recvbuffer : Seq[M] := \oslash;$
    $eventqueue : Seq[Ev] := \oslash;$
    $S : Set[\mathbb{N}] := \oslash;$
    $regs : Map[\mathbb{N}, Region] := empty;$
    $curreg : Null[Region] := nil;$
    $newreg : Null[Region] := nil;$
    $jointrigger : \mathbb{R} := -1;$
    $now : \mathbb{R} := 0;$

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

     input $rcv(m, i)$     // message $m$ is received
       eff $recvbuffer := recvbuffer \vdash m;$

     internal $enter\_region(i)$
       pre $eventqueue = \oslash \wedge getregion(traj[now]) \neq val(curreg);$
       eff $curreg := embed(getregion(traj[now]));$
         if $\forall t : \mathbb{R}(t \geq now \wedge t \leq now + \delta_{LU} + L + \delta_{LD}$//if staying in new region long enough
             $\Rightarrow getregion(traj[t]) = val(curreg))$ then
           $sendbuffer := sendbuffer \vdash [[join, val(curreg), nil], i];$     //send $join$ message
           $active := true;$ // set active flag to true

    internal $leave\_region(i)$
       pre $eventqueue = \oslash \wedge active$     //if active flag is set to true
       $\wedge getregion(traj[now + \delta_{LD}]) \neq val(curreg);$ //about to cross boundary of curent region
       eff $newreg := embed(getregion(traj[now + \delta_{LD}]));$
         $active := false;$
         if $\exists t : \mathbb{R}(t \geq now + \delta_{LD} \wedge$
             $t \leq now + \delta_{LD} + \delta_{LU} + L + \delta_{LD}$
             $\Rightarrow getregion(traj[t]) \neq val(newreg))$ then
           $newreg := nil;$     //if not staying in the new region for sufficiently long
         $sendbuffer := sendbuffer \vdash$
                 $[[leave, val(curreg), newreg], i];$     //send $leave\_region$ message
         for $j : \mathbb{N}$ in $S$
           if $hops(regs[j], val(newreg)) > k$ then
             $eventqueue := eventqueue \vdash [down, j, regs[j]];$     //do a $link\_down$
```

```
internal $process\_message(m, i)$
  pre $eventqueue = \oslash \wedge m = head(recvbuffer)$
    $\wedge getregion(traj[now]) = val(curreg)$;
  eff $recvbuffer := tail(recvbuffer)$;
    if $m.sender \in S$ then
      $regs := update(regs, m.sender, m.msg.reg)$;
    if $hops(m.msg.reg, val(curreg)) \leq k$ then //message from neighbor within $k$ hops
      if $m.msg.type = join \wedge m.sender \notin S \wedge$ //$join$ received and sender not in neighbor set
            $(\forall t : \mathbb{R}(t \geq now \wedge t \leq now + \delta_{LU} + \delta_{LD}$ //if in current region for sufficiently long
            $\Rightarrow getregion(traj[t]) = val(curreg)))$ then
        if $jointrigger = -1$ then
          $jointrigger := now + F^{+}_{ack}$;                //send $join\_reply$ after $F^{+}_{ack}$
        $eventqueue := eventqueue \vdash [up, m.sender, m.msg.reg]$;    //do a $link\_up$
      if $m.msg.type = leave \wedge m.sender \in S \wedge$            //if a $leave$ message is received
            $hops(val(m.msg.dest), val(curreg)) > k$ then //if sender will not be within $k$ hops
        $eventqueue := eventqueue \vdash$
            $[down, m.sender, regs[m.sender]]$;          // do a $link\_down$
      if $m.msg.type = join\_reply \wedge m.sender \notin S$// if $join\_reply$ sender is not in neighbor set
            $\wedge active$ then    //check active flag
        $eventqueue := eventqueue \vdash [up, m.sender, m.msg.reg]$;    // do a $link\_up$

internal $send\_join\_reply(i)$    // batch processing of $join\_reply$ messages
  pre $eventqueue = \oslash \wedge jointrigger = now$;
  eff $jointrigger := -1$;
    $sendbuffer := sendbuffer \vdash$
            $[[join\_reply, val(curreg), nil], i]$;

output $link\_down(j, i)$
  pre $\exists reg : Region(head(eventqueue) = [down, j, reg])$;
  eff $S := S - j$;    // remove node $j$ from neighbor set after doing a $link\_down$
    $regs := remove(regs, j)$;
    $eventqueue := tail(eventqueue)$;

output $link\_up(j, i)$
  pre $\exists reg : Region(head(eventqueue) = [up, j, reg])$;
  eff $S := S \cup j$;    // add node $j$ to neighbor set after doing a $link\_up$
    $regs := update(regs, j, head(eventqueue).reg)$;
    $eventqueue := tail(eventqueue)$;
```

Algorithm 5 MAC Broker

```
vocabulary BrokerTypes($M : Type$)
  types
    $MessageType : Enumeration[usr, ndp]$,
    $P : Tuple[msg : M, id : MessageType]$
  end
automaton MACBroker($i : Nat, q : Nat, M : Type$)
  imports BrokerTypes($TypeM$)
  signature
    input $bcast\_usr(m : M, consti), bcast\_ndp(m : M, consti)$
    output $rcv\_usr(m : M, consti), rcv\_ndp(m : M, consti)$
     output$bcast(m : P, consti)$
     input $ack(m : P, consti), rcv(m : P, consti)$

  states
    $usr\_out\_queue : Seq[P] := \oslash;$
    $ndp\_out\_queue : Seq[P] := \oslash;$
    $usr\_in\_queue : Seq[M] := \oslash;$
    $ndp\_in\_queue : Seq[M] := \oslash;$
    $cts : Bool := true;$

  transitions
    input $bcast\_ndp(m, i)$
      eff if $len(usr\_out\_queue) + len(ndp\_out\_queue) < q$ then
        $ndp\_out\_queue := ndp\_out\_queue \vdash [m, ndp];$

    input $bcast\_usr(m, i)$
      eff if $len(usr\_out\_queue) + len(ndp\_out\_queue) < q$ then
        $usr\_out\_queue := usr\_out\_queue \vdash [m, usr];$

    output $rcv\_ndp(m, i)$
      pre $m = head(ndp\_in\_queue);$
      eff $ndp\_in\_queue := tail(ndp\_in\_queue);$

    output $rcv\_usr(m, i)$
      pre $m = head(usr\_in\_queue);$
      eff $usr\_in\_queue := tail(usr\_in\_queue);$

    output $bcast(p, i)$
      pre $cts = true \wedge (p = head(ndp\_out\_queue) \vee (p = head(usr\_out\_queue)$
        $\wedge ndp\_out\_queue = \oslash));$
      eff $cts := false;$
        if $len(ndp\_out\_queue) > 0$ then
          $ndp\_out\_queue := tail(ndp\_out\_queue);$
        else
          $usr\_out\_queue := tail(usr\_out\_queue);$

    input $rcv(p, i)$
      eff if $p.id = ndp$then
          $ndp\_in\_queue := ndp\_in\_queue \vdash p.msg;$
        else
          $usr\_in\_queue := usr\_in\_queue \vdash p.msg;$

    input $ack(m, i)$
      eff $cts := true;$
```

VITA

Saira Viqar received her Masters in Computer Science from The National University of Computer and Emerging Sciences in Pakistan in 2006. She entered the doctoral program in the department of Computer Science and Engineering at Texas A&M University in August 2006. Her research interests include the theory of distributed systems and wireless mobile ad hoc networks and systems.

Saira Viqar can be reached by snail-mail at Department of Computer Science and Engineering, MS 3112 TAMU, College Station TX 77840. Her email address is viqar@cse.tamu.edu.