pdfauthor

Self-Stabilizing Robot Formations over Unreliable Networks

Seth Gilbert, Ecole Polytechnique Fédérale, Lausanne
Nancy Lynch, Massachusetts Institute of Technology
Sayan Mitra, California Institute of Technology
and
Tina Nolte, Massachusetts Institute of Technology

We describe how a set of mobile robots can arrange themselves on any specified curve on the plane in the presence of dynamic changes both in the underlying ad hoc network and the set of participating robots. Our strategy is for the mobile robots to implement a *self-stabilizing virtual layer* consisting of mobile client nodes, stationary Virtual Nodes (VNs), and local broadcast communication. The VNs are associated with predetermined regions in the plane and coordinate among themselves to distribute the client nodes relatively uniformly among the VNs' regions. Each VN directs its local client nodes to align themselves on the local portion of the target curve. The resulting motion coordination protocol is self-stabilizing, in that each robot can begin the execution in any arbitrary state and at any arbitrary location in the plane. In addition, self-stabilization ensures that the robots can adapt to changes in the desired target formation.

Categories and Subject Descriptors: F.1.2 [**Computation by Abstract Devices**]: Modes of Computation–*Interactive and Reactive Computation*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*

General Terms: Algorithms, Reliability

Additional Key Words and Phrases: cooperative mobile robotics, distributed algorithms, pattern formation, self-stabilization

1. INTRODUCTION

In this paper, we study the problem of coordinating the behavior of a set of autonomous mobile robots in the presence of changes in the underlying communication network as well as changes in the set of participating robots. Consider, for example, a system of firefighting robots deployed throughout forests and other arid wilderness areas. Significant levels of coordination are required in order to combat the fire: to prevent the fire from spreading, it has to be surrounded; to put out the fire, firefighters need to create "firebreaks" and spray water; they need to direct

S. Gilbert, Laboratory for Distributed Programming, Ecole Polytechnique Fédérale, Lausanne. Email: seth.gilbert@epfl.ch. N. Lynch, Dept. of Electrical Engineering and Computer Science, MIT Email:lynch@csail.mit.edu. S. Mitra, Center for Mathematics of Information, California Inst. of Tech. Supported by NSF CSR program (Embedded & Hybrid systems area) under grant NSF CNS-0614993. Email:mitras@caltech.edu. T. Nolte, Dept. of Electrical Engineering and Computer Science, MIT tnolte@mit.edu.

the actions of (potentially autonomous) helicopters carrying water. All this has to be achieved with the set of participating agents changing and with unreliable (possibly wireless) communication between agents. Similar scenarios arise in a variety of contexts, including search and rescue, emergency disaster response, remote surveillance, and military engagement, among many others. In fact, autonomous coordination has long been a central problem in mobile robotics.

We focus on a generic coordination problem that, we believe, captures many of the complexities associated with coordination in real-world scenarios. We assume that the mobile robots are deployed in a large two-dimensional plane, and that they can coordinate their actions by local communication using wireless radios. The robots must arrange themselves to form a particular pattern, specifically, a continuous curve drawn in the plane. The robots must spread themselves uniformly along this curve. In the firefighting example described above, this curve might form the perimeter of the fire.

These types of coordination problems can be quite challenging due to the dynamic and unpredictable environment that is inherent to wireless ad hoc networks. Robots may be continuously joining and leaving the system, and they may fail. In addition, wireless communication is notoriously unreliable due to collisions, contention, and various wireless interference.

Recently, *virtual infrastructure* has been proposed as a new tool for building reliable and robust applications in unreliable and unpredictable wireless ad hoc networks (e.g., [Dolev et al. 2003; Dolev et al. 2005; Chockler et al. 2008]). The basic principle motivating virtual infrastructure is that many of the challenges resulting from a dynamic networks could be obviated if there were reliable network infrastructure available. Unfortunately, in many contexts, such infrastructure is unavailable. The virtual infrastructure abstraction emulates real reliable infrastructure in ad hoc networks. Thus, it provides a programming abstraction for developing applications that *assumes* reliable communication infrastructure. It has already been observed that virtual infrastructure simplifies several problems in wireless ad hoc networks, including distributed shared memory implementations [Dolev et al. 2003], tracking mobile devices [Nolte and Lynch 2007b], geographic routing [Dolev et al. 2005b], and point-to-point routing [Dolev et al. 2004].

In this paper, we rely on a virtual infrastructure known as the Virtual Stationary Automata Layer (VSA Layer) [Dolev et al. 2005a; Nolte and Lynch 2007a]. In the VSA Layer, each robot is modelled as a *client*; clients interact with *virtual stationary automata* (VSAs) via a (virtual) communication service. VSAs are distributed throughout the world, each assigned to its own unique region. VSAs remain always at a known and predictable location, and they are less likely to fail than any individual mobile robot. Notice that the VSAs do not actually exist in the real world; they are emulated by the underlying mobile robots. The VSA layer is envisioned as a programming abstraction emulated by some underlying set of broadcast-equipped physical devices, such as mobile robots, with access to a time and location information.

Our main contribution is that we show how to use the VSA Layer to implement a reliable and robust protocol for coordinating mobile robots. The protocol relies on the VSAs to organize the mobile robots in a consistent fashion. Each VSA must decide based on its own local information which robots to keep in its own region, and which to assign to neighboring regions; for each robot that remains, the VSA determines where on the curve the robot should reside. In order that the robot coordination be truly robust, our coordination protocol is *self-stabilizing*, meaning that each robot can begin in an arbitrary state, in an arbitrary location in the network, and yet the distribution of the robots will still converge to the specified curve. When combined with a self-stabilizing implementation of the VSA Layer, as is presented in [Dolev et al. 2005a; Nolte and Lynch 2007a], we end up with entirely self-stabilizing solution for the problem of autonomous robot coordination.

Self-stabilization provides many advantages. First, given the unreliable nature of wireless networks, it is possible that occasionally (due to aberrant interference) a significant fraction of messages may be lost, disrupting the protocol; a self-stabilizing algorithm can readily recover from this. Second, a self-stabilizing algorithm can cope with more dynamic coordination problems. In real-life scenarios, the required formation of the mobile nodes may change. In the firefighting example above, as the fire advances or retreats, the formation of firefighting robots must adapt. A self-stabilizing algorithm can adapt to these changes, continually re-arranging the robots along the newly chosen curve.

A second technical contribution of this paper is the exemplification of a proof technique for showing self-stabilization of systems implemented using virtual infrastructure. The proof technique has three parts. First, using invariant assertions and standard control theory results we show that from any initial state, the application protocol, in this case, the motion coordination algorithm converges to an *acceptable state*. Next, we show that the algorithm always reaches a *legal state* even when it starts from some arbitrary state after failures. From any legal state the algorithm gets to an acceptable state provided there are no further failures. Finally, using a simulation relation we show that the above set of legal states is in fact equal to the set of reachable states of the complete system—the coordination algorithm composed with the VSA layer. It has already been shown in [Dolev et al. 2005a; Nolte and Lynch 2007a] that the VSA layer itself is self-stabilizing. Thus, combining the stabilization of the VSA layer and the application protocol, we are able to conclude self-stabilization of the complete system.

The remainder of this paper is organized as follows. First, in Section 2, we discuss some of the related work. Next, in Section 3, we introduce the underlying mathematical model used for specifying the VSA layer. In Section 4 we discuss the VSA Layer model. In Section 5 we describe the motion coordination problem, and our algorithm that solves it. In Section 6, we show that the algorithm is correct, and in Section 7, we show that the algorithm is self-stabilizing.

2. RELATED WORK

In the distributed computing literature, a self-stabilizing system is one which regains normal functionality and behavior sometime after disturbances, such as node failures and message losses cease [Dolev 2000]. The idea of self-stabilization has been widely employed for designing resilient distributed systems over unreliable communication and computing components (see [Herman 1996] for a comprehensive list of applications).

The problem of motion coordination has been studied in a variety of contexts, focusing on several different goals: flocking [Jadbabaie et al. 2003]; rendezvous [Ando et al. 1999; Lin et al. 2003; Martinez et al. 2005]; aggregation [Gazi and Passino 2003]; deployment and regional coverage [Cortes et al. 2004]. Control theory literature contains several algorithms for achieving spatial patterns [Fax and Murray 2004; Clavaski et al. 2003; Blondel et al. 2005; Olfati-Saber et al. 2007]. These algorithms assume that the agents process information and communicate synchronously, and hence, they are analyzed based on differential or difference equations models of the system. Convergence of this class of algorithms over unreliable and delay-prone communication channels have been studied recently in [Chandy et al. 2008].

Geometric pattern formation with vision-based models for mobile robots have been investigated in [Suzuki and Yamashita 1999; Prencipe 2001; Flocchini et al. 2001; Efrima and Peleg 2007; Prencipe 2000; Défago and Konagaya 2002]. In these weak models, the robots are oblivious, identical, anonymous, and often without memory of past actions. For the memoryless models, the algorithms for pattern formation are often automatically self-stabilizing. In [Défago and Konagaya 2002; Défago and Souissi 2008], for instance, a self-stabilizing algorithm for forming a circle has been presented. These weak models have been used for characterizing the class of patterns that can be formed and for studying the computational complexity of formation algorithms, under different assumptions about the level of common knowledge amongst agents, such as, knowledge of distance, direction, and coordinates [Suzuki and Yamashita 1999; Prencipe 2000].

We have previously presented a protocol for coordinating mobile devices using virtual infrastructure in [Lynch et al. 2005]. The paper described how to implement a simple asynchronous virtual infrastructure, and proposed a protocol for motion coordination. This earlier protocol relies on a weaker (i.e., untimed) virtual layer (see [Dolev et al. 2005a; Nolte and Lynch 2007a]), while the current relies on a stronger (i.e., timed) virtual layer. As a result, our new coordination protocol is somewhat simpler and more elegant than the previous version. Moreover, the new protocol is self-stabilizing, which allows both for better fault-tolerance, and also the ability to tolerate dynamic changes in the desired pattern of motion. Virtual infrastructure has also been considered in [Brown 2007] for collision prevention of airplanes.

3. PRELIMINARIES

In this paper we mathematically model the the virtual infrastructure, the motion of the robots, and the motion coordination protocols using the Timed Input/Output Automata (TIOA) framework. TIOA is a formal modelling framework for real-time, distributed systems where computing and physical processes interact. Here we define the key concepts in the framework and refer the reader to [Kaynar et al. 2005] for details.

3.1 Timed I/O Automata

A Timed I/O Automaton is a non-deterministic state transition system in which the state may change either (a) instantaneously through a transition, or (b) continuously over an interval of time following a *trajectory*. Let V be a set of variables. Each variable $v \in V$ is associated with a *type* which defines the set of values v

can take. The set of valuations of V is denoted by $val(V)$. Each variable may be discrete or continuous. Discrete variables are used to model protocol state, data structures, while continuous variables are used to model physical quantities such as time, position, and velocity.

The semi-infinite real line $\mathbb{R}_{\geq 0}$ is used to model time. A *trajectory* for a set of variables V maps a left-closed interval of $\mathbb{R}_{\geq 0}$ with left endpoint 0 to $val(V)$. It models continuous evolution of values of the variables. The domain τ is denoted by $\tau.dom$. A trajectory is *closed* if $\tau.dom = [0, t]$ for some $t \in \mathbb{R}_{\geq 0}$, in which case we define $\tau.ltime \triangleq t$ and $\tau.lstate \triangleq \tau(t)$.

Definition 3.1. *A TIOA $\mathcal{A} = (X, Q, \Theta, A, \mathcal{D}, \mathcal{T})$ consists of (a) A set X of* variables*. (b) A set $Q \subseteq val(V)$ of* states*. (c) A set $\Theta \subseteq S$ of* start states*. (d) A set A of* actions *partitioned into* input, output *and* internal *actions I, O, and H, (e) A set $\mathcal{D} \subseteq S \times A \times S$ of* discrete transitions*. An action $a \in A$ is said to be* enabled *at $\mathbf{x}$ iff $(\mathbf{x}, a, \mathbf{x}') \in \mathcal{D}$, and we write this as $\mathbf{x} \xrightarrow{a} \mathbf{x}'$. (f) A set $\mathcal{T}$ of trajectories for V that is closed*[1] *under prefix, suffix and concatenation. In addition, $\mathcal{A}$ must be input action and input trajectory enabled.*

For a TIOA $\mathcal{A}$, we refer to its components $X, Q, \mathcal{D}$, etc., by $X_{\mathcal{A}}, Q_{\mathcal{A}}, \mathcal{D}_{\mathcal{A}}$, respectively. And for TIOA $\mathcal{A}_1$, we refer to the components by $X_1, Q_1, \mathcal{D}_1$, etc.

Executions. An execution of $\mathcal{A}$ records the valuations of all its variables and the occurrences of all actions over a particular run. An *execution fragment* of $\mathcal{A}$ is a finite or infinite sequence $\tau_0 a_1 \tau_1 a_2 \ldots$, such that for all i in the sequence, $\tau_i.lstate \xrightarrow{a_{i+1}} \tau_{i+1}(0)$. An execution fragment is an *execution* if $\tau_0(0) \in \Theta$. The first state of α, $\alpha.fstate$, is $\tau_0(0)$, and for a closed α, its last state, $\alpha.lstate$, is the last state of its last trajectory. The *limit time* of α, $\alpha.ltime$, is defined to be $\sum_i \tau_i.ltime$. The set of executions and reachable states of $\mathcal{A}$ are denoted by $\mathsf{Execs}_{\mathcal{A}}$ and $\mathsf{Reach}_{\mathcal{A}}$.

A nonempty set of states $L \subseteq Q_{\mathcal{A}}$ is said to be a *legal set* for $\mathcal{A}$ if it is closed under the transitions and the closed trajectories of $\mathcal{A}$. That is, (a) if $\mathbf{x} \xrightarrow{a} \mathbf{x}'$ and $\mathbf{x} \in L$, then $\mathbf{x}' \in L$, and (b) if $\tau \in \mathcal{T}_{\mathcal{A}}$, τ is closed, and $\tau(0) \in L$ then $\tau.lstate \in L$. A set of states $I \subseteq S$ is said to be an *invariant* of $\mathcal{A}$ iff $\mathsf{Reach}_{\mathcal{A}} \subseteq I$. An invariant set I captures the notion that the states outside I are never reached by the TIOA $\mathcal{A}$. It is easy to check that if L is a legal and $\Theta_{\mathcal{A}} \subseteq L$, then L is an invariant.

Traces. Often we are interested in studying the externally visible behavior of a TIOA $\mathcal{A}$, instead of its execution. The visible behavior or the *trace* corresponding to a given execution α is obtained by (a) removing all internal actions, and (b) replacing each trajectory with just its domain. Thus, the trace of an execution α, denoted by $trace(\alpha)$, has information about input/output actions and the duration of time that elapses between the occurrence of successive actions. The set of traces of $\mathcal{A}$ is defined as $\mathsf{Traces}_{\mathcal{A}} \triangleq \{\beta \mid \exists \alpha \in \mathsf{Execs}_{\mathcal{A}}, trace(\alpha) = \beta\}$.

Implementation. Our proof techniques often rely on showing that any behavior of a given TIOA $\mathcal{A}$ is externally indistinguishable from some behavior of another TIOA $\mathcal{B}$. This is formalized by the notion of implementation which we define next. Two TIOAs are said to be *comparable* if their external interfaces are identical, that

[1] See Sections 3-4 of [Kaynar et al. 2005] for formal definitions of these closure properties.

is, they have the same input and output actions. Given two comparable TIOAs $\mathcal{A}$ and $\mathcal{B}$, $\mathcal{A}$ is said to *implement* $\mathcal{B}$, if $\mathsf{Traces}_{\mathcal{A}} \subseteq \mathsf{Traces}_{\mathcal{B}}$. The standard technique for proving that $\mathcal{A}$ implements $\mathcal{B}$ is to come up with a *simulation relation* $\mathcal{R} \subseteq Q_{\mathcal{A}} \times Q_{\mathcal{B}}$ which satisfies the following condition: if $\mathbf{x}\mathcal{R}\mathbf{y}$, then every one-step move of $\mathcal{B}$ from a state $\mathbf{y}$, can be simulated by some execution fragment of $\mathcal{A}$ starting from $\mathbf{x}$, such that (a) the corresponding final states are also related by $\mathcal{R}$, and (b) the traces of the moves are identical (see [Kaynar et al. 2005] for the formal definition).

Composition. It is convenient to model a complex system, such as our VSA layer, as a collection of TIOAs running in parallel and interacting through input and output actions. A pair of TIOAs are said to be *compatible* if they do not share variables, internal actions, or output actions. Given compatible TIOAs $\mathcal{A}$ and $\mathcal{B}$, their *composition* is another TIOA which is denoted by $\mathcal{A}\|\mathcal{B}$.

3.2 Failure transform for TIOAs

In order to model failure of robots and self-stabilization in the face of failures and recoveries, we introduce a general *failure transformation* of TIOAs, such that the transformed TIOAs can be crashed and restarted.

A TIOA $\mathcal{A}$ is said to be is *fail-transformable* if it does not have any variable called $failed$, and it does not have actions called fail and restart. The transformed automaton *Fail*($\mathcal{A}$) has one additional discrete state variable, *failed*, indicating whether or not the machine is failed, and two additional input actions, fail and restart. The states of the new automaton are states of the old automaton, together with a valuation of *failed*. The start states are defined to be ones where *failed* is arbitrary, but if *failed* is false then the rest of the variables are set to values consistent with a start state of $\mathcal{A}$. The transitions for $Fail(\mathcal{A})$ are derived from those of $\mathcal{A}$ as follows: (a) transitions on input actions from a failed state leaves the state unchanged, (b) transitions from unfailed states remain the same as in $\mathcal{A}$, (c) a fail action sets $failed$ to true, (d) if a restart action occurs at a failed state then $failed$ is set to false and all other state variables are set to arbitrary initial values, otherwise it does not change the state. The set of trajectories of $Fail(\mathcal{A})$ can be divided into two sets of trajectories based on the value of the *failed* variable. If *failed* is false over the course of the trajectory τ, then τ is such that τ restricted to $X_{\mathcal{A}}$ is a trajectory of $\mathcal{A}$. While $Fail(\mathcal{A})$ is not failed its trajectories basically look like those of $\mathcal{A}$ with the value of the *failed* variable remaining constant. If the machine is failed then all variables are constant over trajectories. This means that if the machine is failed, then its state variables are frozen. This does not constrain time from passing– any constant trajectory is allowed.

Performing a *Fail*-transform on the composition $\mathcal{A}_1\|\mathcal{A}_2$ of two automata results in a TIOA whose executions constrained to actions and variables of $Fail(\mathcal{A}_1)$ or $Fail(\mathcal{A}_2)$ are executions of $Fail(\mathcal{A}_1)$ or $Fail(\mathcal{A}_2)$ respectively.

3.3 Self-Stabilization of TIOAs

A self-stabilizing system is one which regains normal functionality and behavior sometime after disturbances cease. For a given TIOA $\mathcal{A}$, suppose L is a legal set and further, assume that all execution fragments starting from L correspond to normal behavior. Then, $\mathcal{A}$ is self-stabilizing with respect to L if any execution fragment of $\mathcal{A}$ starting from an *arbitrary state* ultimately reaches some state in L. Starting from arbitrary states captures the possibility of starting from states where disturbances, such as failures and restarts, have just occurred.

Throughout this section A, A_1, A_2, etc., are sets of actions and V is a set of variables. An (A, V)*-sequence* is a (possibly infinite) alternating sequence of actions in A and trajectories of V. An (A, V)-sequence is *closed* if it is finite and its final trajectory is closed.

Definition 3.2. *Given (A, V)-sequences α, α' and $t \geq 0$, α' is a* t-suffix of α *if there exists a closed (A, V)-sequence α'' of duration t such that $\alpha = \alpha''\alpha'$. α' is a* state-matched t-suffix of α *if it is a t-suffix of α, and $\alpha'.fstate$ equals the $\alpha''.lstate$.*

Informally, α' is a state-matched t suffix of α if there exists a closed fragment of duration t, with the same last state as the first state of α' and which when prefixed to α' equals to α. One set of executions or traces (say, behavior including failures and message losses) self-stabilizes to another set (say, desirable behavior) in time t if each state-matched t-suffix of each behavior in the former set is included the latter set.

Definition 3.3. *Given a set of (A_1, V)-sequences S_1, a set of (A_2, V)-sequences S_2, and $t \geq 0$, S_1 is said to* stabilize in time t *to S_2 if each state-matched t-suffix α of each sequence in S_1 is in S_2.*

The *stabilizes to* relation is transitive as stated by the following lemma.

Lemma 3.4. *Let S_i be a set of (A_i, V)-sequences, for $i \in \{1, 2, 3\}$. If S_1 stabilizes to S_2 in time t_1, and S_2 stabilizes to S_3 in time t_2, then S_1 stabilizes to S_3 in time $t_1 + t_2$.*

The following definitions are necessary for starting TIOAs at arbitrary states: For any $L \subseteq Q_{\mathcal{A}}$, $Start(\mathcal{A}, L)$ is defined to be the TIOA that is identical to $\mathcal{A}$ except that $\Theta_{Start(\mathcal{A},L)} = L$. We define $U(\mathcal{A}) \triangleq Start(\mathcal{A}, Q_{\mathcal{A}})$ and $R(\mathcal{A}) \triangleq Start(\mathcal{A}, \mathsf{Reach}_{\mathcal{A}})$. It is straightforward to check that for any TIOA $\mathcal{A}$, $Fail$ and U operators are interchangeable. Finally we define self-stabilization of composed TIOAs.

Definition 3.5. *Let $\mathcal{B}$ and $\mathcal{A}$ be compatible TIOAs, and L be a legal set for the composed TIOA $\mathcal{A}\|\mathcal{B}$. $\mathcal{A}$* self-stabilizes in time t to L relative to $\mathcal{B}$ *if the set of executions of $U(\mathcal{A})\|\mathcal{B}$ stabilizes in time t to executions of $Start(\mathcal{A}\|\mathcal{B}, L)$.*

4. VIRTUAL STATIONARY AUTOMATA

The Virtual Stationary Automata (VSA) infrastructure has been presented earlier in [Dolev et al. 2005a; Nolte and Lynch 2007a]. The VSA infrastructure can be seen as an abstract system model implemented in middleware, thus providing a simpler and more predictable programming model for the application developer. The main components of the VSA layer are (1) Virtual Stationary Automata (VSA), (2) Client Nodes, (3) Real world (RW) and Virtual World (VW) automata, (4) $VBDelay$ buffers, and (5) $VBcast$ broadcast service. The interaction of these components shown in Figure 1. Each of these components are formally modeled as TIOAs, and the complete system is the composition of the component TIOAs or the corresponding *fail* transformed TIOAs, as the case may be. First, we informally describe the architecture of this layer and then briefly sketch its implementation.

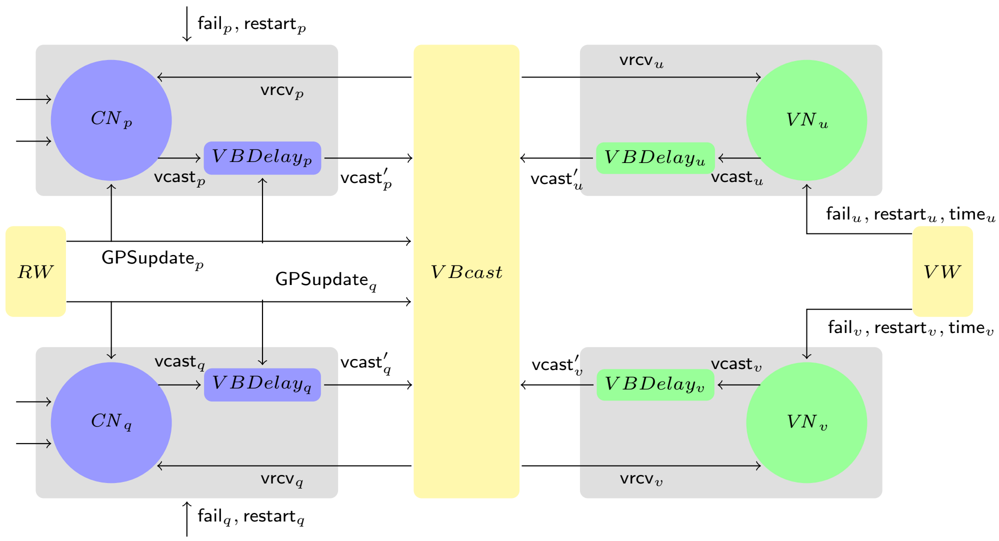

Fig. 1. Virtual Stationary Automata layer.

4.1 VSA Architecture

For the remainder of this paper, we fix R to be a closed, bounded and connected subset of $\mathbb{R}^2$, U to be a totally ordered index set, and P to be another index set. R models the physical space in which the robots reside; we call it the *deployment space*. U and P serve as the index sets for regions in R and for the participating robots, respectively.

Network tiling. A network tiling divides the deployment space R into a set of *regions* $\{R_u\}_{u \in U}$, such that: (i) for each $u \in U$, R_u is a closed, connected subset of R, and (ii) for any $u, v \in U$, R_u and R_v may overlap only at their boundaries. For any $u, v \in U$, the corresponding regions are said to be *neighbors* if $R_u \cap R_v \neq \emptyset$. This neighborhood relation, $nbrs$, induces a graph on the set of regions. We assume that the network tiling divides R in such a way that the resulting graph is connected. For any $u \in U$, we denote the ids of its neighboring regions by $nbrs(u)$, and $nbrs^+(u) \triangleq nbrs(u) \cup \{u\}$. We define the distance between two regions u and v, denoted by $regDist(u, v)$, as the number of hops on the shortest path between u and v in the graph. The diameter of the graph, i.e., the distance between the farthest regions in the tiling, is denoted by D, and the largest Euclidean distance between any two points in any region is denoted by r.

One example of a network tiling is the *grid tiling*, where R is divided into $b \times b$ square regions, for some constant $b > 0$. Non-border regions in this tiling have have eight neighbors. For a grid tiling with a given b, r could be any value greater than or equal to $2\sqrt{2}\ b$.

Real World (RW) *Automaton.* RW is an external source of occasional but reliable time and location information for participating robots. The RW automaton is parameterized by: (a) $v_{max} > 0$, a maximum speed, and (b) $\epsilon_{sample} > 0$, a maximum time gap between successive updates for each robot. The RW automaton maintains three key variables: (a) a continuous variable *now* representing true system time; *now* increases monotonically at the same rate as real-time starting from 0. (b) An array $vel[P \rightarrow R \cup \{\perp\}]$; for $p \in P$, $vel(p)$ represents the current velocity of robot p. Initially $vel(p)$ is set to $\perp$, and it is updated by the robots when their velocity changes. (c) an array $loc[P \rightarrow R]$; for $p \in P$, $loc(p)$ represents the current location of robot p. Over any interval of time, robot p may move arbitrarily in R provided its path is continuous and its maximum speed is bounded by v_{max}. Automaton RW performs the $\mathsf{GPSupdate}(l,t)_p$ action, $l \in R, t \in \mathbb{R}_{\geq 0}, p \in P$, to inform robot p about its current location and time. For each p, some $\mathsf{GPSupdate}(,)_p$ action must occur every ϵ_{sample} time.

Virtual World (VW) *Automaton.* VW is an external source of occasional but reliable time information for VSAs. Similar to RW's GPSupdate action for clients, VW performs $\mathsf{time}(t)_u$ output actions notifying VSA u of the current time. One such action occurs at time 0, and they are repeated at least every ϵ_{sample} time thereafter. Also, VW nondeterministically issues fail_u and $\mathsf{restart}_u$ outputs for each $u \in U$, modelling the fact that VSAs may fail and restart.

Mobile client nodes. For each $p \in P$, the mobile client node CN_p is a TIOA modeling the client-side program executed by the robot with identifier p. CN_p has a local clock variable, *clock* that progresses at the rate of real-time, and is initially $\perp$. CN_p may have arbitrary local non-*failed* variables. Its external interface at least includes the GPSupdate inputs, $\mathsf{vcast}(m)_p$ outputs, and $\mathsf{vrcv}(m)_p$ inputs. CN_p may have additional arbitrary non-fail and non-restart actions. An example of a client node appears in Figure 2.

Virtual Stationary Automata (VSAs). A VSA is a clock-equipped abstract virtual machine. For each $u \in U$, there is a corresponding VSA VN_u which is associated with the geographic region R_u. VN_u has a local clock variable *clock* which progresses at the rate of real-time (it is initially $\perp$ before the first time input). VN_u has only the following external interface: (a) **Input** $\mathsf{time}(t)_u, t \in \mathbb{R}^{\geq 0}$: models a time update at time t; it sets node VN_u's *clock* to t. (b) **Output** $\mathsf{vcast}(m)_u, m \in Msg$: models VN_u broadcasting message m; (c) **Input** $\mathsf{vrcv}(m)_u, m \in Msg$: models VN_u receiving a message m. VN_u may have additional arbitrary non-*failed* variables and non-fail and non-restart internal actions. All such actions must be deterministic.

VBDelay *Automata.* Each client and each VSA node, is associated with a *VBDelay* buffer that delays messages when they are broadcast for up to e time. This buffer takes as input a $\mathsf{vcast}(m)$ from the node, and passes the message on to the *VBcast* service after some interval of time at most e. In the case of VSA nodes, the message is passed on immediately to the *VBcast* service with no delay.

VBcast *Automaton.* Each client and virtual node has access to the virtual local broadcast communication service *VBcast*. The service is parameterized by a constant $d > 0$ which models the upper bound on message delays. *VBcast* takes each $\mathsf{vcast}'(m,f)_i$ input from client and virtual node delay buffers and delivers the message m via $\mathsf{vrcv}(m)$ at each client or virtual node. It delivers the message to every client and VSA that is in the same region as the initial sender, when the message was first sent, along with those in neighboring regions. The *VBcast* service guarantees that in each execution α of *VBcast* there is a correspondence between $\mathsf{vrcv}(m)$ actions and $\mathsf{vcast}'(m,f)_i$ actions, such that: (i) each vrcv occurs *after and within d time* of the corresponding vcast′, (ii) at most one vrcv at a particular process is mapped to each vcast′. (iii) a message originating from some region u must be received by all robots that are in R_u or its neighbors throughout the transmission period.

A VSA layer *algorithm* or a *V-algorithm* is an assignment of a TIOA program to each client and VSA. We denote the set of all V-algorithms is as $VAlgs$. Since we are interested in providing this layer using failure-prone robots, we then define a *VLayer*, a VSA layer with failure-prone clients and VSAs, i.e., one in which each client is modified so as to fail by crashing.

Definition 4.1. *Let alg be an element of* $Valgs$. $VLNodes[alg]$, *the* fail-transformed *nodes of the VSA layer parameterized by alg, is the composition of* $Fail(alg(i))$ *with a* VBDelay *buffer, for all* $i \in P \cup U$. $VLayer[alg]$, *the* VSA layer parameterized by *alg, is the composition of* $VLNodes[alg]$ *with* $RW \| VW \| VBcast$.

4.2 VSA Layer Implementation

In [Dolev et al. 2005a; Nolte and Lynch 2007a], we show how mobile nodes can emulate the VSA Layer in a wireless network; additional details of this implementation are in the [Nolte 2008]. The emulation algorithm is based on a replicated-state-machine paradigm, where there is a leader (i.e., a "primary") that is responsible for maintaining the state, and that the replicas alternate being the leader. The key feature of our replicated state machine is that it guarantees certain timing properties, and therefore, the emulation algorithm has to ensure that these timing properties are respected.

Mobile robots in a region R_u use a leader-based emulation algorithm to implement the region u's virtual node. Each mobile robot runs a totally ordered broadcast service, *TOBcast*, leader election service, and a Virtual Node Emulation (*VNE*) algorithm, for each virtual node. The *TOBcast* service ensures that each *VNE* in the same region receives the same set of messages in the same order. Assuming mobile robots are equipped with a real local broadcast service *Pbcast*, with communication radius $R_p \geq \sqrt{5}b$ and message delay d_p, *TOBcast* is implemented using a hold strategy for received messages, where robots do not "receive" a message until enough ($d_p + \epsilon$, ϵ small) time has passed that all other robots in the region will have received the message as well. Each *VNE* then independently maintains the state of the region's virtual node.

Periodically a leader is selected in a zone by the leader election service. This service is implemented by having each mobile robot in a region periodically send out a message indicating its id, its region, and whether or not it is currently participating in the emulation of the region's VSA. The leader of a region is selected from amongst these processes in its region based first on whether it is participating in the region's VSA emulation (robots that indicate they are participating have priority), and then on the basis of process id (robots with lower process ids are preferred).

A leader is responsible for both broadcasting the messages that would have been sent by the virtual machine in its region in the last e time, where e is the $VBDelay$ buffer delay parameter, and broadcasting an up-to-date version of the VSA state. This broadcast is used to both stabilize the state of the emulation algorithm, forcing all emulators in the same region to have the same virtual machine state, and to allow newly joining emulators (those that have just restarted or moved into the region) to start participating in emulation. This virtual machine state is frozen from the point of the sending of this virtual machine state message, until the mobile robots again participate in the leader election service. During that time, the virtual machine runs at an accelerated pace, simulating the receipt of messages received from $TOBcast$ while doing so, until the machine is caught up with real-time and the next leader is chosen. Any broadcasts that this emulation of the virtual machine produces are stored in a local outgoing queue for broadcast if the emulator becomes a leader.

5. MOTION COORDINATION USING VIRTUAL NODES

In this paper we fix $\Gamma : A \to R$ to be a simple, differentiable curve on R that is parameterized by arc length. The domain set A of parameter values is an interval in the real line. We also fix a particular network tiling given by the collection of regions $\{R_u\}_{u \in U}$ such that each point in Γ is also in some region R_u. Let $A_u \triangleq \{p \in A : region(\Gamma(p)) = u\}$ be the domain of Γ in region u. We assume that A_u is convex for every region u; it may be empty for some u. The local part of the curve Γ in region u is the restriction $\Gamma_u : A_u \to R_u$. We write $|A_u|$ for the length of the curve Γ_u. We define the *quantization* of a real number x with quantization constant $\sigma > 0$ as $q_\sigma(x) = \lceil \frac{x}{\sigma} \rceil \sigma$. We fix σ, and write q_u as an abbreviation for $q_\sigma(|A_u|)$, q_{min} for the minimum nonzero q_u, and q_{max} for the maximum q_u.

5.1 Problem Statement

Our goal is to design an algorithm for mobile robots such that, once the failures and recoveries cease, within finite time all the robots are located on Γ and as time progresses they eventually become equally spaced on Γ. Formally, if no fail and restart actions occur after time t_0, then:

(1) there exists a constant T, such that for each $u \in U$, within time $t_0 + T$ the set of robots located in R_u becomes fixed and its cardinality is roughly proportional to q_u; moreover, if $q_u \neq 0$ then the robots in R_u are located on[2] Γ_u, and

(2) in the limit, as time goes to infinity, all robots in R_u are uniformly spaced[3] on Γ_u.

5.2 Overview of Solution: Motion Coordination Algorithm (MC)

The VSA Layer is used as a means to coordinate the movement of client nodes, i.e., robots. A VSA controls the motion of the clients in its region by setting and broadcasting target waypoints for the clients: VSA VN_u, $u \in U$, periodically receives

[2]For a given point $x \in R$, if there exists $p \in A$ such that $\Gamma(p) = x$, then we say that the point x is on the curve Γ; abusing the notation, we write this as $x \in \Gamma$.

[3]A sequence $x_1, \ldots, x_n$ of points in R is said to be *uniformly spaced* on a curve Γ if there exists a sequence of parameter values $p_1 < p_2 \ldots < p_n$, such that for each i, $1 \leq i \leq n$, $\Gamma(p_i) = x_i$, and for each i, $1 < i < n$, $p_i - p_{i-1} = p_{i+1} - p_i$.

information from clients in its region, exchanges information with its neighbors, and sends out a message containing a calculated target point for each client node "assigned" to region u. VN_u performs two tasks when setting the target points: (1) it re-assigns some of the clients that are assigned to itself to neighboring VSAs, and (2) it sends a target position on Γ to each client that is assigned to itself. The objective of (1) is to prevent neighboring VSAs from getting depleted of robots and to achieve a distribution of robots over the regions that is proportional to the length of Γ in each region. The objective of (2) is to space the nodes uniformly on Γ within each region. The client algorithm, in turn, receives its current position information from RW and computes a velocity vector for reaching its latest received target point from a VSA.

Each virtual node VN_u uses only information about the portions of the target curve Γ in region u and neighboring regions. For the sake of simplicity, we assume that all client nodes know the complete curve Γ. We could as well have modeled the client nodes in u as receiving external information about the nature of the curve in region u and neighboring regions only.

5.3 Client Node Algorithm (CN)

The algorithm for the client node $CN(\delta)_p$, $p \in P$ (see Figure 2) follows a round structure, where rounds begin at times that are multiples of δ. At the beginning of each round, a CN stops moving and sends a cn-update message to its local VSA (that is, the VSA in whose region the CN currently resides). The cn-update message tells the local VSA the CN's *id* and its current location in R. The local VN then sends a response to the client, i.e., a target-update message. Each such message describes the new target location x_p^* for CN_p, and possibly an assignment to a different region. CN_p computes its velocity vector v_p, based on its current position x_p and its target position x_p^*, as $v_p = (x_p - x_p^*)/||x_p - x_p^*||$ and communicates $v_{max} v_p$ to RW. As a result then RW moves the position of CN_p (with maximum velocity) towards x_p^*.

5.4 Virtual Stationary Node Algorithm (VN)

The algorithm for virtual node $VN(k, \rho_1, \rho_2)_u$, $u \in U$, appears in Figure 3, where $k \in \mathbb{Z}^+$ and $\rho_1, \rho_2 \in (0, 1)$ are parameters of the TIOA. VN_u collects cn-update messages sent at the beginning of the round from CN's located in region R_u, and aggregates the location and round information in a table, M. When $d + \epsilon$ time passes from the beginning of the round, VN_u computes from M the number of client nodes assigned to it that it has heard from in the round, and sends this information in a vn-update message to all of its neighbors.

When VN_u receives a vn-update message from a neighboring VN, it stores the CN population information in a table, V. When $e + d + \epsilon$ time from the sending of its own vn-update passes, VN_u uses the information in its tables M and V about the number of CNs in its and its neighbors' regions to calculate how many CNs assigned to itself should be reassigned and to which neighbor. This is done through the assign function, and these assignments are then used to calculate new target points for local CNs through the calctarget function (see Figure 4).

If the number of CNs assigned to VN_u exceeds the minimum *safe number* k, then assign reassigns some CNs to neighbors. Let In_u denote the set of neighboring VNs

Signature:
  Input GPSupdate$(l,\ t)_p,\ l \in R,\ t \in \mathbb{R}^{\geq 0}$
  Input vrcv$(m)_p,\ m \in \{$target-update$\} \times (P \rightarrow R)$
  Output vcast$(\langle$cn-update$,\ p,\ l\rangle)_p,\ l \in R$
  Output velocity$(v)_p,\ v \in R^2$

State:
  analog $clock$: $\mathbb{R}^{\geq 0} \cup \{\perp\}$, initially $\perp$
  analog $x \in R \cup \{\perp\}$, location, initially $\perp$
  $x^* \in R \cup \{\perp\}$, target point, initially $\perp$
  $v \in \{\perp, 0\} \cup \{v : \mathbb{R}^2 \mid |v| = 1\}$, *initially* $\perp$

Trajectories:
  evolve
  if $clock \neq \perp$
    then $\mathbf{d}(clock) = 1$ **else** $\mathbf{d}(clock) = 0$
  if $v \neq \perp$
    then $\mathbf{d}(x) = v \cdot v_{max}$ **else** $\mathbf{d}(x) = 0$
  stop when $[x \neq \perp \wedge x^* \neq \perp$
      $\wedge\ clock$ **mod** $\delta = 0]$
    $\vee [x \neq \perp \wedge x^* \neq \perp \wedge v||x^* - x|| \neq x^* - x]$
    $\vee [(x = x^* \vee x = \perp \vee x^* = \perp) \wedge v \neq 0]$

Transitions:

Effect
if $\langle x,\ clock\rangle \neq \langle l,\ t\rangle \vee$
  $\|x^*\text{-}l\| \geq v_{max}(\delta\lceil t/\delta\rceil\text{-}t\text{-}d_r) \vee$
  $x^* = \perp \vee\ t$ **mod** $\delta \notin (e{+}2d{+}2\epsilon,\ \delta\text{-}d_r)$
then $x, x^* \leftarrow l$; $clock \leftarrow t$
$v \leftarrow \perp$

Input vrcv$(\langle$target-update$,\ target\rangle)_p$
Effect
if $\|target(p)\text{-}x\| <\ v_{max}(\delta\lceil \frac{clock}{\delta}\rceil\text{-}clock\text{-}d_r)$
    $\wedge\ clock$ **mod** $\delta > e + 2d + 2\epsilon$
then $x^* \leftarrow target(p)$

Output vcast$(\langle$cn-update$,\ p,\ x\rangle)_p$
Precondition
  $x = x \neq \perp \wedge\ clock$ **mod** $\delta = 0 \wedge x^* \neq \perp$
Effect
  $x^* \leftarrow \perp$

Output velocity$(v)_p$
Precondition
  $v = v_{max} \cdot (x^* - x)/||x^* - x||$
  $\vee\ (v = 0 \wedge [x = x^* \vee x^* = \perp \vee x = \perp])$
Effect
  $v \leftarrow v\ /\ v_{max}$

Fig. 2. Client node $CN(\delta)_p$ automaton.

Signature:
  Input time$(t)_u,\ t \in \mathbb{R}^{\geq 0}$
  Input vrcv$(m)_u$,
    $m \in (\{$cn-update$\} \times P \times R) \cup (\{$vn-update$\} \times U \times \mathbb{N})$
  Output vcast$(m)_u$,
    $m \in (\{$vn-update$\} \times \{u\} \times \mathbb{N}) \cup (\{$target-update$\} \times (P \rightarrow R))$

State:
  analog $clock$: $\mathbb{R}^{\geq 0} \cup \{\perp\}$, *initially* $\perp$.
  M:$P \rightarrow R$, *initially* $\emptyset$.
  $V : U \rightarrow \mathbb{N}$, *initially* $\emptyset$.

Trajectories:
  evolve
  if $clock \neq t$
    then $\mathbf{d}(clock) = 1$ **else** $\mathbf{d}(clock) = 0$
  stop when Any precondition is satisfied.

Transitions:
  Input time$(t)_u$
  Effect

if $clock \neq t \vee t$ **mod** $\delta \notin (0,\ e + 2d + 2\epsilon]$
  then $M,\ V \leftarrow \emptyset$; $clock \leftarrow t$

Input vrcv$(\langle$cn-update$,\ id,\ loc\rangle)_u$
Effect
if $u = region(loc) \wedge clock$ **mod** $\delta \in (0,\ d]$
  then $M(id) \leftarrow loc$; $V \leftarrow \emptyset$

Output vcast$(\langle$vn-update$,\ u,\ n\rangle)_u$
Precondition
  $clock$ **mod** $\delta = d{+}\epsilon$
  $n = |M| \neq 0 \wedge V \neq \{\langle u,\ n\rangle\}$
Effect
  $V \leftarrow \{\langle u,\ n\rangle\}$

Input vrcv$(\langle$vn-update$,\ id,\ n\rangle)_u$
Effect
  if $id \in nbrs(u)$ **then** $V(id) \leftarrow n$

Output vcast$(\langle$target-update$,\ target\rangle)_u$
Precondition
  $clock$ **mod** $\delta = e + 2d + 2\epsilon \wedge M \neq \emptyset$
  $target =$ calctarget(assign$(id(M),\ V),\ M)$
Effect
  $M,\ V \leftarrow \emptyset$

Fig. 3. $VN(k, \rho_1, \rho_2)_u$ TIOA, with parameters: safety k, and damping ρ_1, ρ_2.

of VN_u that are on the curve Γ and $y_u(g)$, denote the number $num(V_u(g))$ of CNs assigned to VN_g, where g is either u or a neighbor of u. If $q_u \neq 0$, meaning VN_u is on the curve then we let $lower_u$ denote the subset of $nbrs(u)$ that are on the curve and have fewer assigned CNs than VN_u has after normalizing with $\frac{q_g}{q_u}$. For each $g \in lower_u$, VN_u reassigns the smaller of the following two quantities of CNs to

```
function assign(assignedM: 2^P, y: nbrs^+(u) → ℕ) =
2   assign: P → U, initially {⟨i, u⟩} for each i ∈ assignedM
  n: ℕ, initially y(u); ra: ℕ, //initially 0
4   if y(u) > k then
    if q_u ≠ 0 then
6       let lower = {g ∈ nbrs(u): (q_g/q_u) y(u) > y(g)}
      for each g ∈ lower
8         ra ← min(⌊ρ_2 · [(q_g/q_u) y(u) − y(g)]/2(|lower|+1)⌋, n − k)
        update assign by reassigning ra nodes from u to g
10        n ← n − ra
    else if {v ∈ nbrs(u): q_v ≠ 0} = ∅ then
12      let lower = {g ∈ nbrs(u) : y(u) > y(g)}
      for each g ∈ lower
14        ra ← min(⌊ρ_2 · [y(u) − y(g)]/2(|lower|+1)⌋, n − k)
        update assign by reassigning ra nodes from u to g
16        n ← n − ra
    else  ra ← ⌊ (y(u) -k)/ |{v ∈ nbrs(u): q_v ≠ 0}| ⌋
18      for each g ∈ {v ∈ nbrs(u): q_v ≠ 0}
        update assign by reassigning ra nodes from u to g
20 return assign

22 function calctarget(assign: P → U, locM: P → R) =
  seq, indexed list of pairs in A × P, initially the list,
24   for each i ∈ P : assign(i)= u ∧ locM(i) ∈ Γ_u, of ⟨p, i⟩
  where p= Γ_u^{-1}(locM(i)),  sorted by p, then i
26   for each i ∈ P : assign(i) ≠ null
    if assign(i) = g ≠ u then locM(i) ← o_g
28     else if locM(i) ∉ Γ_u then locM(i) ← choose {min_{x∈Γ_u} {dist(x, locM(i))}}
    else let p = Γ_u^{-1}(locM(i)), seq(k) = ⟨p, i⟩
30       if k = first(seq) then locM(i) ← Γ_u(inf(A_u))
      else if k = last(seq) then locM(i) ← Γ_u(sup(A_u))
32       else let seq(k − 1) = ⟨p_{k−1}, i_{k−1}⟩,
        seq(k + 1) = ⟨p_{k+1}, i_{k+1}⟩
34         locM(i) ← Γ_u(p + ρ_1 · ((p_{k−1}+p_{k+1})/2 − p))
return locM
```

Fig. 4. $VN\ (k, \rho_1, \rho_2)_u$ TIOA functions.

VN_g: (1) $ra = \rho_2 \cdot [\frac{q_g}{q_u} y_u(u) - y_u(g)]/2(|lower_u| + 1)$, where $\rho_2 < 1$ is a *damping factor*, and (2) the remaining number of CNs over k still assigned to VN_u.

If $q_u = 0$, meaning VN_u is not on the curve, and VN_u has no neighbors on the curve (lines 11–15), then we let $lower_u$ denote the subset of $nbrs(u)$ with fewer assigned CNs than VN_u. For each $g \in lower_u$, VN_u reassigns the smaller of the following two quantities of CNs: (1) $ra = \rho_2 \cdot [y_u(u) - y_u(g)]/2(|lower_u| + 1)$ and (2) the remaining number of CNs over k still assigned to VN_u. VN_u is on a *boundary* if $q_u = 0$, but there is a $g \in nbrs(u)$ with $q_g \neq 0$. In this case, $y_u(u) - k$ of VN_u's CNs are assigned equally to neighbors in In_u (lines 17–19).

The calctarget function assigns to every CN_p assigned to VN_u a target point $locM_u(p)$ in region R_g, where $g = u$ or it is one of u's neighbors. The target point $locM_u(p)$ is computed as follows: If CN_p is assigned to VN_g, $g \neq u$, then its target is set to the center $\mathbf{o}_g$ of region g (line 27); if CN_p is assigned to VN_u but is not located on the curve Γ_u then its target is set to the nearest point on the curve, nondeterministically choosing one if there are several (line 28); if CN_p is either the first or last client node on Γ_u then its target is set to the corresponding endpoint of Γ_u (lines 30–31); if CN_p is on the curve but is not the first or last client node then its target is moved to the mid-point of the locations of the preceding and succeeding CNs on the curve (line 34). For the last two computations a sequence *seq* of nodes on the curve sorted by curve location is used (line 25).

Lastly, VN_u broadcasts new waypoints for the round via a target-update message to its CNs.

5.5 Complete System

The complete algorithm, MC, is the instantiation of each component in Figure 1 with fail-transformed CN and VN algorithms. Formally, it is the parallel composition of the following TIOAs: (a) RW, (b) VW, (c) $VBcast$, (d) $Fail(VBDelay_p \| CN_p)$, one for each $p \in P$, and (e) $Fail(VBDelay_u \| VN_u)$. Recall that $Fail(\mathcal{A})$ denotes the fail-transformed version of TIOA $\mathcal{A}$.

Round length. Given the maximum Euclidean distance, r, between points in neighboring regions, it can take up to $\frac{r}{v_{max}}$ time for a client to reach its target. Also, after the client arrives in the region it was assigned to, it could find the local VN has failed. Let d_r be the time it takes a VN to startup, once a new node enters the region. To ensure a round is long enough for a client node to send the cn-update, allow VNs to exchange information, allow clients to receive a target-update message and arrive at new assigned target locations, and be sure virtual nodes are alive in their region before a new round begins, we require that δ, the CN parameter, satisfy $\delta > 2e + 3d + 2\epsilon + r/v_{max} + d_r$.

6. CORRECTNESS OF ALGORITHM

In this section, we show that *starting from an initial state* the system described in Section 5.2, satisfies the requirements specified in Section 5.1. In the following section we show self-stabilization. The proofs of the results in this section parallel those presented in [Lynch et al. 2005], albeit the semantics of the Virtual Layers used here is different. Here we describe the key ideas; some of the detailed proofs appear in the Appendix.

We define round t as the interval of time $[\delta(t-1), \delta \cdot t)$. That is, round t begins at time $\delta(t-1)$ and is completed by time $\delta \cdot t$. We say $CN_p, p \in P$, is *active* in round t if node p is not failed throughout round t. A VN_u, $u \in U$, is *active* in round t if there is some active CN_p such that $region(x_p) = u$ for the duration of rounds $t-1$ and t. Thus, by definition, none of the VN s is active in the first round. We also define the following notation:

- $In(t) \subseteq U$ is the subset of VN ids that are is active in round t and $q_u \neq 0$;
- $Out(t) \subseteq U$ is the subset of VNs that are active in round t and $q_u = 0$;
- $C(t) \subseteq P$ is the subset of active CNs at round t;
- $C_{in}(t) \subseteq P$ is the set of active CNs located in regions with id in $In(t)$ at the beginning of round t;
- $C_{out}(t) \subseteq P$ is subset of active CNs located in regions with id in $Out(t)$ at the beginning of round t.

For every pair of regions u, w and for every round t, we define $y(w,t)_u$ to be the value of $V(w)_u$ (i.e., the number of clients u believes are available in region w)

immediately prior to VN_u performing a vcast_u in round t, i.e., at time $e + 2d + 2\epsilon$ after the beginning of round t. If there are no new client failures or recoveries in round t, then for every pair of regions $u, w \in nbrs^+(v)$, we can conclude that $y(v,t)_u = y(v,t)_w$, which we denote simply as $y(v,t)$. We define $\rho_3 \triangleq \frac{q_{max}^2}{(1-\rho_2)\sigma}$. The rate ρ_3 effects the rate of convergence, and will be used in the analysis. Notice that $\rho_3 > 1$. Notice that for any $v, w \in nbrs(u) \cup \{u\}$, in the absence of failures and recoveries of CNs in round t, $y_{v,t} = y_{w,t}$; we write this simply as $y_h(t)$.

6.1 Approximately Proportional Distribution

For the rest of this section we fix a particular round number t_0 and assume that, for all $p \in P$, no fail_p or $\mathsf{recover}_p$ events occur at or after round t_0. The first lemma states some basic facts about the assign function.

Lemma 6.1. *In every round $t \geq t_0$: (1) If $y(u,t) \geq k$ for some $u \in U$, then $y(u,t+1) \geq k$; (2) $In(t) \subseteq In(t+1)$; (3) $Out(t) \subseteq Out(t+1)$.*

Proof. We fix round $t \geq t_0$.

(1) From line 4 of the assign function (Figure 4) it is clear that VN_u, $u \in U$, reassigns some of its CN s in round t only if $y(u,t) > k$. And if a CN is not reassigned and does not fail, it remains active in the same region.
(2) For any VN_u, $u \in In(t)$, if $y(u,t) < k$ then VN_u does not reassign CN s, and $y(u,t+1) = y(u,t)$. Otherwise, from line 8 of Figure 4 it follows that $y(u,t+1) \geq k$. In both cases $u \in In(t+1)$.
(3) For any VN_u, $u \in Out(t)$, if $y(u,t) < k$ then VN_u does not reassign CN s, and $y(u,t+1) = y(u,t)$. Otherwise, from line 14 and line 17 of Figure 4 it follows that $y(u,t+1) \geq k$. In both cases $u \in Out(t+1)$.

□

We now identify a round $t_1 \geq t_0$ after which the set of regions $In(t)$ and $Out(t)$ remain fixed.

Lemma 6.2. *There exists a round $t_1 \geq t_0$ such that for every round $t \in [t_1, t_1 + (1+\rho_3)m^2n^2]$: (1) $In(t) = In(t_1)$; (2) $Out(t) = Out(t_1)$; (3) $C_{in}(t) \subseteq C_{in}(t+1)$; and (4) $C_{out}(t+1) \subseteq C_{out}(t)$.*

Proof. By Lemma 6.1, Part 2, we know that the set $In(t) \subseteq U$ is non-decreasing as t increases. From Part 3, we know that set $Out(t) \subseteq U$ is non-decreasing as t increase. Since U is finite, we conclude from this that there is some round t_1 after which no new regions $u \in U$ are added to either $In(t)$ or $Out(t)$. Thus we have satisfied Parts 1 and 2. Notice that this occurs no later than round $t_0 + 2m^2 \cdot (1 + \rho_3)m^2n^2$.

For Part 3, consider a client CN_p, $p \in C_{in}(t)$, that is currently assigned in round t to VN_u, $u \in In(t)$. From lines 5–9 of Figure 4 we see that CN_p is assigned to some VN_w, $w \in nbrs^+(u)$ where $q_w \neq 0$. If VN_w is inactive in round $t+1$, then client CN_p remains in VN_w until it becomes active, resulting in VN_w being added to $In(t)$, thus contradicting the fact that for every round $t' \geq t_1$, $In(t') = In(t_1)$. We conclude that VN_w is active in round t, and hence round $t+1$, from which the claim follows.

For Part 4, notice that since there are no failures and recoveries of CN s, $C(t) = C(t+1)$. By definition, $C_{in}(t) \cup C_{out}(t) = C(t)$, $C_{in}(t) \cap C_{out}(t) = \emptyset$, and $C_{in}(t+1) \cup C_{out}(t+1) = C(t+1)$, $C_{in}(t+1) \cap C_{out}(t+1) = \emptyset$. The result follows from Part (3). □

Fix t_1 for the rest of this section such that it satisfies Lemma 6.2. The next lemma states that eventually, regions bordering on the curve stop assigning clients to regions that are on the curve. That is, assume that u is a region where $q_u = 0$, but that u has a neighbor v where $q_v \neq 0$; then, eventually, from some round onwards, u never again assigns clients to v.

Lemma 6.3. *There exists some round $t_2 \in [t_1, t_1 + (1+\rho_3)m^2n^2]$ such that for every round $t \in [t_2, t_2 + (1+\rho_3)m^2n]$: if $u \in Out(t)$ and $v \in In(t)$ and if u and v are neighboring regions, then u does not assign any clients to v in round t.*

Proof. Notice that if u assigns a client to v, then C_{out} decreases by one. During the interval $[t_1, t_1 + (1+\rho_3)m^2n^2]$, we know that C_{out} is non-increasing by Lemma 6.2. Thus, eventually, there is some round t_2 after which either $C_{out} = \emptyset$ or after which no further clients are assigned from a region $Out(\cdot)$ to a region $In(\cdot)$. Since there are at most n clients, we can conclude that this occurs at latest by round $t_1 + n \cdot [(1+\rho_3)m^2n]$. □

Fix t_2 for the rest of this section such that it satisfies Lemma 6.3. Lemma 6.2 implies that in every round $t \geq t_1$, $In(t) = In(t_1)$ and $Out(t) = Out(t_1)$; we denote these simply as In and Out. The next lemma states a key property of the assign function after round t_1. For a round $t \geq t_1$, consider some VN_u, $u \in Out(t)$, and assume that VN_w is the neighbor of VN_u assigned the most clients in round t. Then we can conclude that VN_u is assigned no more clients in round $t+1$ than VN_w is assigned in round t. A similar claim holds for regions in $In(t)$, but in this case with respect to the *density* of clients with respect to the quantized length of the curve. The proof of this lemma (see the Appendix) is based on careful analysis of the behavior of the assign function.

Lemma 6.4. *In every round $t \in [t_2, t_2 + (1+\rho_3)m^2n]$, for $u, v \in U$ and $u \in nbrs(v)$:*

(1) *If $u, v \in Out(t)$ and $y(v,t) = \max_{w \in nbrs(u) \cap Out(t)} y(w,t)$ and $y(u,t) < y(v,t)$, then $y(u,t+1) < y(v,t)$.*

(2) *If $u, v \in In(t)$ and $y(v,t)/q_v = \max_{w \in nbrs(u) \cap In(t)} [y(w,t)/q_w]$ and $y(u,t)/q_u < y(v,t)/q_v$, then:*

$$\frac{y(u,t+1)}{q_u} \leq \frac{y(v,t)}{q_v} - (1-\rho_2)\frac{\sigma}{q_{max}^2} .$$

The next lemma states that there exists a round T_{out} such that in every round $t \geq T_{out}$, the set of CNs assigned to region $u \in Out(t)$ does not change.

Lemma 6.5. *There exists a round $T_{out} \in [t_2, t_2 + m^2n$ such that in any round $t \geq T_{out}$, the set of CNs assigned to VN_u, $u \in Out(t)$, is unchanged.*

Proof. First, we show that there exists some round T_{out} such that the aggregate number of CN s assigned to VN_u remains the same in both T_{out} and $T_{out}+1$ for all $u \in Out(t_2)$. We then show that the actual assignment of individual clients remains the same in T_{out} and $T_{out} + 1$.

We consider a vector $E(t)$ that represents the distribution of clients among regions in $Out(t)$. That is, the first element in $E(t)$ represents the largest number of clients in any region; the second element in $E(t)$ represents the second largest number of clients in any region; and so forth. We then argue that, compared lexicographically, $E(t+1) \leq E(t)$. Since the elements in $E(t)$ are integers, we conclude from this that eventually the distribution of clients becomes stables and ceases to change.

We proceed to define $E(t)$ as follows for $t \geq t_2$. Let $N_{out} = |Out|$. Let $\Pi(t)$ be a permutation of Out that orders the regions by the number of assigned clients, i.e., if u precedes v in $\Pi(t)$, then $y(u,t) \leq y(v,t)$. When we say that some region u has index k, we mean that $\Pi(t)_k = u$. Define $E(t)$ as follows:

$$E(t) = \langle y(\Pi(t)_{N_{out}}, t), y(\Pi(t)_{N_{out}-1}, t), \ldots, y(\Pi(t)_1, t) \rangle \ .$$

We use the notation $E(t)_\ell$ to refer to the ℓ^{th} component of $E(t)$ counting from the right, i.e., it refers to $\Pi(t)_\ell$. Any two vectors $E(t)$ and $E(t+1)$ can be compared lexicographically, examining each of the elements in turn from left to right, i.e., largest to smallest.

We now consider some round $t \in [t_2, t_2 + m^2 n]$, and show that $E(t) \geq E(t+1)$. Consider the case where $E(t) \neq E(t+1)$, and let u be the region with maximum index that assigns clients to another region. Let k be the index of region u.

First, we argue that for every region v with index $\leq k$, we can conclude that $y(v, t+1) < y(u,t)$. Consider some particular region v. Notice that v has no neighbors in Out that are assigned more than $y(u,t)$ clients in round t; otherwise, such a neighbor would assign clients to v, contradicting our choice of u. Thus, by Lemma 6.4, Part 1, we can conclude that $y(v, t+1) < y(u,t)$ (as long as $t \in [t_2, t_2 + 2m^2 n]$, which we will see to be sufficient).

Since this implies that there are at least k regions assigned fewer than $y(u,t) = E(t)_k$ clients in round $t{+}1$, we can conclude that $E(t{+}1)_k < E(t)_k$. In order to show that $E(t+1) < E(t)$, it remains to show that for every $k' > k$, $E(t)_{k'} = E(t+1)_{k'}$.

Consider some region v with index $> k$. By our choice of u, it is clear that v is not assigned any clients by a region with index $> k$. It is also easy to see that v is not assigned any clients by a region w with index $\leq k$, since $y(v,t) \geq y(u,t) \geq y(w,t)$; as per line 12, region w does not assign any clients to a region with $\geq y(w,t)$ clients. Thus no new clients are assigned to region v. Moreover, by choice of u, region v assigns none of its clients elsewhere. Finally, since $t \geq t_0$, none of the clients fail. Thus, $y(v,t) = y(v, t+1)$.

Since the preceding logic holds for all $N_{out} - k + 1$ regions with index $> k$, and all have more than $y(u,t) > y(u, t+1)$ clients, we conclude that for every $k' > k$, $E(t)_{k'} = E(t+1)_{k'}$, implying that $E(t) > E(t+1)$, as desired.

Since $E(\cdot)$ is non-increasing, and since it is bounded from below by the zero vector, we conclude that eventually there is a round T_{out} such that for all $t \geq T_{out}$, $E(t) = E(t+1)$.

Now suppose the set of clients assigned to region u changes in some round $t \geq T_{out}$. The only way the set of clients assigned to region u could change, without changing $y(u,t)$ and the set C_{out}, is if there existed a cyclic sequence of VN s with ids in Out in which each VN gives up $c > 0$ CN s to its successor VN in the sequence, and receives c CN s from its predecessor. However, such a cycle of VN s cannot exist because the *lower* set imposes a strict partial ordering on the VN s.

Finally, we observe that if $E(t) = E(t+1)$ for any t, then the assignment of clients does not change from that point onwards: since all the clients remained in the same regions in $E(t)$ and $E(t+1)$, we can conclude that the *assign* function produced the same assignment in $E(t+1)$ as in $E(t)$. Since the vector $E(\cdot)$ has at most m^2 elements, each with at most n values, we can conclude that T_{out} is at most m^2n rounds after t_2. □

For the rest of the section we fix T_{out} to be the first round after t_0, at which the property stated by Lemma 6.5 holds. Lemma 6.5, together with Lemmas 6.1, 6.2, and 6.3, imply that in every round $t \geq T_{out}$, $C_{In}(t) = C_{In}(t_1)$ and $C_{Out}(t) = C_{Out}(t_1)$; we denote these simply as C_{In} and C_{Out}. The next lemma states a property similar to that of Lemma 6.5 for VN_u, $u \in In$, and the argument is similar to the proof of Lemma 6.5, and uses Part (2) of Lemma 6.4.

Lemma 6.6. *There exists a round $T_{stab} \in [T_{out}, T_{out} + \rho_3 m^2 n]$ such that in every round $t \geq T_{stab}$, the set of CNs assigned to VN_u, $u \in In$, is unchanged.*

Proof. We proceed to define $E(t)$ as follows for $t \geq T_{out}$. Let $N_{in} = |In|$. Let $\Pi(t)$ be a permutation of In that orders the regions by the density of assigned clients, i.e., if u precedes v in $\Pi(t)$, then $y(u,t)/q_u \leq y(v,t)/q_v$. When we say that some region u has index k, we mean that $\Pi(t)_k = u$. Define $E(t)$ as follows:

$$E(t) = \left\langle \frac{y(\Pi(t)_{N_{in}}, t)}{q_{\Pi(t)_{N_{in}}}}, \frac{y(\Pi(t)_{N_{in}-1}, t)}{q_{\Pi(t)_{N_{in}-1}}}, \ldots, \frac{y(\Pi(t)_1, t)}{q_{\Pi(t)_1}} \right\rangle .$$

We use the notation $E(t)_\ell$ to refer to the ℓ^{th} component of $E(t)$ counting from the right, i.e., it refers to $\Pi(t)_\ell$. Any two vectors $E(t)$ and $E(t+1)$ can be compared lexicographically, examining each of the elements in turn from left to right, i.e., largest to smallest.

We now consider some round $t \geq T_{out}$, and show that $E(t) \geq E(t+1)$. Consider the case where $E(t) \neq E(t+1)$, and let u be the region with maximum index that assigns clients to another region. Let k be the index of region u.

First, we argue that for every region v with index $\leq k$, we can conclude that $y(v,t+1)/q_v \leq y(u,t)/q_u - \zeta$ for some constant ζ. Consider some particular region v. Notice that v has no neighbors in In that have density greater than $y(u,t)/q_u$ in round t; otherwise, such a neighbor would assign clients to v, contradicting our choice of u. Thus, by Lemma 6.4, Part 2, we can conclude that $y(v,t+1)/q_v \leq y(u,t)/q_u - \zeta$ where $\zeta = (1-\rho_2)\frac{\sigma}{q_{max}^2}$ (as long as $t \in [t_2, t_2 + (1+\rho_3)m^2n]$, which we will see to be sufficient).

Since this implies that there are at least k regions assigned fewer than $y(u,t) = E(t)_k$ clients in round $t+1$, we can conclude that $E(t+1)_k \leq E(t)_k - \zeta$. In order to show that $E(t+1) < E(t)$, it remains to show that for every $k' > k$, $E(t)_{k'} = E(t+1)_{k'}$.

Consider some region v with index $> k$. By our choice of u, it is clear that v is not assigned any clients by a region with index $> k$. It is also easy to see that v is not assigned any clients by a region w with index $\leq k$, since $y(v,t)/q_v \geq y(u,t)/q_u \geq y(w,t)/q_w$; as per line 6, region w does not assign any clients to a region with a density $\geq y(w,t)/q_w$. Thus no new clients are assigned to region v. Moreover, by choice of u, region v assigns none of its clients elsewhere. Finally, since $t \geq t_0$, none of the clients fail. Thus, $y(v,t)/q_v = y(v,t+1)/q_v$.

Since the preceding logic holds for all $N_{in} - k + 1$ regions with index $> k$, and all have more than $y(u,t)/q_u$ clients, we conclude that for every $k' > k$, $E(t)_{k'} = E(t+1)_{k'}$, implying that $E(t) > E(t+1)$, as desired.

Since $E(\cdot)$ is non-increasing, and since it decreases by at least a constant ζ in every round in which it decreases, and since it is bounded from below by the zero vector, we conclude that eventually there is a round T_{stab} such that for all $t \geq T_{stab}$, $E(t) = E(t+1)$.

Now suppose the set of clients assigned to region u changes in some round $t \geq T_{stab}$. The only way the set of clients assigned to region u could change, without changing $y(u,t)/q_u$ and the set C_{in}, is if there existed a cyclic sequence of VN s with ids in In in which each VN gives up $c > 0$ CN s to its successor VN in the sequence, and receives c CN s from its predecessor. However, such a cycle of VN s cannot exist because the *lower* set imposes a strict partial ordering on the VN s.

Finally, we observe that if $E(t) = E(t+1)$ for any t, then the assignment of clients does not change from that point onwards: since all the clients remained in the same regions in $E(t)$ and $E(t+1)$, we can conclude that the *assign* function produced the same assignment in $E(t+1)$ as in $E(t)$. Since the vector $E(\cdot)$ has at most m^2 elements, each with at most $n\frac{q_{max}^2}{(1-\rho)\sigma}$ values, we can conclude that T_{stab} is at most $\rho_3 m^2 n$ rounds after T_{out}, and hence at most $(1+\rho_3)m^2 n$ rounds after t_2, as needed. □

For the rest of the section we fix T_{stab} to be the first round after T_{out}, at which the property stated by Lemma 6.6 holds. The next lemma states that the number of clients assigned to each VN_u, $u \in In$, in the stable assignment after T_{stab} is proportional to q_u within a constant additive term. The proof follows by induction on the number of hops from between any pair of VNs.

Lemma 6.7. *In every round $t \geq T_{stab}$, for $u, v \in In(t)$:*

$$\left| \frac{y(u,t)}{q_u} - \frac{y(v,t)}{q_v} \right| \leq \left\lceil \frac{10(2m-1)}{q_{min}\rho_2} \right\rceil.$$

6.2 Uniform Spacing

From line 28 of Figure 4, it follows that by the beginning of round $T_{stab}+2$, all CNs in C_{in} are located on the curve Γ. Thus, the algorithm satisfies our first goal. The next lemma states that the locations of the CNs in each region u, $u \in In$, are uniformly spaced on Γ_u in the limit, and it is proved by analyzing the behavior of calctarget as a discrete time dynamical system.

Lemma 6.8. *Consider a sequence of rounds $t_1 = T_{stab}, \ldots, t_n$. As $n \to \infty$, the locations of CNs in u, $u \in In$, are uniformly spaced on Γ_u.*

Proof. From Lemma 6.6 we know that the set of CN s assigned to each VN_u, $u \in In$, remains unchanged. Then, at the beginning of round t_2, every CN assigned to VN_u is located in region u and is on the curve Γ_u. Assume w.l.o.g. that VN_u is assigned at least two CN s. Then, at the beginning of round t_3, one CN is positioned at each endpoint of Γ_u, namely at $\Gamma_u(inf(P_u))$ and $\Gamma_u(sup(P_u))$. From lines 30–31 of Figure 4, we see that the target points for these *endpoint* CN s are not changed in successive rounds.

Let $seq_u(t_2) = \langle p_0, i_{(0)} \rangle, \ldots, \langle p_{n+1}, i_{(n+1)} \rangle$, where $y_u = n+2$, $p_0 = inf(P_u)$, and $p_{n+1} = sup(P_u)$. From line 34 of Figure 4, for any i, $1 < i < n$, the i^{th} element in seq_u at round t_k, $k > 2$, is given by:

$$p_i(t_{k+1}) = p_i(t_k) + \rho_1 \left(\frac{p_{i-1}(t_k) + p_{i+1}(t_k)}{2} - p_i(t_k) \right).$$

For the endpoints, $p_i(t_{k+1}) = p_i(t_k)$. Let the i^{th} uniformly spaced point on the curve Γ_u between the two endpoints be x_i. The parameter value $\bar{p}_i$ corresponding to x_i is given by $\bar{p}_i = p_0 + \frac{i}{n+1}(p_{n+1} - p_0)$. In what follows, we show that as $n \to \infty$, the p_i converge to $\bar{p}_i$ for every i, $0 < i < n+1$, that is, the location of the non-endpoint CN s are uniformly spaced on Γ_u. The rest of this proof is exactly the same as the proof of Theorem 3 in [Goldenberg et al. 2004] in which the authors prove convergence of points on a straight line with uniform spacing.

Observe that $\bar{p}_i = \frac{1}{2}(\bar{p}_{i-1} + \bar{p}_{i+1}) = (1-\rho_1)\bar{p}_i + \frac{\rho_1}{2}(\bar{p}_{i-1} + \bar{p}_{i+1})$. Define error at step k, $k > 2$, as $e_i(k) = p_i(t_k) - \bar{p}_i$. Therefore, for each i, $2 \le i \le n-1$, $e_i(k+1) = p_i(t_{k+1}) - \bar{p}_i = (1-\rho_1)e_i(k) + \frac{\rho_1}{2}(e_{i-1}(k) + e_{i+1}(k))$, $e_1(k+1) = (1-\rho_1)e_1(k) + \frac{\rho_1}{2}e_2(k)$, and $e_n(k+1) = (1-\rho_1)e_n(k) + \frac{\rho_1}{2}e_{n-1}(k)$. The matrix for this can be written as: $e(k+1) = Te(k)$, where T is an $n \times n$ matrix:

$$\begin{bmatrix} 1-\rho_1 & \rho_1/2 & 0 & 0 & \ldots & 0 \\ \rho_1/2 & 1-\rho_1 & \rho_1/2 & 0 & \ldots & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & \ldots & 0 & \rho_1/2 & 1-\rho_1 & \rho_1/2 \\ 0 & \ldots & 0 & 0 & 1-\rho_1 & \rho_1/2 \end{bmatrix}.$$

Using symmetry of T, $\rho_1 \le 1$, and some standard theorems from control theory, it follows that the largest eigenvalue of T is less than 1. This implies $lim_{k\to\infty} T^k = 0$, which implies $lim_{k\to\infty} e(k) = 0$. □

Thus we conclude by summarizing the results in this section:

Theorem 6.9. *If there are no* fail *or* restart *actions for robots at or after some round t_0, then within a finite number of rounds after t_0:*

(1) *The set of CNs assigned to each VN_u, $u \in U$, becomes fixed, and the size of the set is proportional to the quantized length q_u, within an a constant additive term $\frac{10(2m-1)}{q_{min}\rho_2}$.*

(2) *All client nodes in a region $u \in U$ for which $q_u \ne 0$ are located on Γ_u and uniformly spaced on Γ_u in the limit.*

7. SELF-STABILIZATION OF ALGORITHM

In this section we show that the VSA-based motion coordination scheme is self-stabilizing. Specifically, we show that when the VSA and client components in the VSA layer start out in some *arbitrary state* owing to failures and restarts, they eventually return to a reachable state. Thus, the traces of $VLayer[MC]$ running with some reachable state of $Vbcast \| RW \| VW$, eventually, becomes indistinguishable from a reachable trace of $VLayer[MC]$. Recall Definition 4.1 and note that the virtual layer algorithm alg is instantiated here with the motion coordination algorithm MC of Section 5.

We first show that our motion coordination algorithm $VLNodes[MC]$ is self-stabilizing to some set of legal states L_{MC}. Then, we show that these legal states correspond to reachable states of $VLayer[MC]$; hence, the traces of our motion coordination algorithm, where clients and VSAs start in an arbitrary state, eventually look like reachable traces of the correct motion coordination algorithm.

An *emulation* is a kind of implementation relationship between two sets of TIOAs. A VSA layer emulation algorithm is a mapping that takes a VSA layer algorithm, alg, and produces TIOA programs for an underlying system consisting of emulator physical nodes (corresponding to clients), such that when those programs are run with external oracles such as RW the resulting system has traces that are closely related to the traces of a VSA layer. In particular, the traces restricted to non-broadcast actions at the client nodes are the same.

In [Dolev et al. 2005a; Nolte and Lynch 2007a] we have shown how to implement a self-stabilizing VSA Layer. In particular, that implementation guarantees that (1) each algorithm $alg \in VAlgs$ stabilizes in some t_{Vstab} time to traces of executions of $U(VLNodes[alg]) \| R(RW \| VW \| Vbcast)$, and (2) for any $u \in U$, if there exists a client that has been in region u and alive for d_r time and no alive clients in the region failed or left the region in that time, then VSA V_u is not failed. Thus, if the coordination algorithm MC is such that $VLNodes[MC]$ self-stabilizes in some time t to L_{MC} relative to $R(RW \| VW \| Vbcast)$, then we can conclude that physical node traces of the emulation algorithm on MC stabilize in time $t_{Vstab} + t$ to client traces of executions of the VSA layer started in legal set L_{MC} and that satisfy the above failure-related properties.

7.1 Legal Sets

First we describe two legal sets for $VLayer[MC]$, L^1_{MC} and L_{MC}. The first legal set L^1_{MC} describes a set of states that result after the first GPSupdate occurs at each client node and the first timer occurs at each virtual node.

Definition 7.1. *A state* $\mathbf{x}$ *of* $VLayer[MC]$ *is in* L^1_{MC} *iff the following hold:*

(1) $\mathbf{x} \lceil X_{Vbcast\|RW\|VW} \in Reach_{Vbcast\|RW\|VW}$.

(2) $\forall u \in U : \neg failed_u : clock_u \in \{RW.now, \bot\} \wedge (M_u \neq \emptyset \Rightarrow clock_u \mod \delta \in (0, e + 2d + 2\epsilon])$.

(3) $\forall p \in P : \neg failed_p \Rightarrow \mathbf{v}_p \in \{RW.vel(p)/v_{max}, \bot\}$.

(4) $\forall p \in P : \neg failed_p \wedge x_p \neq \bot$:
  (a) $x_p = RW.loc(p) \wedge clock_p = RW.now$.
  (b) $x^*_p \in \{x_p, \bot\} \vee ||x^*_p - x_p|| < v_{max}(\delta \lceil clock_p/\delta \rceil - clock_p - d_r)$.
  (c) $Vbcast.reg(p) = region(x_p) \vee clock \mod \delta \in (e + 2d + 2\epsilon, \delta - d_r + \epsilon_{sample})$.

Part (1) requires that $\mathbf{x}$ restricted to the state of $Vbcast \| RW \| VW$ to be a reachable state of $Vbcast \| RW \| VW$. Part (2) states that nonfailed VSAs have *clock*s that are either equal to real-time or $\bot$, and have nonempty M only after the beginning of a round and up to $e + 2d + 2\epsilon$ time into a round. Part (3) states that nonfailed clients have velocity vectors that are equal either to $\perp$ or equal to the client's velocity vector in RW, scaled down by v_{max}. Finally, Part (4) states that nonfailed clients with non-$\perp$ positions have: (4a) positions equal to their actual location and local *clock*s equal to the real-time, (4b) targets that are one of $\perp$, the location, or a point reachable from the current location within d_r before the end of the round, and (4c) $Vbcast$ last region updates that match the current region or the time is within a certain time window in a round. It is routine to check that L^1_{MC} is indeed a legal set for *VLayer[MC]*.

Now we describe the main legal set L_{MC} for our algorithm. First we describe a set of *reset* states, states corresponding to states of $VLayer[MC]$ at the start of a round. Then, L_{MC} is defined as the set of states reachable from these reset states.

Definition 7.2. *A state* $\mathbf{x}$ *of* $VLayer[MC]$ *is in* $Reset_{MC}$ *iff:*

(1) $\mathbf{x} \in L^1_{MC}$.
(2) $\forall p \in P : \neg failed_p \Rightarrow$
$[to\_send^-_p = to\_send^+_p = \lambda \wedge (x_p = \perp \vee \ (x^*_p \neq \perp \wedge v_p = 0))]$.
(3) $\forall u \in U : \neg failed_u \Rightarrow to\_send_u = \lambda$.
(4) $\forall \langle m, u, t, P' \rangle \in vbcastq : P' = \emptyset$.
(5) $RW.now \mod \delta = 0 \wedge \forall p \in P : \forall \langle l, t \rangle \in RW.updates(p) : t < RW.now$.

L_{MC} *is the set of reachable states of* $Start(VLayer[MC], Reset_{MC})$.

$Reset_{MC}$ consists of states in which (1) in L^1_{MC}, (2) nonfailed clients have empty queues in its $VBDelay$ and either has a position variable equal to $\perp$ or has both a non-$\perp$ target and 0 velocity, (3) nonfailed VSA's have an empty queue in its $VBDelay$, (4) there are no still-processing messages in $Vbcast$, and (5) the time is the starting time for a round and that no GPSupdates have yet occurred at this time. Once again, it is routine to check that that L_{MC} is a legal set for *VLayer[MC]*.

7.2 Stabilization to L_{MC}

First, we state following the stabilization result. To see this, consider the moment after each client has received a GPSupdate and each virtual node has received a timer, which takes at most ϵ_{sample} time.

Lemma 7.3. $VLNodes[MC]$ *is self-stabilizing to* L^1_{MC} *in time* $t > \epsilon_{sample}$ *relative to the automaton* $R(Vbcast \| RW \| VW)$.

Next we show that starting from a state in L^1_{MC}, we eventually arrive at a state in $Reset_{MC}$, and hence, a state in L_{MC}.

Lemma 7.4. *Executions of* $VLayer[MC]$ *started in states in* L^1_{MC} *stabilize in time* $\delta + d + e$ *to executions started in states in* L_{MC}.

PROOF. It suffices to show that for any length-$\delta + d + e$ prefix α of an execution fragment of $VLayer[MC]$ starting from L^1_{MC}, $\alpha.lstate \in L_{MC}$. By the definition of L_{MC}, it suffices to show that there is at least one state in $Reset_{MC}$ that occurs in α.

Let t_0 be equal to $\alpha.fstate(RW.now)$, the time of the first state in α. We consider all the "bad" messages that are about to be delivered after $\alpha.fstate$. (1) There may be messages in $Vbcast.vbcastq$ that can take up to d time to be dropped or delivered at each process. (2) There may be messages in to_send^- or to_send^+ queues at clients that can submitted to $Vbcast$ and take up to d time to be dropped or delivered at each process. And (3), there may be messages in to_send queues at VSAs that can take up to e time to be submitted to $Vbcast$ and an additional d time to be dropped or delivered at each process. We know that all "bad" messages will be processed (dropped or delivered at each process) by some state $\mathbf{x}$ in α such that $x(RW.now) = t_1 = t_0 + d + e$.

Consider the state $\mathbf{x}^*$ at the start of the first round after state $\mathbf{x}$. Since $\mathbf{x}^*(RW.now) = \delta(\lfloor t_1/\delta \rfloor + 1)$, we have that $\mathbf{x}^*(RW.now) - t_0 = \mathbf{x}^*(RW.now) - t_1 + e + d \leq \delta + e + d$. The only thing remaining to show is that $\mathbf{x}^*$ is in $Reset_{MC}$. It's obvious that $\mathbf{x}^*$ satisfies (1) and (5) of Definition 7.2. Code inspection tells us that for any state in L^1_{MC}, and hence, for any state in α, any new vcast transmissions of messages will fall into one of three categories:

(1) Transmission of cn-update by a client at a time t such that $t \mod \delta = 0$. Such a message is delivered by time $t + d$.
(2) Transmission of vn-update by a virtual node at a time t such that $t \mod \delta = d + \epsilon$. Such a message is delivered by time $t + d + e$.
(3) Transmission of target-update by a virtual node at a time t such that $t \mod \delta = 2d + e + 2\epsilon$. Such a message is delivered by time $t + d + e$.

In each of these cases, any vcast transmission is processed before the start of the next round. Thus, $\mathbf{x}^*$ satisfies properties (2), (3), and (4) of Definition 7.2. To check (2), we just need to verify that for all nonfailed clients if x_p is not $\perp$ then x^*_p is not $\perp$ and v_p is 0. It suffices to show that at least one GPSupdate occurs at each client between state $\mathbf{x}$ and state $\mathbf{x}^*$. (Such an update at a nonfailed client would update x^*_p to be x_p for clients with $x^*_p = \perp$ or x^*_p too far away from x_p to arrive at x^*_p before $\mathbf{x}^*$. Any subsequent receipts of target-update messages will only result in an update to x^*_p if the client will be able to arrive at x^*_p before $\mathbf{x}^*$. This implies that v_p can only be $\perp$ or 0, and since no GPSupdates could have occurred at the same time as $\mathbf{x}^*$, stopping conditions ensure that $v_p \neq \perp$.)

To see that at least one GPSupdate occurs at each client between state $\mathbf{x}'$ and state $\mathbf{x}^*$, we need that $\mathbf{x}^*(RW.now) - \mathbf{x}'(RW.now) > \epsilon_{sample}$. Since $\mathbf{x}^*(RW.now) - \mathbf{x}'(RW.now) = \delta - (\mathbf{x}'(RW.now) \mod \delta) \geq \delta - e - 2d - 2\epsilon$, $\delta > e + 2d + 2\epsilon + d_r$, and $d_r > \epsilon_{sample}$ it follows that $\delta > e + 2d + 2\epsilon + \epsilon_{sample}$. □

Combining our stabilization results we conclude that $VLNodes[MC]$ started in an arbitrary state and run with $R(Vbcast\|RW\|VW)$ stabilizes to L_{MC} in time $\delta + d + e + \epsilon_{sample}$. From transitivity of stabilization and 7.4, the next result follows.

Theorem 7.5. *$VLNodes[MC]$ is self-stabilizing to L_{MC} in time $\delta + d + e + \epsilon_{sample}$ relative to $R(Vbcast\|RW\|VW)$.*

7.3 Relationship between L_{MC} and reachable states

In the previous section we showed that $VLNodes[MC]$ is self-stabilizing to L_{MC} relative to $R(Vbcast\|RW\|VW)$. In order to conclude anything about the traces of $VLayer[MC]$ after stabilization, however, we need to show that traces of $VLayer[MC]$

starting in a state in L_{MC} are reachable traces of $VLayer[MC]$. This is accomplished by first defining a simulation relation $\mathcal{R}_{\mathcal{MC}}$ on the states of $VLayer[MC]$, and then proving that for each state $\mathbf{x} \in L_{MC}$, there exists a state $\mathbf{y} \in \mathsf{Reach}_{VLayer[MC]}$ such that $\mathbf{x}$ and $\mathbf{y}$ are related by $\mathcal{R}_{\mathcal{MC}}$. This implies that the trace of any execution fragment starting with $\mathbf{x}$ is the trace of an execution fragment starting with $\mathbf{y}$, which is a reachable trace of $VLayer[MC]$. We define the candidate relation $\mathcal{R}_{\mathcal{MC}}$ and prove that it is indeed a simulation relation.

Definition 7.6. *$\mathcal{R}_{MC}$ is a relation between states of $VLayer[MC]$ such for any states $\mathbf{x}$ and $\mathbf{y}$ of $VLayer[MC]$, $\mathbf{x}\mathcal{R}_{MC}\mathbf{y}$ iff the following conditions are satisfied:*

(1) $\mathbf{x}(RW.now) = \mathbf{y}(RW.now) \wedge \mathbf{x}(RW.loc) = \mathbf{y}(RW.loc)$.

(2) *For all* $p \in P$, $\mathbf{y}(vel(p)) \in \{\mathbf{x}(vel(p)), \perp\} \wedge$
$\{t \in \mathbb{R}^{\geq 0} \mid \exists l \in R : \langle l, t\rangle \in \mathbf{x}(RW.updates(p))\}$
$= \{t \in \mathbb{R}^{\geq 0} \mid \exists l \in R : \langle l, t\rangle \in \mathbf{y}(RW.updates(p))\}$.

(3) $\mathbf{x}(VW) = \mathbf{y}(VW) \wedge \mathbf{x}(Vbcast.now) = \mathbf{y}(Vbcast.now)$.

(4) $\mathbf{x}(Vbcast.reg) = \mathbf{y}(Vbcast.reg) \wedge$
$\{\langle m, u, t, P'\rangle \in \mathbf{x}(Vbcast.vbcastq) \mid P' \neq \emptyset\}$
$= \{\langle m, u, t, P'\rangle \in \mathbf{y}(Vbcast.vbcastq) \mid P' \neq \emptyset\}$.

(5) *For all* $i \in P \cup U$, $\mathbf{x}(failed_i) = \mathbf{y}(failed_i)$.

(6) *For all* $u \in U : \neg\mathbf{x}(failed_u)$*:*
   (a) $\mathbf{x}(clock_u) = \mathbf{y}(clock_u) \wedge \mathbf{x}(M_u) = \mathbf{y}(M_u)$
   $\wedge\, [\mathbf{x}(M_u) \neq \emptyset \Rightarrow \forall v \in nbrs^{+}(u) : \mathbf{x}(V_u(v)) = \mathbf{y}(V_u(v))]$.
   (b) $|\mathbf{x}(to\_send_u)| = |\mathbf{y}(to\_send_u)| \wedge \forall i \in [1, |\mathbf{x}(to\_send_u)|] : \forall\langle m, t\rangle = \mathbf{x}(to\_send_u[i]) :$
   $\mathbf{y}(to\_send_u[i]) = \langle m, t + \mathbf{y}(rtimer_u) - \mathbf{x}(rtimer_u)\rangle$.

(7) *For all* $p \in P : \neg\mathbf{x}(failed_p)$*:*
   (a) $\mathbf{x}(CN_p) = \mathbf{y}(CN_p) \vee [\mathbf{x}(\mathbf{x}_p) = \mathbf{y}(\mathbf{x}_p) = \perp \wedge \mathbf{x}(v_p) = \mathbf{y}(v_p)]$.
   (b) $\mathbf{x}(VBDelay_p) = \mathbf{y}(VBDelay_p)$.
   (c) $\mathbf{x}(to\_send_p^{-}) \neq \lambda \Rightarrow \mathbf{x}(Vbcast.oldreg(p)) = \mathbf{y}(Vbcast.oldreg(p))$.

We describe the various conditions two related states $\mathbf{x}$ and $\mathbf{y}$ must satisfy. Part (1) requires that they share the same real-time and locations for *CN*s. Part (2) requires that for each client, the velocity at RW is equal or the velocity in $\mathbf{y}$ is $\perp$, and GPSupdate records in the two states are for the same times. Part (3) requires that VW's state and $Vbcast.now$ are the same in $\mathbf{x}$ and $\mathbf{y}$. Part (4) requires that the unprocessed message tuples are the same and that the last recorded regions in $Vbcast$ for clients are the same in both states. Part (5) says that failure status of each *CN* and *VN* is the same in both states. Part (6a) requires that for a nonfailed VSA, local time and the set M are equal in $\mathbf{x}$ and $\mathbf{y}$, and further, if M is nonempty then V is equal for local regions in both states. Part (6b) says that the *to_send* queues for a nonfailed VSA are the same, except with the timestamps for messages in $\mathbf{y}$ adjusted up by the difference between $rtimer_u$ in state $\mathbf{y}$ and $\mathbf{x}$. Part (7a) requires that the algorithm state of a nonfailed *CN* is either the same, or both states share the same local v and have locations equal to $\perp$. Part (7b) says that the $VBDelay$ state is the same for each nonfailed *CN* in $\mathbf{x}$ and $\mathbf{y}$. Finally, Part (7b) requires that if the $to\_send_p^{-}$ buffer is nonempty in state $\mathbf{x}$ for a nonfailed client, then $Vbcast.oldreg(p)$ is the same in both states.

The proof of the following lemma is also routine and it breaks-down into a large case analysis. Say that $\mathbf{x}$ and $\mathbf{y}$ are states in $Q_{VLayer[MC]}$ such that $\mathbf{x}\mathcal{R}_{MC}\mathbf{y}$. For any action or closed trajectory σ of $VLayer[MC]$, suppose $\mathbf{x}'$ is the state reached from $\mathbf{x}$, then, we have to show there exists a closed execution fragment β of $VLayer[MC]$ with $\beta.fstate = \mathbf{y}$, $trace(\beta) = trace(\sigma)$, and $\mathbf{x}'\mathcal{R}_{MC}\beta.lstate$.

Lemma 7.7. *$\mathcal{R}_{\mathcal{MC}}$ is a simulation relation for $VLayer[MC]$.*

To show that each state in L_{MC} is related to a reachable state of $VLayer[MC]$, it is enough to show that each state in $Reset_{MC}$ is related to a reachable state of $VLayer[MC]$. The proof proceeds by providing a construction of an execution of $VLayer[MC]$ for each state in L_{MC}.

Lemma 7.8. *For each state $\mathbf{x} \in Reset_{MC}$, there exists a state $\mathbf{y} \in Reach_{VLayer[MC]}$ such that $\mathbf{x}\mathcal{R}_{MC}\mathbf{y}$.*

PROOF. Let $\mathbf{x}$ be a state in $Reset_{MC}$. We construct an execution α based on state $\mathbf{x}$ such that $\mathbf{x}\mathcal{R}_{MC}\alpha.lstate$. The construction of α is in three phases. Each phase is constructed by modifying the execution constructed in the prior phase to produce a new valid execution of *VLayer[MC]*. After Phase 1, the final state of the constructed execution shares client locations and real-time values with state $\mathbf{x}$. Phase 2 adds client restarts and velocity actions for nonfailed clients in state $\mathbf{x}$, making the final state of clients consistent with state $\mathbf{x}$. Phase 3 adds VSA restart actions to make the final state of VSAs consistent with state $\mathbf{x}$.

1. Let α_1 be an execution of *VLayer[MC]* where each client and VSA starts out failed, no restart or fail events occur, and $\alpha_1.ltime = \mathbf{x}(RW.now)$. For each failed $p \in P$, there exists some history of movement that never violates a maximum speed of v_{max}, is consistent with stored updates for p, and that lead to the current location of p. We move each failed p in just such a way and add a $\mathsf{GPSupdate}(\langle l, t\rangle)_p$ at time t for each $\langle l, t\rangle \in \mathbf{x}(RW.updates(p))$.
   For each nonfailed $p \in P$ and each state in α_1, we set $RW.loc(p) = \mathbf{x}(RW.loc(p))$ (meaning the client does not move). For each nonfailed $p \in P$, add a $\mathsf{GPSupdate}(\mathbf{x}(RW.loc(p)), t)_p$ action for each t such that $\exists\langle l, t\rangle \in \mathbf{x}(RW.updates(p))$.
   For each $u \in U$, if $\mathbf{x}(last(u)) \neq \bot$ then add a $\mathsf{timer}(t)_u$ output at time t in α_1 for each t in the set $\{t^* \mid t^* = \mathbf{x}(last(u)) \vee (t^* < \mathbf{x}(last(u)) \wedge t^* \mod \epsilon_{sample} = 0)\}$.

   Validity. It is obvious that the resulting execution is a valid execution of *VLayer[MC]*.

   Relation between $\mathbf{x}$ and $\alpha_1.lstate$. They satisfy (1)-(4) of Definition 7.6.
2. In order to construct α_2, we modify α_1 in the following way for each $p \in P$ such that $\neg\mathbf{x}(failed_p)$: If $\mathbf{x}(x_p) \neq \bot$, we add a $\mathsf{restart}_p$ event immediately before and a $\mathsf{velocity}(0)_p$ immediately after the last $\mathsf{GPSupdate}_p$ event in α_1. If $\mathbf{x}(x_p) = \bot$ and $\mathbf{x}(v_p) = 0$, then we add a $\mathsf{restart}_p$ and $\mathsf{velocity}(0)_p$ event immediately after the last $\mathsf{GPSupdate}_p$ event in α_1. If $\mathbf{x}(x_p) = \bot$ and $\mathbf{x}(v_p) = \bot$, then we add a $\mathsf{restart}_p$ event at time $\mathbf{x}(RW.now)$ in α_1.

   Validity. Since restart actions are inputs they are always enabled, and a $\mathsf{velocity}_p$ action is always enabled at client CN_p. Also, there can be no trajectory violations since any alive clients receive their first GPSupdate within ϵ_{sample} time of $\mathbf{x}(RW.now)$ in α_2, meaning that since δ is larger than ϵ_{sample} and $\mathbf{x}(RW.now)$ is a round boundary, there is no time before $\mathbf{x}(RW.now)$ in α_2 where a cn-update should have been sent. It is obvious that this is a valid execution of *VLayer[MC]*.

Relation betweem $\mathbf{x}$ *and* $\alpha_2.lstate$. They satisfy (1)-(4) and (7) of Definition 7.6.

3. To construct α, we modify α_2 in the following way for each $u \in U$ such that $\neg\mathbf{x}(failed_u)$: If $\mathbf{x}(clock_u) = \bot$, we add a $\mathsf{restart}_u$ event after any time_u actions. If $\mathbf{x}(clock_u) \neq \bot$, we add a $\mathsf{restart}_u$ event immediately before the last time_u action.

   Validity. A restart action is always enabled. Also, there can be no trajectory violations since no outputs at a VSA are enabled until its local M is nonempty. Since M is empty, we can conclude that this is a valid execution of *VLayer[MC]*.

   Relation between $\mathbf{x}$ *and* $\alpha.lstate$. $\mathbf{x}\mathcal{R}_{MC}\alpha.lstate$.

We conclude that α is an execution of *VLayer[MC]* such that if we take $\mathbf{y} = \alpha.lstate$, then $\mathbf{y} \in Reach_{VLayer[MC]}$ and $\mathbf{x}\mathcal{R}_{MC}\mathbf{y}$. □

From Lemmas 7.8 and 7.7 it follows that the set of trace fragments of $VLayer[MC]$ corresponding to execution fragments starting from $Reset_{MC}$ is contained in the set of traces of $R(VLayer[MC])$. Bringing our results together we arrive at the main theorem:

Theorem 7.9. *The traces of $VLNodes[MC]$, starting in an arbitrary state and executed with automaton $R(Vbcast\|RW\|VW)$, stabilize in time $\delta + d + e + \epsilon_{sample}$ to reachable traces of $R(VLayer[MC])$.*

Thus, despite starting from an arbitrary configuration of the VSA and client components in the VSA layer, if there are no failures or restart of client nodes (robots) at or after some round t_0, then within a finite number of rounds after t_0, the clients are located on the curve and are uniformly spaced in the limit.

8. CONCLUSION

We have described how we can use the Virtual Stationary Automaton infrastructure to design protocols for coordinating mobile robots. In particular, we presented a protocol by which the participating robots can be uniformly spaced on an arbitrary curve. The VSA layer implementation and the coordination protocol are both self-stabilizing. Thus, each robot can begin in an arbitrary state, in an arbitrary location in the network, and the distribution of the robots will still converge to the specified curve. The proposed coordination protocol uses only local information, and hence, should adapt well to flocking or tracking problems where the target formation is dynamically changing.

REFERENCES

Ando, H., Oasa, Y., Suzuki, I., and Yamashita, M. 1999. Distributed memoryless point convergence algorithm for mobile robots with limited visibility. *IEEE Transactions on Robotics and Automation 15,* 5, 818–828.

Blondel, V., Hendrickx, J., Olshevsky, A., and Tsitsiklis, J. 2005. Convergence in multiagent coordination consensus and flocking. In *Proceedings of the Joint forty-fourth IEEE Conference on Decision and Control and European Control Conference*. 2996–3000.

Brown, M. D. 2007. Air traffic control using virtual stationary automata. M.S. thesis, Massachusetts Institute of Technology.

Chandy, K. M., Mitra, S., and Pilotto, C. 2008. Convergence verification: From shared memory to partially synchronous systems. In *In proceedings of Formal Modeling and Analysis of Timed Systems (FORMATS'08)*. LNCS, vol. 5215. Springer Verlag, 217–231.

Chockler, G., Gilbert, S., and Lynch, N. 2008. Virtual infrastructure for collision-prone wireless networks. In *Proceedings of PODC*. To appear.

Clavaski, S., Chaves, M., Day, R., Nag, P., Williams, A., and Zhang, W. 2003. Vehicle networks: achieving regular formation. In *Proceedings of the American control Conference*.

Cortes, J., , Martinez, S., Karatas, T., and Bullo, F. 2004. Coverage control for mobile sensing networks. *IEEE Transactions on Robotics and Automation 20,* 2, 243–255.

Défago, X. and Konagaya, A. 2002. Circle formation for oblivious anonymous mobile robots with no common sense of orientation. In *Proc. 2nd Int'l Workshop on Principles of Mobile Computing (POMC'02)*. ACM, Toulouse, France, 97–104.

Défago, X. and Souissi, S. 2008. Non-uniform circle formation algorithm for oblivious mobile robots with convergence toward uniformity. *Theor. Comput. Sci. 396,* 1-3, 97–112.

Dolev, S. 2000. *Self-stabilization*. MIT Press, Cambridge, MA, USA.

Dolev, S., Gilbert, S., Lahiani, L., Lynch, N., and Nolte, T. 2005a. Virtual stationary automata for mobile networks. In *Proceedings of OPODIS*.

Dolev, S., Gilbert, S., Lahiani, L., Lynch, N. A., and Nolte, T. A. 2005b. Virtual stationary automata for mobile networks. *Technical Report MIT-LCS-TR-979*.

Dolev, S., Gilbert, S., Lynch, N., Shvartsman, A., and Welch, J. 2003. Geoquorums: Implementing atomic memory in ad hoc networks. In *Distributed algorithms*, F. E. Fich, Ed. Lecture Notes in Computer Science, vol. 2848/2003. 306–320.

Dolev, S., Gilbert, S., Lynch, N. A., Schiller, E., Shvartsman, A. A., and Welch, J. L. 2004. Virtual mobile nodes for mobile ad hoc networks. In *18th International Symposium on Distributed Computing (DISC)*. 230–244.

Dolev, S., Gilbert, S., Lynch, N. A., Shvartsman, A. A., and Welch, J. 2005. Geoquorums: Implementing atomic memory in mobile ad hoc networks. *Distributed Computing*.

Efrima, A. and Peleg, D. 2007. Distributed models and algorithms for mobile robot systems. In *SOFSEM (1)*. Lecture Notes in Computer Science, vol. 4362. Springer, Harrachov, Czech Republic, 70–87.

Fax, J. and Murray, R. 2004. Information flow and cooperative control of vehicle formations. *IEEE Transactions on Automatic Control 49*, 1465–1476.

Flocchini, P., Prencipe, G., Santoro, N., and Widmayer, P. 2001. Pattern formation by autonomous robots without chirality. In *SIROCCO*. 147–162.

Gazi, V. and Passino, K. M. 2003. Stability analysis of swarms. *IEEE Transactions on Automatic Control 48,* 4, 692–697.

Goldenberg, D. K., Lin, J., and Morse, A. S. 2004. Towards mobility as a network control primitive. In *MobiHoc '04: Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing*. ACM Press, 163–174.

Herman, T. 1996. Self-stabilization bibliography: Access guide. *Theoretical Computer Science*.

Jadbabaie, A., Lin, J., and Morse, A. S. 2003. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Transactions on Automatic Control 48,* 6, 988–1001.

Kaynar, D. K., Lynch, N., Segala, R., and Vaandrager, F. 2005. *The Theory of Timed I/O Automata*. Synthesis Lectures on Computer Science. Morgan Claypool. Also available as Technical Report MIT-LCS-TR-917.

Lin, J., Morse, A., and Anderson., B. 2003. Multi-agent rendezvous problem. In *42nd IEEE Conference on Decision and Control*.

Lynch, N., Mitra, S., and Nolte, T. 2005. Motion coordination using virtual nodes. In *Proceedings of 44th IEEE Conference on Decision and Control (CDC05)*. Seville, Spain.

Martinez, S., Cortes, J., and Bullo, F. 2005. On robust rendezvous for mobile autonomous agents. In *IFAC World Congress*. Prague, Czech Republic. To appear.

Nolte, T. and Lynch, N. A. 2007a. Self-stabilization and virtual node layer emulations. In *Proceedings of SSS*. 394–408.

Nolte, T. and Lynch, N. A. 2007b. A virtual node-based tracking algorithm for mobile networks. In *ICDCS*.

Nolte, T. A. 2008. Virtual stationary timed automata for mobile networks. Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA. In Preparation.

Olfati-Saber, R., Fax, J., and Murray, R. 2007. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE 95,* 1 (January), 215–233.

Prencipe, G. 2000. Achievable patterns by an even number of autonomous mobile robots. Tech. Rep. TR-00-11. 17.

Prencipe, G. 2001. Corda: Distributed coordination of a set of autonomous mobile robots. In *ERSADS*. 185–190.

Suzuki, I. and Yamashita, M. 1999. Distributed autonomous mobile robots: Formation of geometric patterns. *SIAM Journal of computing 28,* 4, 1347–1363.

A. PROOF OF CORRECTNESS

Lemma 6.4. In every round $t \in [t_2, t_2 + (1+\rho_3)m^2n]$, for $u, v \in U$ and $u \in nbrs(v)$:

(1) If $u, v \in Out(t)$ and $y(v,t) = \max_{w \in nbrs(u) \cap Out(t)} y(w,t)$ and $y(u,t) < y(v,t)$, then $y(u,t+1) < y(v,t)$.

(2) If $u, v \in In(t)$ and $y(v,t)/q_v = \max_{w \in nbrs(u) \cap In(t)} [y(w,t)/q_w]$ and $y(u,t)/q_u < y(v,t)/q_v$, then:

$$\frac{y(u,t+1)}{q_u} \le \frac{y(v,t)}{q_v} - (1-\rho_2)\frac{\sigma}{q_{max}^2} \ .$$

Proof. For Part 1, fix u, v and t, as in the statement of the lemma. Consider some region w that is a neighbor of u and that assigns clients to u in round $t+1$. Since $q_u = 0$, notice that w assigns clients to u only if the conditions of lines 11–16 in Figure 4 are met. This implies that $w \in Out(t)$, and hence $y(w,t) \le y(v,t)$, by assumption. We can also conclude that $lower_w \ge 1$, as w assigns clients to u only if $u \in lower_w$. Finally, from line 14 of Figure 4, we observe that the number of clients that are assigned to u by w in round t is at most:

$$\frac{\rho_2 [y(w,t) - y(u,t)]}{2(|lower_w(t)| + 1)} \le \frac{\rho_2 [y(v,t) - y(u,t)]}{4}$$

Since u has at most four neighbors, we conclude that it is assigned at most $\rho_2 [y(v,t) - y(u,t)]$ clients. Since $\rho_2 < 1$ and $y(u,t) < y(v,t)$, this implies that:

$$\begin{aligned} y(u,t+1) &\le y(u,t) + \rho_2 [y(v,t) - y(u,t)] \\ &\le \rho_2 \cdot y(v,t) + (1-\rho_2) y(u,t) \\ &< \rho_2 \cdot y(v,t) + (1-\rho_2) y(v,t) \\ &< y(v,t) \ . \end{aligned}$$

For Part 2, as in Part 1, fix u, v and t as in the lemma statement. Recall we have assumed that $y(u,t)/q_u < y(v,t)/q_v$. We begin by showing that, due to the manner in which the curve is quantized, $y(u,t)/q_u \le y(v,t)/q_v - \sigma/q_{max}^2$. Since q_u is defined as $\lceil P_u/\sigma \rceil \sigma$, and since q_v is defined as $\lceil P_v/\sigma \rceil \sigma$, we notice that, by assumption:

$$y(u,t)\left\lceil \frac{P_v}{\sigma} \right\rceil \sigma < y(v,t)\left\lceil \frac{P_u}{\sigma} \right\rceil$$

We divide both sides by σ, and since both sides are integral, we exchange the '$<$' with a '$\leq$':

$$y(u,t)\left\lceil \frac{P_v}{\sigma} \right\rceil \leq y(v,t)\left\lceil \frac{P_u}{\sigma} \right\rceil - 1$$

From this we conclude:

$$\frac{y(u,t)}{\left\lceil \frac{P_u}{\sigma} \right\rceil} \leq \frac{y(v,t)}{\left\lceil \frac{P_v}{\sigma} \right\rceil} - \frac{\sigma^2}{q_u q_v}$$

Dividing everything by σ, and bounding q_u and q_v by q_{max}, we achieve the desired calculation.

Now, consider some region w that is a neighbor of u and that assigns clients to u in round $t+1$. First, notice that $w \notin Out(t)$, since by Lemma 6.3, no clients are assigned from an Out region to an In region after round t_2 (prior to $t_2+(1+\rho_3)m^2n$). Thus, w assigns clients to u only if the conditions of lines 5–10 in Figure 4 are met. This implies that $w \in In(t)$, and hence $y(w,t)/q_w \leq y(v,t)/q_v$, by assumption. We can also conclude that $lower_w \geq 1$, as w assigns clients to u only if $u \in lower_w$. Finally, from line 8 of Figure 4, we observe that the number of clients that are assigned to u by w in round t is at most:

$$\frac{\rho_2\left[\left(\frac{q_u}{q_w}\right)y(w,t) - y(u,t)\right]}{2(|lower_w(t)|+1)} \leq \frac{\rho_2\left[\left(\frac{q_u}{q_v}\right)y(v,t) - y(u,t)\right]}{4}$$

Since u has at most four neighbors, we conclude that it is assigned at most $\rho_2\left[(q_u/q_v)y(v,t) - y(u,t)\right]$ clients. This implies that:

$$\begin{aligned} y(u,t+1) \;&\leq\; y(u,t) + \rho_2\left[\left(\frac{q_u}{q_v}\right)y(v,t) - y(u,t)\right] \\ &\leq\; \rho_2\left(\frac{q_u}{q_v}\right)\cdot y(v,t) + (1-\rho_2)\,y(u,t) \end{aligned}$$

Thus, dividing everything by q_u, and recalling that $y(u,t)/q_u \leq y(v,t)/q_v - \sigma/q_{max}^2$:

$$\begin{aligned} \frac{y(u,t+1)}{q_u} \;&\leq\; \rho_2\left(\frac{y(v,t)}{q_v}\right) + (1-\rho_2)\cdot\left(\frac{y(u,t)}{q_u}\right) \\ &\leq\; \rho_2\left(\frac{y(v,t)}{q_v}\right) + (1-\rho_2)\cdot\left(\frac{y(v,t)}{q_v} - \frac{\sigma}{q_{max}^2}\right) \\ &\leq\; \frac{y(v,t)}{q_v} - (1-\rho_2)\frac{\sigma}{q_{max}^2} \end{aligned}$$

□

Lemma A.1. *In every round $t \geq T_{out}$, $|C_{out}(t)| = O(m^3)$.*

Proof. From Lemma 6.5, the set of CN s assigned to each VN_u, $u \in Out(t)$, is unchanged in every round $t \geq T_{out}$. This implies that in any round $t \geq T_{out}$, the number of CN s assigned by VN_u to any of its neighbors is 0. Therefore, from line 17 of Figure 4, for any boundary VN_v, $(y(v,t)-k)/|In_v| < 1$. Recall that In_v is the (constant) set of neighbors of v with quantized curve length $\neq 0$. Since $|In_v| \leq 4$, $y(v,t) < 4+k$.

From line 14 of Figure 4, for any non-boundary VN_v, $v \in Out(t)$, if v is 1-hop away from a boundary region u, then $\frac{\rho_2(y(v,t)-y(u,t))}{2(|lower_v(t)|+1)} < 1$. Since $|lower_v(t)| \leq 4$, $y(v,t) \leq \frac{10}{\rho_2} + 4 + k$. Inducting on the number of hops, the maximum number of clients assigned to a VN_v, $v \in Out(t)$, at ℓ hops from the boundary is at most $\frac{10\ell}{\rho_2} + k + 4$. Since for any ℓ, $1 \leq \ell \leq 2m-1$, there can be at most m VN s at ℓ-hop distance from the boundary, summing gives $|C_{out}| \leq (k+4)(2m-1)m + \frac{10m^2(2m-1)}{\rho_2} = O(m^3)$. □

Lemma 6.7 In every round $t \geq T_{stab}$, for $u,v \in In(t)$:

$$\left|\frac{y(u,t)}{q_u} - \frac{y(v,t)}{q_v}\right| \leq \left\lceil \frac{10(2m-1)}{q_{min}\rho_2} \right\rceil.$$

Proof. Consider a pair of VN s for neighboring regions u and v, $u,v \in In$. Assume w.l.o.g. $y(u,t) \geq y(v,t)$. From line 8 of Figure 4, it follows that $\rho_2(\frac{q_v}{q_u}y(u,t) - y(v,t)) \leq 2(|lower_u(t)|+1)$. Since $|lower_u(t)| \leq 4$, $|\frac{y(u,t)}{q_u} - \frac{y(v,t)}{q_v}| \leq \frac{10}{q_v\rho_2} \leq \frac{10}{q_{min}\rho_2}$. By induction on the number of hops from 1 to $2m-1$ between any two VN s, the result follows. □