Motion Coordination using Virtual Nodes

Nancy Lynch, Sayan Mitra, and Tina Nolte
CSAIL, MIT
32 Vassar Street
Cambridge, MA 02139, USA
{lynch,mitras,tnolte}@csail.mit.edu

***Abstract*— We describe how a virtual node abstraction layer can be used to coordinate the motion of real mobile nodes in a region of 2-space. In particular, we consider how nodes in a mobile ad hoc network can arrange themselves along a predetermined curve in the plane, and can maintain themselves in such a configuration in the presence of changes in the underlying mobile ad hoc network, specifically, when nodes may join or leave the system or may fail. Our strategy is to allow the mobile nodes to implement a virtual layer consisting of mobile client nodes, stationary Virtual Nodes (VNs) at predetermined locations in the plane, and local broadcast communication. The VNs coordinate among themselves to distribute the client nodes relatively evenly among the VNs' regions, and each VN directs its local client nodes to form themselves into the local portion of the target curve.**

***Index Terms*— Motion coordination, virtual nodes, hybrid systems, hybrid I/O automata.**

I. INTRODUCTION

Motion coordination is the general problem of achieving some global spatial pattern of movement in a set of autonomous agents. An important motivation for studying distributed motion coordination, that is, coordination among agents with only local communication ability and therefore limited knowledge about the state of the entire system, stems from the developments in the field of mobile sensor networks. Previous work in this area includes different coordination goals, for example: flocking [9], rendezvous [1], [10], [13], deployment [2], pattern formation [15], and aggregation [7]. Owing to the intrinsic decentralized nature of sensor network applications like surveillance, search and rescue, monitoring, and exploration, centralized or leader based approaches are ruled out. However, the lack of central control makes the programming task quite difficult.

In prior work [3], [4], [5], [6], we have developed a notion of "virtual nodes" for mobile ad hoc networks. A virtual node is an abstract, relatively well-behaved active node that is implemented using less well-behaved real nodes. Virtual nodes can be used to solve problems such as providing atomic memory [4], geographic routing [3], and point-to-point routing [6].

In this paper, we explore the use of virtual nodes in solving motion coordination problems. Namely, we consider virtual nodes associated with predetermined, well-distributed locations in the plane, communicating among themselves and with mobile "client nodes" using local broadcast. We describe one way of implementing such virtual nodes using the real mobile nodes, and describe how such virtual nodes can be used to solve a simple motion coordination problem. We use the Hybrid I/O Automata (HIOA) mathematical framework [11] for describing the components in our systems.

The paper is organized as follows: Section II describes the underlying mobile network. Section III describes our virtual node layer. Section IV defines the motion coordination problem we consider. Section V describes an algorithm for solving this motion coordination problem using the virtual node layer. Section VII gives the proofs of correctness of the algorithm. Section VII outlines one way to implement the virtual node layer, and Section VIII concludes.

II. THE PHYSICAL LAYER

Our physical model of the system consists of a finite but unknown number of communicating physical nodes in a bounded square $\mathcal{B}$ in R^2. We assume that each node has a unique identifier from a set $\mathcal{I}$. Formally, our physical layer model consists of three types of HIOA (see Figure 1): (1) automata PN_i to model physical nodes with identifiers $i \in \mathcal{I}$, (2) a $LBcast$ automaton that models the local broadcast communication service between the physical nodes, and (3) a "real world" automaton RW to model the physical location of all the nodes and the real time.

Figure 2 shows the required components of each automaton PN_i; it may have other internal variables (initially set to unique initial values) and actions, which are not specified here. PN_i continuously receives from RW the current time as the input variable $realtime$ and its position as the input variable $\mathbf{x}_i$, and communicates its velocity to RW through the output variable $\mathbf{v}_i$. The speed of PN_i is bounded by v_c. The trajectories of the continuous variable $\mathbf{v}_i$ and the effects of the send and receive actions are unspecified. At each point PN_i is either in active or inactive mode; we assume that, initially, finitely many nodes are active. The fail_i input action sets the mode to inactive and the $\mathsf{recover}_i$ input action sets it to active. In inactive mode, all internal and output actions are disabled, no input action except $\mathsf{recover}_i$ affects the internal or output variables, and during trajectories, the locally-controlled variables remain

*Research supported by AFRL contract number F33615-010C-1850, DARPA/AFOSR MURI contract number F49620-02-1-0325, NSF ITR contract number CCR-0121277, and DARPA-NEST contract number F33615-01-C-1896.

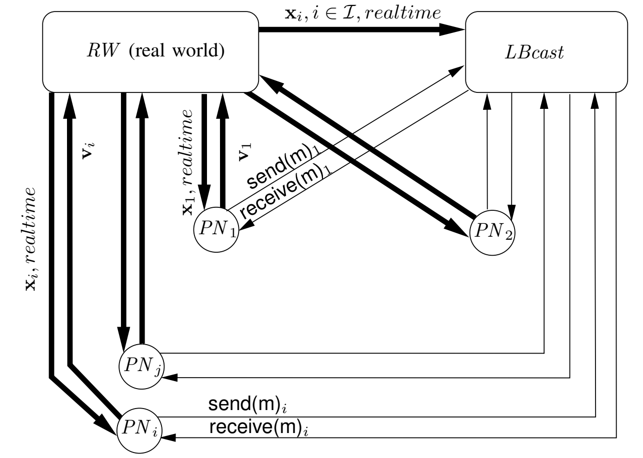

Fig. 1. The Physical Layer: PN automata communicate with each other through an $LBcast$ service and receive time and location information continuously from RW.

constant and the velocity $\mathbf{v}_i$ remains zero. Thus, we assume that, in inactive mode, PN_i stops moving. We model the departure of a node from $\mathcal{B}$ as a failure. For convenience, we assume that transitions are instantaneous.

Signature:
  Input
    receive$(m)_i$
    fail$_i$
    recover$_i$
  Output
    send$(m)_i$

Variables:
  Input
    $\mathbf{x}_i \in \mathcal{B}$
    $realtime \in R^{\geq 0}$
  Output
    $\mathbf{v}_i \in R^2, |\mathbf{v}_i| \leq v_c$, initially $\mathbf{0}$
  Internal
    $mode \in$ {active, inactive}
    Finite set of other variables, initially set to unique initial values.

Transitions:
  Input fail$_i$
  Effect
    $\mathbf{v}_i \leftarrow 0$
    $mode \leftarrow$ inactive
    Other internal variables $\leftarrow$ initial

  Input recover$_i$
  Effect
    $mode \leftarrow$ active

Fig. 2. Hybrid I/O Automaton PN_i.

The PNs communicate using a local broadcast service, *LBcast*, which is a generic local broadcast service parameterized by a radius R_p and a maximum message delay d_p. The $LBcast(R_p, d_p)$ service guarantees that when PN_i performs a send$(m)_i$ action at some time t, the message is delivered within the interval $[t, t + d_p]$, by a receive$(m)_j$ action, to every PN_j that remains in active mode and within R_p distance of PN_i for the entire interval $[t, t+d_p]$.

The RW automaton (see Figure 3) reads the velocity output $\mathbf{v}_i$ from each PN_i, $i \in \mathcal{I}$, and produces the position $\mathbf{x}_i$ for PN_i and the $LBcast$ automaton. $LBcast$ requires the node position information because it guarantees delivery only between "nearby" nodes. RW also produces $realtime$ for all physical layer components.

Variables:
  Input
    $\mathbf{v}_i \in R^2$, for each $i \in \mathcal{I}$
  Output
    $\mathbf{x}_i \in \mathcal{B}$, for each $i \in \mathcal{I}$, initially arbitrary
    $realtime \in R^{\geq 0}$, initially 0

Trajectories:
  Invariant
    $\mathbf{x}_i \in \mathcal{B}$, for each $i \in \mathcal{I}$
  Evolve
    $d(\mathbf{x}_i) = \mathbf{v}_i$, for each $i \in \mathcal{I}$
    $d(realtime) = 1$

Fig. 3. RW automaton.

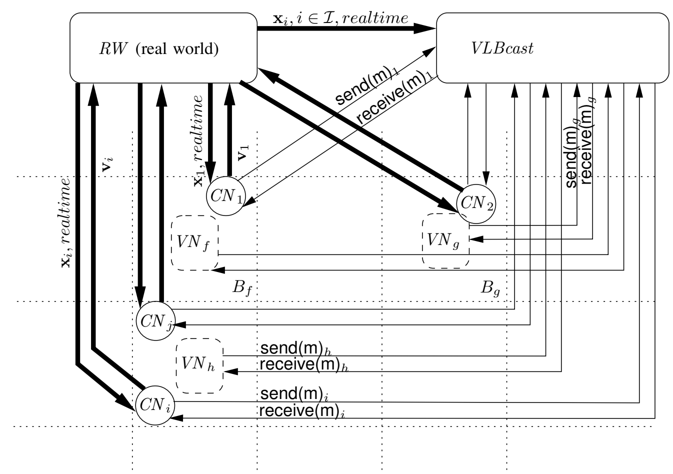

Fig. 4. Virtual Node Layer: VNs and CNs communicate using the $VLBcast$ service.

III. The Virtual Layer

The bounded square $\mathcal{B}$ is partitioned into a finite set of zones B_h, $h \in \mathcal{H}$. For simplicity we assume $\mathcal{B}$ is a $m \times m$ square grid, with each grid square corresponding to a zone and having sides of length b. Each boundary point of a square is unambiguously assigned to one zone. The index set $\mathcal{H}$ is the set of coordinates of the centers of all squares. For each B_h, the set $Nbrs_h$ contains the zone identifiers of the north, south, east, and west neighboring grid squares.

Our virtual layer abstraction (see Figure 4) consists of: (1) client node automata CN_i with identifiers $i \in \mathcal{I}$, (2) one stationary virtual node automaton VN_h for each $h \in \mathcal{H}$, located at the center $\mathbf{o}_h$ of the square B_h, (3) a virtual communication service, $VLBcast$ = $LBcast(R_v, d_v)$, for the VNs and the CNs, and (4) an automaton RW to model the physical location of all the CNs and the real time.

A client node automaton CN_i, $i \in \mathcal{I}$, is a portion of a PN_i automaton that has the input variables $realtime$ and $\mathbf{x}_i$ from the RW automaton and an output variable $\mathbf{v}_i$ to the RW automaton. With respect to failures, an automaton CN_i behaves the same as PN_i. CN_i also has send and receive actions for interacting with the $VLBcast$ service.

A virtual node automaton VN_h, $h \in \mathcal{H}$, is an MMT automaton [12], [14] parameterized by a time upper bound, d_{MMT}; it has no realtime clock variable. MMT automata are discrete I/O automata that have a "task" structure, which is an equivalence relation on the set of locally-controlled actions, such that from a point in an execution where a task becomes enabled, within at most time d_{MMT}, some action in that task must occur. VN_h can fail, disabling internal and output actions, preventing any inputs other than $\mathsf{recover}_h$ from resulting in state changes, and setting the automaton to an initial state. If a $\mathsf{recover}_h$ occurs, the VN actions become enabled with all tasks restarted. If VN_h is failed and a CN later enters B_h and remains active in the zone for d_r time, then a $\mathsf{recover}_h$ occurs within that d_r time. VN_h communicates with other VNs and CNs using the $VLBcast$ service through send_h and $\mathsf{receive}_h$ actions.

$VLBcast$ is an $LBcast$ service (as described in the physical layer) for the virtual layer, parameterized by radius R_v and maximum message delay d_v, where $R_v \geq b$. It allows VN_h to communicate with the VNs in the set $Nbrs_h$ and with CNs that are located in B_h. It does not allow CN automata to communicate with one another.

The RW automaton in the virtual layer is similar to the one in the physical layer, but here it communicates (through the $realtime$ and $\mathbf{x}$ variables) only with the CN automata and the $VLBcast$ automaton, and not the VN automata.

This virtual layer will be used in Section V to implement a solution to the distributed motion coordination problem. Details of how this virtual layer can be implemented using the physical layer are in Section VII. There we further discuss the relation between the parameters d_{MMT}, d_r, d_v, and R_p, the physical layer broadcast radius.

IV. The Motion Coordination Problem

A *differentiable parameterized curve* Γ is a differentiable map $P \rightarrow \mathcal{B}$, where the domain set P of parameter values is an interval in the real line. The curve Γ is *regular* if for every $p \in P$, $|\Gamma'(p)| \neq 0$. For $a, b \in P$, the *arc length* of a regular curve Γ from a to b, is given by $s(\Gamma, a, b) = \int_a^b |\Gamma'(p)| dp$. Γ is said to be *parameterized by arc length* if for every $p \in P$, $|\Gamma'(p)| = 1$. For a curve parameterized by arc length, $s(\Gamma, a, b) = b - a$.

For a given point $\mathbf{x} \in \mathcal{B}$, if there exists $p \in P$ such that $\Gamma(p) = \mathbf{x}$, then we say that the point $\mathbf{x}$ is on the curve Γ; abusing the notation, we write this as $\mathbf{x} \in \Gamma$. We say that Γ is a simple curve provided for every $\mathbf{x} \in \Gamma$, $\Gamma^{-1}(\mathbf{x})$ is unique. A sequence $\mathbf{x}_1, \ldots, \mathbf{x}_n$ of points in $\mathcal{B}$ are said to be *evenly 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, $1 \leq i \leq n$, $\Gamma(p_i) = \mathbf{x}_i$, and for each i, $1 < i < n$, $p_i - p_{i-1} = p_{i+1} - p_i$.

In this paper we fix Γ to be a simple, differentiable curve that is parameterized by arc length. Let $P_h = \{p \in P : \Gamma(p) \in B_h\}$ be the domain of Γ in zone $B_h \subset \mathcal{B}$. The local part of the curve Γ in zone B_h is the restriction $\Gamma_h : P_h \rightarrow B_h$. We assume that P_h is convex for every zone $B_h \subset \mathcal{B}$; it may be empty for some B_h. We write $|P_h|$ for the length of the curve Γ_h. The quantization of the length of Γ_h, with quantization constant $\sigma > 0$, is defined as $Q_\sigma(|P_h|) = \lceil \frac{|P_h|}{\sigma} \rceil \sigma$. For the remainder of the paper we fix σ and write $Q_\sigma(|P_h|)$ as Q_h. We also write Q_{min} and Q_{max} for the minimum and maximum Q_h, such that $P_h \neq \emptyset$.

Our goal is to design an algorithm that runs on the physical nodes such that, if there are no failures or recoveries after a certain point in time, then: (1) within finite time the set of nodes in each zone B_h, $h \in \mathcal{H}$, becomes fixed, and the size of this set is "approximately" proportional to the quantized length Q_h, (2) within finite time all physical nodes in B_h for which $Q_h \neq 0$ are located on Γ_h, and (3) in the limit all the nodes in each B_h are evenly spaced on Γ_h.

V. Solution Using Virtual Node Layer

In our algorithm each virtual node VN_h, $h \in \mathcal{H}$, uses only information about the portions of the target curve Γ in zone B_h and the neighboring zones. For convenience, we assume that all client nodes know the complete curve Γ; we could instead model the client nodes in B_h as receiving inputs from another automaton about the nature of the curve in zone B_h and neighboring zones only.

The Virtual Node abstraction is used as a means to coordinate the movement of client nodes in a zone. A VN controls the motion of the CNs in its zone by setting and broadcasting target waypoints for the CNs: VN_h periodically receives information from clients in its zone, exchanges information with its neighbors, and sends out a message containing a calculated target point for each client node "assigned" to zone h. Informally, VN_h performs two tasks when setting the target points: (1) it re-assigns some of the CNs that are assigned to itself to neighboring VNs, and (2) it sends a target position on Γ to each CN that is assigned to itself. The objective of (1) is to prevent neighboring VNs from getting depleted of CNs and to achieve a distribution of CNs over the zones that is proportional to the length of Γ in each zone. The objective of (2) is to space the nodes evenly on Γ in each zone. A CN, in turn, receives its current position information from RW and its target location from a VN, and continuously computes a velocity vector that will take it to its latest received target point.

A. Client Node Algorithm

The $CN(\delta)_i$, $i \in \mathcal{I}$, algorithm (see Figure 5) follows a *round* structure, where rounds begin at times that are multiples of δ. Recall that VN automata do not have access to $realtime$ whereas CN automata do. To help VNs follow the round structure, the CNs send "trigger" messages to prompt the VNs to perform transitions.

At the beginning of each round, a CN sends a $\mathsf{cn\text{-}update}$ message. The $\mathsf{cn\text{-}update}$ message tells the local VN (in whose zone the CN currently resides) the CN's *id*, assigned VN, current location in $\mathcal{B}$, and current round number.

Signature:
  Input
    receive$(m)_i$, $m \in (\{\text{target-update}\} \times \mathcal{H} \times \mathcal{B})$
  Output
    send$(m)_i$, $m \in (\{\text{cn-update}\} \times \mathcal{I} \times \mathcal{H} \times \mathcal{B} \times \mathsf{N})$
      $\cup (\{\text{exchange-trigger, target-trigger}\} \times \mathcal{B} \times \mathsf{N})$

Variables:
  Input
    $\mathbf{x}_i \in \mathcal{B}$
    $realtime \in R^{\geq 0}$
  Output
    $\mathbf{v}_i \in R^2$, velocity vector, initially $\mathbf{0}$
  Internal
    assigned $\in \mathcal{H}$, initially $h \in \mathcal{H}$ such that $\mathbf{x}_i \in B_h$
    $\mathbf{x}^* \in \mathcal{B}$, target point, initially same as $\mathbf{x}$
    round $\in \mathsf{N}$, initially $\lceil realtime/\delta \rceil$
    next-vn $\in R$, initially $\lceil realtime/\delta \rceil \cdot \delta + d_v + \epsilon$
    next-target $\in R$, initially $\lceil realtime/\delta \rceil \cdot \delta + d_{MMT} + 3d_v + 2\epsilon$

Transitions:
  Input receive($\langle$target-update, *h*, *target*$\rangle)_i$
  Effect
    if (*assigned* $= h \wedge$ *target(i)* $\neq$ *null*) **then**
      $\mathbf{x}^* \leftarrow target(i)$
      assigned $\leftarrow h \in \mathcal{H}$ such that $\mathbf{x}^* \in B_h$

  Internal send($\langle$cn-update, *i*, *assigned*, $\mathbf{x}_i$, *round*$\rangle)_i$
  Precondition
    realtime $=$ *round* $\cdot\ \delta$
  Effect
    round $\leftarrow$ *round* + 1

  Internal send($\langle$exchange-trigger, $\mathbf{x}_i$, *round* -1$\rangle)_i$
  Precondition
    realtime $=$ *next-vn*
  Effect
    next-vn $\leftarrow$ *next-vn* + δ

  Internal send($\langle$target-trigger, $\mathbf{x}_i$, *round* -1$\rangle)_i$
  Precondition
    realtime $=$ *next-target*
  Effect
    next-target $\leftarrow$ *next-target* + δ

Trajectories:
  Evolve
    if $\mathbf{x}_i = \mathbf{x}^*$ **then** $\mathbf{v}_i = \mathbf{0}$
    else $\mathbf{v}_i = v_c \cdot (\mathbf{x}^* - \mathbf{x}_i)/||\mathbf{x}^* - \mathbf{x}||$
  Stop when
    realtime $=$ *round* $\cdot\ \delta$ *or next-vn or next-target*

Fig. 5. Client node $CN(\delta)_i$ automaton.

The CN then sends an exchange-trigger message $d_v + \epsilon$ later to its local VN. An additional $d_{MMT} + 2d_v + \epsilon$ time later, the CN sends a target-trigger message to its local VN. Both these messages are trigger messages that include the CN's current location and the current round number, used by the local VN to determine whether the CN is in its zone and what the current round number is.

CN_i processes only one kind of message, target-update messages sent by its assigned VN (to which it is currently assigned). Each such message describes the new target location $\mathbf{x}_i^*$ for CN_i, and possibly an assignment to a different VN. CN_i continuously computes its velocity vector $\mathbf{v}_i$, based on its current position $\mathbf{x}_i$ and its target position $\mathbf{x}_i^*$, as $\mathbf{v}_i = v_c(\mathbf{x}_i - \mathbf{x}_i^*)/||\mathbf{x}_i - \mathbf{x}_i^*||$, moving it with maximum velocity towards the target.

B. Virtual Node Algorithm

In designing the motion coordination algorithm we make use of the apparent synchrony created by the virtual layer implementation. The $VN(e, \rho_1, \rho_2)_h$, $h \in \mathcal{H}$, algorithm (see Figure 6) follows the CNs' round structure. However, VNs do not have access to the $realtime$ variable and must instead rely on trigger messages from CNs to determine when enough time has elapsed to perform required actions. We begin by explaining how we implement the round structure for a VN and then explain the VN algorithm.

Round structure. At the beginning of a round, each CN sends a cn-update message to its local VN. The CNs then send exchange-trigger messages $d_v + \epsilon$ after the beginning of the round, signalling that the VN has received all cn-update messages that were transmitted at the beginning of the round in its zone. The VN waits before using information from the cn-update messages until it receives one of the CNs' exchange-trigger messages. The VN then sends vn-update messages to its neighbors.

Each CN sends a target-trigger message to its local VN $d_{MMT} + 2d_v + \epsilon$ time after it sends an exchange-trigger message. This is late enough in the round that: (1) neighboring VNs have received an exchange-trigger message (d_v time), (2) each neighboring VN has performed a vn-update transmission to its neighboring VNs, including this one (d_{MMT} time), and (3) the neighboring VN transmissions have arrived (d_v time). When a VN first receives a target-trigger message for a particular round from any CN in its region, it knows it has received any vn-update messages from neighboring VNs for the round. The VN then performs some computation and transmits a target-update message to CNs local to it.

A target-update message might not be received by a CN until $d_{MMT} + 2d_v$ time after the CN sent the target-trigger message. This accounts for: (1) the time it can take for the target-trigger message to be received by the VN (d_v), (2) the time it can take for the VN to perform the target-update broadcast (d_{MMT}), and (3) the time for the broadcast to be delivered at the CN (d_v). Given the maximum distance between a point in one zone and the center of a neighboring zone, $\sqrt{2.5}b = \sqrt{(3b/2)^2 + (b/2)^2}$, and a constant speed of v_c for each client node, it can take up to $\frac{\sqrt{2.5}b}{v_c}$ time for the CN to reach its target. Also, after the CN just arrives in the zone it was assigned to, up to $\sqrt{10}b/3 = \sqrt{2.5}b \cdot \frac{2}{3}$ distance from where it started, it could find that the local VN is failed, in which case it could take up to the d_r VN-startup time for the VN to recover.

To ensure a round is long enough for a client node to send the cn-update, exchange-trigger, and target-trigger messages, receive a target-update message, arrive at its new assigned target location, and be sure a virtual node is alive in its zone before a new round begins, we require that δ satisfy $\delta > 2d_{MMT} + 5d_v + 2\epsilon + max(\sqrt{2.5}b/v_c, \sqrt{10}b/3v_c + d_r)$.

VN algorithm. Each VN_h automaton collects cn-update messages sent at the beginning of the round from CNs located in its zone, aggregating the location and round information from the message in a table, M. When VN_h first receives an exchange-trigger message for a particular round from any CN in its zone, VN_h tallies and computes from its table 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 VH_h receives a vn-update message from a neighboring VN, it stores the CN population and round number information from the message in a table, V. When VN_h first receives a target-trigger message for a particular round from any CN in its region, VN_h uses the information in its tables M and V about the number of CNs in its zone and its neighbors' zones to calculate how many of the CNs assigned to itself should be reassigned and to which neighboring VNs. This is done through the assign function (see Figure 7) which calculates a partial function $assign$ mapping CN identifiers to zones that they are assigned to. If the number of CNs $y(h)$ assigned to VN_h exceeds the minimum critical number e, then the assign function reassigns some of the CNs to neighbors of VN_h.

Let In_h denote the set of neighboring VNs of VN_h that are on the curve Γ and $y_h(g)$, $g \in Nbrs_h \cup \{h\}$, denote the number $num(V_h(g))$ of CNs assigned to VN_g. If $Q_h \neq 0$, meaning VN_h is on the curve (lines 7–11), then we let $lower_h$ denote the subset of $Nbrs_h$ that are on the curve and have fewer assigned CNs than VN_h has after normalizing with $\frac{Q_g}{Q_h}$. For each $g \in lower_h$, VN_h reassigns either $ra = \rho_2 \cdot [\frac{Q_g}{Q_h} y_h(h) - y_h(g)]/2(|lower_h| + 1)$ or the number of nodes over e it has not already reassigned, whichever is smaller, of the CNs that are currently assigned to itself to VN_g, where $\rho_2 < 1$ is a damping factor.

If $Q_h = 0$, meaning VN_h is not on the curve, and VN_h has no neighbors on the curve (lines 13–17), then we let $lower_h$ denote the subset of $Nbrs_h$ that have fewer assigned CNs than VN_h. For each $g \in lower_h$, VN_h reassigns either $ra = \rho_2 \cdot [y_h(h) - y_h(g)]/2(|lower_h| + 1)$ or the number of nodes over e it has not already reassigned, whichever is smaller, of the CNs currently assigned to itself to VN_g.

VN_h is on a *boundary* if $Q_h = 0$, but there is a $g \in Nbrs_h$ with $Q_g \neq 0$. In this case, $y_h(h) - e$ of VN_h's CNs are assigned equally to neighbors in In_h (lines 19–22).

The client assignments are then used to calculate new target points for local CNs through the calctarget function (see Figure 7). This function assigns to every CN_i assigned to VN_h a target point $locM_h(i) \in B_g, g \in Nbrs_h \cup \{h\}$, to move to. The target point $locM_h(i)$ is computed as follows: If CN_i is assigned to VN_g, $g \neq h$, then its target is set to the center $\mathbf{o}_g$ of B_g (lines 30–31); if CN_i is assigned to

```
1  Signature:
2    Input
3      receive(m)_h, m ∈ ({cn-update} × I × H × B × N)
4                     ∪ ({exchange-trigger, target-trigger} × B × N)
5                     ∪ ({vn-update} × H × N × N)
6    Output
7      send(m)_h
8
9  Constants:
10   In = {g ∈ Nbrs: |P_g| ≠ 0}
11
12 State variables:
13   M : I → H × B × N, partial map from CN ids to assigned VN id,
14     current location, and round number, initially ∅.
15     Accessors: assigned, loc, round.
16   V : H → N × N, partial map from VN ids to the number of CNs, and
17     round number, initially {⟨g, ⟨0, 0⟩⟩} for each g ∈ Nbrs ∪ {h}.
18     Accessors: num, round.
19   send-buffer, queue of messages, initially ∅.
20   vn-done, target-done ∈ Z, initially 0.
21
22 Derived variables:
23   assignedM = {i ∈ id(M) : assigned(M(i)) = h}
24   locM = λ(i ∈ id(M)). loc(M(i))
25   y = λ(g: Nbrs ∪ {h}). num(V(g))
26
27 Transitions:
28   Input receive(⟨cn-update, id, assigned, loc, round⟩)_h
29   Effect
30     if loc ∈ B_h then
31       M ← M ∪ {⟨id, ⟨assigned, loc, round⟩⟩}
32
33   Input receive(⟨exchange-trigger, loc, round⟩)_h
34   Effect
35     if (loc ∈ B_h ∧ vn-done ≠ round) then
36       for each i ∈ id(M)
37         if round(M(i)) ≠ round then
38           M ← M \ {⟨i, M(i)⟩}
39       V(h) ← ⟨|assignedM|, round⟩
40       send-buffer ← send-buffer ∪ {⟨vn-update, h, y(h), round⟩}
41       vn-done ← round
42
43   Input receive(⟨vn-update, id, n, round⟩)_h
44   Effect
45     if id ∈ Nbrs then
46       V(id) ← ⟨n, round⟩
47
48   Input receive(⟨target-trigger, loc, round⟩)_h
49   Effect
50     if (loc ∈ B_h ∧ target-done ≠ round) then
51       for each i ∈ id(M)
52         if round(M(i)) ≠ round then
53           M ← M \ {⟨i, M(i)⟩}
54       V(h) ← ⟨|assignedM|, round⟩
55       for each g ∈ Nbrs
56         if round(V(g)) ≠ round then
57           V(g) ← ⟨0, 0⟩
58       let target = calctarget(assign(assignedM, y), locM)
59         send-buffer ← send-buffer ∪ {⟨target-update, h, target⟩}
60       target-done ← round
61
62   Output send(m)_i
63   Precondition
64     send-buffer ≠ ∅ ∧ m = head(send-buffer)
65   Effect
66     send-buffer ← tail(send-buffer)
67
68 Tasks and bounds:
69   {send(m)_h}, bounds [0, d_MMT]
```

Fig. 6. $VN(e, \rho_1, \rho_2)_h$ IOA signature, variables, transitions, and tasks, implementing motion coordination algorithm with parameters: safety e, and damping ρ_1, ρ_2.

```
    Functions:
2     function assign(assignedM: 2^𝓘, y: Nbrs ∪{h} → N): 𝓘 → 𝓗 =
        assign: 𝓘 → 𝓗, initially {⟨i, h⟩} for each i ∈ assignedM
4       n: N, initially y(h)
        ra: N, initially 0
6       if y(h) > e then
          if Q_h ≠ 0 then
8           let lower = {g ∈ In: (Q_g/Q_h) y(h) > y(g)}
              for each g ∈ lower
10              ra ← min(⌊ρ_2 · [(Q_g/Q_h) y(h) − y(g)]/2(|lower|+1)⌋, n − e)
                update assign by reassigning ra nodes from h to g
12              n ← n − ra
          else if In = ∅ then
14          let lower = {g ∈ Nbrs : y(h) > y(g)}
              for each g ∈ lower
16              ra ← min(⌊ρ_2 · [y(h) − y(g)]/2(|lower|+1)⌋, n − e)
                update assign by reassigning ra nodes from h to g
18              n ← n − ra
          else
20          ra ← ⌊(y(h) − e)/|In|⌋
            for each g ∈ In
22            update assign by reassigning ra nodes from h to g
        return assign
24
      function calctarget(assign: 𝓘 → 𝓗, locM: 𝓘 → 𝓑): 𝓘 → 𝓑 =
26      seq, indexed list of pairs in P × 𝓘, initially the list, for each i ∈ 𝓘 :
          assign(i)= h ∧ locM(i) ∈ Γ_h, of ⟨p, i⟩ where p= Γ_h^{-1}(locM(i)),
28        sorted by p, then i
        for each i ∈ 𝓘 : assign(i) ≠ null
30        if assign(i) = g ≠ h then
            locM(i) ← o_g
32        else if locM(i) ∉ Γ_h then
            locM(i) ← choose {min_{x∈Γ_h} {dist(x, locM(i))}}
34        else let p = Γ_h^{-1}(locM(i)), seq(k) = ⟨p, i⟩
            if k = first(seq) then locM(i) ← Γ_h(inf(P_h))
36          else if k = last(seq) then locM(i) ← Γ_h(sup(P_h))
            else let seq(k − 1) = ⟨p_{k−1}, i_{k−1}⟩, seq(k + 1) = ⟨p_{k+1}, i_{k+1}⟩
38            locM(i) ← Γ_h(p + ρ_1 · ((p_{k−1}+p_{k+1})/2 − p))
        return locM
```

Fig. 7. $VN(e, \rho_1, \rho_2)_h$ IOA functions.

VN_h but is not located on the curve Γ_h then its target is set to the nearest point on the curve, nondeterministically choosing one if there are several (lines 32–33); if CN_i is either the first or last client node on Γ_h then its target is set to the corresponding endpoint of Γ_h (lines 35–36); if CN_i 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 38). For the last two computations a sequence seq_h of nodes on the curve sorted by curve location is used (line 27).

VN_h finally broadcasts the new target waypoints for the round through a target-update message to its CNs.

VI. Correctness and Performance

We say $CN_i, i \in \mathcal{I}$, is *active* in round t if its mode is active for the duration of round t. A VN_h, $h \in \mathcal{H}$, is active in round t if there is some active CN_i with $\mathbf{x}_i \in B_h$ for the duration of rounds $t-1$ and t. None of the VNs are active in the starting round 0. We use the following notation: $In(t)$ is the set of ids $h \in \mathcal{H}$ of VNs that are active in round t and for which $Q_h \neq 0$. $Out(t)$ is the set of ids $h \in \mathcal{H}$ of VNs that are active in round t and for which $Q_h = 0$. $C(t)$ is the set of active CNs at round t, and $C_{in}(t)$ and $C_{out}(t)$ are the sets of active CNs located in zones in $In(t)$ and $Out(t)$, respectively, at the beginning of round t.

For any pair of neighboring zones g and h, and for any round t, we use $y_g(h)(t)$ to refer to the value of $y_g(h)$ at the point in time in round t when VN_g finishes processing the first target-trigger message of round t. For any $f, g \in Nbrs_h \cup \{h\}$, in the absence of failures and recoveries of CNs in round t, $y_f(h)(t) = y_g(h)(t)$; we write this simply as $y_h(t)$.

In the following subsection we prove that the VN algorithm satisfies our first goal, that is, if there are no failures or recoveries of CNs after a certain round t_0, then within a finite number of rounds after t_0, a round T_{stab} is reached after which: (1) the set of CNs assigned to each VN is fixed, and (2) the number of CNs assigned to each VN_h such that $Q_h \neq 0$ is proportional to Q_h within a constant additive factor.

A. Assignments Stabilize

For each of the following lemmas, we assume that there are no failures or recoveries of CNs after round t_0. The first lemma states some basic facts about the assign function (see Figure 7):

Lemma 1: In every round $t > t_0$: (1) $In(t) \subseteq In(t+1)$, (2) $Out(t) \subseteq Out(t+1)$, (3) $C_{in}(t) \subseteq C_{in}(t+1)$, (4) $C_{out}(t+1) \subseteq C_{out}(t)$, and (5) if $y_h(t) \geq e$ for some $h \in \mathcal{H}$, then $y_h(t+1) \geq e$.

The next lemma states a key property of the assign function after round t_0: $VN_h, h \in Out(t)$, is never assigned a larger number of CNs in round $t+1$ than the largest number of CNs that were assigned to any of VN_h's neighbors in round t. A similar property holds for VN_h, $h \in In(t)$, with respect to the density of CNs.

Lemma 2: In every round $t > t_0$, for $g, h \in \mathcal{H}$ with $h \in Nbrs_g$:
(1) If $g, h \in Out(t)$, $y_h(t) = max_{f \in Nbrs_g} y_f(t)$, and $y_g(t) < y_h(t)$, then $y_g(t+1) \leq y_h(t) - 1$, and
(2) If $g, h \in In(t)$, $\frac{y_h(t)}{Q_h} = max_{f \in Nbrs_g} \frac{y_f(t)}{Q_f}$, and $\frac{y_g(t)}{Q_g} < \frac{y_h(t)}{Q_h}$, then $\frac{y_g(t+1)}{Q_g} \leq \frac{y_h(t)}{Q_h} - \frac{\sigma}{Q_{max}^2}$.

Proof: (1) Fix g, h and t, as in the statement of the lemma. Since $y_h(t) > y_g(t)$ and $g, h \in Out(t)$, we see from line 16 of Figure 7 that the number of CNs that VN_g is assigned from VN_h in round t is at most $\rho_2(y_h(t) - y_g(t))/2(|lower_h(t)| + 1)$. This is at most $\rho_2(y_h(t) - y_g(t))/4$, because $y_h(t) > y_g(t)$ implies that $lower_h(t) \geq 1$. Then, the total number of CNs assigned to VN_g in round t by all four of its neighbors is at most $\rho_2(y_h(t) - y_g(t))$. Therefore, $y_g(t+1) \leq y_g(t) + \rho_2(y_h(t) - y_g(t)) = \rho_2 y_h(t) + (1 - \rho_2) y_g(t)$. As $\rho_2 < 1$, we have $y_g(t+1) < y_h(t)$. The result follows from integrality of $y_g(t+1)$ and $y_h(t)$.

(2) As in part 1, fix g, h and t. Here $\frac{y_h(t)}{Q_h} > \frac{y_g(t)}{Q_g}$ and $g, h \in In(t)$. From line 10 of Figure 7, it follows that the number of CNs that VN_g is assigned from VN_h in round t is at most $\rho_2(\frac{Q_g}{Q_h} y_h(t) - y_g(t))/2(|lower_h(t)| + 1)$. This is at most $\rho_2(\frac{Q_g}{Q_h} y_h(t) - y_g(t))/4$. Then, the total number of CNs assigned to VN_g in round t by all four of its neighbors is at most $\rho_2(\frac{Q_g}{Q_h} y_h(t) - y_g(t))$. Therefore, $y_g(t+1) \leq (1-\rho_2) y_g(t) + \rho_2 \frac{Q_g}{Q_h} y_h(t)$, that is $\frac{y_g(t+1)}{Q_g} \leq (1-\rho_2)\frac{y_g(t)}{Q_g} + \rho_2 \frac{y_h(t)}{Q_h}$. As $\rho_2 < 1$, we have $\frac{y_g(t+1)}{Q_g} < \frac{y_h(t)}{Q_h}$. A simple calculation shows that if $\frac{y_h(t)}{Q_h} \neq \frac{y_g(t)}{Q_g}$, then $\frac{y_h(t)}{Q_h} - \frac{y_g(t)}{Q_g} \geq \frac{\sigma}{Q^2_{max}}$. ∎

Lemma 3: There exists a round $T_{in} > t_0$ such that in any round $t \geq T_{in}$, the number of CNs assigned to VN_h, $h \in Out(t)$, is unchanged: $y_h(t+1) = y_h(t)$.

Proof: Let N_{out} be the total number of $h \in \mathcal{H}$ such that $Q_h = 0$. For any k, $1 \leq k \leq N_{out}$, we define $max_k(t)$ to be the k^{th} largest number of CNs that are assigned to any VN_h, $h \in Out(t)$, at the beginning of round $t > t_0$:

$$max_k(t) \triangleq \begin{cases} max\{y_h(t) \ : h \in Out(t)\}, & \text{if } k = 1 \\ max\{y_h(t) \ : h \in Out(t) \wedge \\ \qquad y_h(t) < max_{k-1}(t)\}, & \text{otherwise.} \end{cases}$$

Let $maxvns_k(t)$ be the set of VN ids that have $max_k(t)$ CNs assigned to them. If there exists an l, $1 \leq l \leq N_{out}$, such that $\forall h \in Out(t) : max_l(t) \geq y_h(t)$, then for all k, $l < k \leq N_{out}$, $max_k(t) = 0$ and $maxvns_k(t) = \emptyset$.

Let $E(t) = (|C_{out}(t)|, \ max_1(t), |maxvns_1(t)|, \ldots, max_{N_{out}}(t), |maxvnx_{N_{out}}(t)|)$. Let w be the minimum $y_h(t_0)$ for any $h \in Out(t_0)$, and $S = \{h \in Out(t_0) \ : y_h(t_0) = w\}$. Observe that if $w < e$, then $E_{min} = (w|S|, w, |S|, 0, 0 \ldots, 0, 0)$ is a minimum value for $E(t)$, otherwise $E_{min} = (e|S|, e, |S|, 0, 0 \ldots, 0, 0)$ is a minimum value. It suffices to show that for any round $t > t_0$, either $E(t+1) = E(t)$, that is, $t = T_{in}$, or $E(t+1)$ is less than $E(t)$ by some constant amount, meaning there is a $k, 1 \leq k \leq N_{out}$, such that for every $l, 1 \leq l < k$, the l^{th} component of $E(t+1)$ is equal to the l^{th} component of $E(t)$, and the k^{th} component of $E(t+1)$ is less than the k^{th} component of $E(t)$ by at least 1.

Consider any round t after t_0. From Lemma 1 we know that $|C_{out}(t+1)| \leq |C_{out}(t)|$. If $|C_{out}(t+1)| < |C_{out}(t)|$, then the first component of $E(t+1)$ is less than that of $E(t)$ by at least 1. Otherwise, $|C_{out}(t+1)| = |C_{out}(t)|$. If for every $h \in Out(t)$, $ra = 0$ for all $g \in lower_h(t)$ (see line 16 of Figure 7), then none of the CNs in $C_{out}(t)$ are reassigned in round $t+1$, and $E(t+1) = E(t)$. Setting $T_{in} = t$, we are done. Otherwise, there exists a nonempty set of VNs with ids in $Out(t)$ that reassign some CNs to a neighboring VN. We select the nonempty set A of such VNs with the highest number of assigned CNs. Let $A \subseteq maxvns_k(t)$, for some $k, 1 \leq k \leq N_{out}$.

For any $g \in Out(t)$ with $y_g(t) < max_k(t)$, the maximum value of $y_h(t)$ for any $h \in Nbrs_g$ such that VN_g gets some CNs from VN_h in round t is at most $max_k(t)$. From Part(1) of Lemma 2 it follows that $y_g(t+1) \leq max_k(t) - 1$.

For any VN_h, $h \in A$, since no VN with $y > max_k(t)$ assigns any CNs to VN_h, $y_h(t+1) = y_h(t) - \sum_{g \in lower_h(t)} ra_g(t)$, where ra_g is the number of CNs VN_h assigns to its neighbor VN_g in round t. We have shown above that for any $g \in Out(t)$, if $y_g(t) < max_k(t)$ then $y_g(t+1) \leq max_k(t) - 1$. There are two possible cases: (1) if $maxvns_k(t) = A$, then the k^{th} max decreases, $max_k(t+1) \leq max_k(t) - 1$. That is, the $(2k+1)^{st}$ component of E decreases by at least 1, and (2) if $A \subset maxvns_k(t)$, then $max_k(t+1) = max_k(t)$ and $|maxvns_k(t+1)| = |maxvns_k(t)| - |A|$. That is, the $(2k+2)^{nd}$ component of E decreases by at least 1. ∎

Corollary 1: In every round $t \geq T_{in}$, the set of CNs assigned to VN_h, $h \in Out(t)$, is unchanged.

Proof: Suppose the set of CNs assigned to VN_h changes in some round $t \geq T_{in}$. We know that $y_h(t+1) = y_h(t)$ for all $h \in Out(t)$. Summing, $|C_{out}(t+1)| = |C_{out}(t)|$ and using Lemma 1 we get $C_{out}(t+1) = C_{out}(t)$. The only way the set of CNs assigned to VN_h could change, without changing y_h and the set C_{out}, is if there existed a cyclic sequence of VNs with ids in $Out(t)$ in which each VN gives up $c > 0$ CNs to its successor VN in the sequence, and receives c CNs from its predecessor. However, such a cycle of VNs cannot exist because the $lower$ set imposes a strict partial ordering on the VNs. ∎

Corollary 1 implies that in every round $t \geq T_{in}$, $In(t) = In(T_{in}), C_{in}(t) = C_{in}(T_{in})$, and $C_{out}(t) = C_{out}(T_{in})$; we denote these simply as In, C_{in}, and C_{out}.

Corollary 2: $|C_{out}| = O(m^3)$.

Proof: From Corollary 1, the set of CNs assigned to each VN_h, $h \in Out(t)$, is unchanged in every round $t \geq T_{in}$. This implies that in any round $t \geq T_{in}$, the number of CNs assigned by VN_h to any of its neighbors is 0. Therefore, from line 20 of Figure 7, for any boundary VN_g, $(y_g(t) - e)/|In_g| < 1$. In_g is the (constant) set of $h \in Nbrs_g$ with $Q_h \neq 0$. Since $|In_g| \leq 4$, $y_g(t) < 4 + e$. From line 16 of Figure 7, for any non-boundary VN_g, $g \in Out(t)$, that is 1-hop away from a boundary VN_h, $\frac{\rho_2(y_g(t) - y_h(t))}{2(|lower_g(t)|+1)} < 1$. Since $|lower_g(t)| \leq 4$, $y_g(t) \leq \frac{10}{\rho_2} + 4 + e$. Inducting on the number of hops, the maximum number of CNs assigned to a VN_g, $g \in Out(t)$, at l hops from the boundary is at most $\frac{10l}{\rho_2} + e + 4$. Since for any l, $1 \leq l \leq 2m-1$, there can be at most m VNs at l-hop distance from the boundary, summing gives $|C_{out}| \leq (e+4)(2m-1)m + \frac{10m^2(2m-1)}{\rho_2} = O(m^3)$. ∎

Lemma 4: There exists a round $T_{stab} \geq T_{in}$ such that in every round $t \geq T_{stab}$, the set of CNs assigned to VN_h, $h \in In$, is unchanged.

The next lemma states that the number of CNs assigned to each VN_h, $h \in In$, in the stable assignment after T_{stab} is proportional to Q_h within a constant additive factor.

Lemma 5: In every round $t \geq T_{stab}$, for $g, h \in In(t)$:

$$\left|\frac{y_h(t)}{Q_h} - \frac{y_g(t)}{Q_g}\right| \leq \left[\frac{10(2m-1)}{Q_{min}\rho_2}\right].$$

Proof: Consider a pair of VNs for neighboring zones B_g and B_h, $g, h \in In$. Assume w.l.o.g. $y_h(t) \geq y_g(t)$. From line 10 of Figure 7, it follows that $\rho_2(\frac{Q_g}{Q_h} y_h(t) - y_g(t)) \leq 2(|lower_h(t)|+1)$. Since $|lower_h(t)| \leq 4$, $|\frac{y_h(t)}{Q_h} - \frac{y_g(t)}{Q_g}| \leq \frac{10}{Q_g \rho_2} \leq \frac{10}{Q_{min}\rho_2}$. By induction on the number of hops from 1 to $2m-1$ between any two VNs, the result follows. ∎

B. On the Curve and Evenly Spaced

We continue to assume that there are no failures or recoveries of CNs after round t_0.

From line 33 of Figure 7, it follows immediately that by the beginning of round $T_{stab}+2$, all CNs in C_{in} are located on the curve Γ. This establishes that the VN algorithm satisfies our second goal. In the remainder of this section, we prove that the locations of the CNs in each zone B_h, $h \in In$, are evenly spaced on Γ_h in the limit.

Lemma 6: Consider a sequence of rounds $t_1 = T_{stab}, \ldots, t_n$. As $n \to \infty$, the locations of CNs in B_h, $h \in In$, are evenly spaced on Γ_h.

Proof: From Lemma 4 we know that the set of CNs assigned to each VN_h, $h \in In$, remains unchanged. Then, at the beginning of round t_2, every CN assigned to VN_h is located in B_h and is on the curve Γ_h. Assume w.l.o.g. that VN_h is assigned at least two CNs. Then, at the beginning of round t_3, one CN is positioned at each endpoint of Γ_h, namely at $\Gamma_h(inf(P_h))$ and $\Gamma_h(sup(P_h))$. From lines 35–36 of Figure 7, we see that the target points for these *endpoint* CNs are not changed in successive rounds. Let $seq_h(t_2) = \langle p_0, i_{(0)}\rangle, \ldots, \langle p_{n+1}, i_{(n+1)}\rangle$, where $y_h = n+2$, $p_0 = inf(P_h)$, and $p_{n+1} = sup(P_h)$. From line 38 of Figure 7, for any i, $1 < i < n$, the i^{th} element in seq_h 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} evenly spaced point on the curve Γ_h between the two endpoints be $\bar{\mathbf{x}}_i$. The parameter value $\bar{p}_i$ corresponding to $\bar{\mathbf{x}}_i$ 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 CNs are evenly spaced on Γ_h. [[The rest of this proof is exactly the same as the proof of Theorem 3 in [8]. They prove convergence of points on a straight line with even spacing, which is the same as proving convergence of the parameters in our case. I am writing this here to make the proof complete, but we should just cite their paper.]]

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, $2 \leq i \leq n-1$, $e_i(k+1) = p_i(t_{k+1}) - \bar{p}_i = (1-\rho_1)e_i(k) + \frac{\rho_1}{2}(e_{i-1}(k) + e_{i+1}(k))$, $e_1(k+1) = (1-\rho_1)e_1(k) + \frac{\rho_1}{2}e_2(k)$, 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, $\rho_1 \leq 1$, 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 $lim_{k\to\infty} e(k) = 0$. ∎

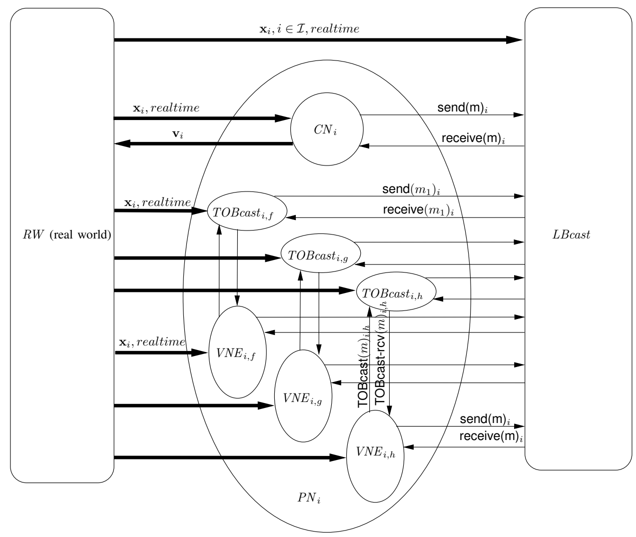

Fig. 8. PN_i's subautomata: A physical node runs several programs, including VNE and $TOBcast$ automata as well as a CN automaton.

VII. Implementing the Virtual Node Layer

In addition to client CN_i, a physical node PN_i, $i \in \mathcal{I}$, in zone B_h runs a $TOBcast_{i,h}$ service and a $VNE_{i,h}$, $h \in \mathcal{H}$, algorithm (see Figure 8) to help implement each virtual node VN_h and the $VLBcast$ service of the virtual layer.

In this section we present a sketch of our implementation of the virtual layer by the physical layer. Our implementation is an adaptation of techniques from [6] to emulate a virtual mobile node. The only substantive changes made in our current implementation are: (1) the changing of virtual node locations to be stationary, (2) the replacement of a periodic location update with a continuous real-time location update, and (3) the restart of a virtual node as soon as a physical node discovers it is in a failed virtual node's zone. The virtual nodes we implement here are also modeled differently from those in [6], as MMT automata, rather than simple I/O automata.

We use a standard replicated state machine approach to implement robust virtual nodes that takes advantage of a $TOBcast$ service to ensure that all VNEs in a zone receive the same messages in the same order. Using the $LBcast$

service of the physical nodes and common knowledge about $realtime$, the totally ordered broadcast service $TOBcast$ for a zone can be implemented as follows: At the time of sending, a message is tagged with the sender's identifier, zone id, and a timestamp, which is the current value of $realtime$. Assuming that a PN does not make multiple broadcasts at the same point in time, the tags define a total order on sent messages. Before delivering a message $TOBcast_{i,h}$ waits until $d_p + \epsilon$ time has elapsed since it was sent, ensuring that earlier messages were received. $TOBcast_{i,h}$ only processes messages tagged for zone B_h.

Each $VNE_{i,h}$ independently maintains the state of VN_h and simulates performing actions of the VN on that state. In order to keep the state replication consistent across different VNEs running on different physical nodes in the same zone, when $VNE_{i,h}$ wants to simulate an action of the VN, it broadcasts a suggestion to perform the action to the other VNEs of the region using the $TOBcast$ service. This action could, for example, be a suggestion to receive a message on behalf of the VN that was actually received by $VNE_{i,h}$. When an action suggestion is received by $VNE_{i,h}$, it is saved in a $pending\text{-}action$ queue. Actions are removed from a $pending\text{-}action$ queue in order by $VNE_{i,h}$ and simulated on $VNE_{i,h}$'s local version of the VN state. A completed action is then moved into a $completed\text{-}action$ queue, referenced by $VNE_{i,h}$ to prevent reprocessing of completed actions.

When a VNE enters a zone, it executes a join protocol to get the zone's VN state. The join protocol begins by using $TOBcast$ to send a join-req message. Whenever a VNE receives its own join-req message, it starts saving messages to process in its $pending\text{-}action$ queue. If a VNE that has already joined receives the join-req, it uses $TOBcast$ to send a join-ack containing a copy of its version of the VN state. When the joining VNE receives the join-ack, it copies the included VN state and starts processing the actions in its $pending\text{-}action$ queue. If a VNE's join-req is not answered in $2d_p + 2\epsilon$ time, indicating the VN is failed, the VNE will reset the VN ϵ time later by using $TOBcast$ to send a reset message. When a VNE receives a reset message, it sets the VN state to its initial state, clears the $pending\text{-}action$ queue, and starts simulating the VN.

Theorem 1: Assuming $R_p \geq \sqrt{5}b$, the $TOBcast_{i,h}$, $VNE_{i,h}$, $i \in \mathcal{I}, h \in \mathcal{H}$, and trivial client implementation correctly implement the Virtual Node abstraction with VN task upper time bound $d_{MMT} = d_p + \epsilon$, VN-startup time $d_r = 3d_p + 4\epsilon$, $VLBcast$ broadcast radius $R_v \geq b$, and $VLBcast$ maximum message delay $d_v = 2d_p + \epsilon$.

Proof: The correctness of the implementation of the Virtual Node layer largely follows from the proof of correctness for the implementation of the VMN layer in [6]. We here discuss the correctness of the implementation with respect to: (1) the task upper bound, (2) the VN-startup time, and (3) the requirements for $LBcast$ and $VLBcast$.

(1) Once one of an abstract VN_h's output or internal transitions is enabled, the precondition for sending a suggestion to simulate the action through $TOBcast$ is satisfied at all $VNE_{i,h}$ for PN_i in B_h, and the broadcast occurs. It takes at most $d_p + \epsilon$ time for the message to be delivered at other $VNE_{i,h}$ for PN_i in B_h, after which the action is simulated. Given that PN transitions are assumed to be instantaneous, $d_{MMT} = d_p + \epsilon$.

(2) If PN_i enters a zone B_h with a failed VN, its $VNE_{i,h}$'s join-req will not be answered in $2d_p + 2\epsilon$ time, and the VNE will send a reset message an additional ϵ later. It takes the VNE at most $d_p + \epsilon$ time to receive the reset message and restart the VN. The total time $3d_p + 4\epsilon$ for a joining node to succeed in restarting a VN is d_r.

(3) As in [6], $d_v = 2d_p + \epsilon$ since the underlying $LBcast$ service used to implement $VLBcast$ takes up to d_p time to deliver a transmitted message from a VN or CN, after which $TOBcast$ takes an additional $d_p + \epsilon$ time to redeliver a message at a receiving VN. Also similarly to [6], we require that $R_p \geq \sqrt{5}b$, in order to guarantee that $R_v \geq b$, allowing a CN_i in B_h, $i \in \mathcal{I}$, $h \in \mathcal{H}$, and VN_h to communicate, and a VN_h (located at $\mathbf{o}_h$) and each of its neighboring zones' VN_g, $g \in Nbrs(h)$, (located at $\mathbf{o}_g$) to communicate. This is because a VNE emulating a zone B_h can be as far away as $\sqrt{(2b)^2 + b^2}$ from a VNE emulating the VN of neighboring zone B_g. To guarantee the two can communicate while emulating their respective VNs, the broadcast radius R_p of the physical $LBcast$ service must be be at least $\sqrt{5}b$. Unlike [6], however, we do not require an additional tolerance factor to account for periodic location updates from the RW; here, the RW automaton is assumed to continually update the VNE of its current location. ■

VIII. Conclusions

Future work/extensions: In our algorithm each virtual node VN_h, $h \in \mathcal{H}$, uses only local information about the target curve Γ. We can consider a problem extension where the curve is dynamically changing. The curve (or point, even) could be moving targets being tracked. In this case, the coordination of nodes we talked about here is important for two big reasons: (1) maintaining alive VNs to detect targets and (2) guiding physical nodes to the moving targets. The fact that we employed a local solution here for curve discovery should adapt well to this more dynamic problem.

It would be possible to modify our algorithm to allow shorter rounds that don't require completed relocation of client nodes; instead we could, for example, have VNs update their neighboring region VNs of the client nodes that are currently in transit to them.

References

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

[2] Jorge Cortes, Sonia Martinez, Timur Karatas, and Francesco Bullo. Coverage control for mobile sensing networks. *IEEE Transactions on Robotics and Automation*, 20(2):243–255, 2004.

[3] Shlomi Dolev, Seth Gilbert, Limor Lahiani, Nancy Lynch, and Tina Nolte. Virtual stationary automata for mobile networks. *Technical Report MIT-LCS-TR-979*, 2005.

[4] Shlomi Dolev, Seth Gilbert, Nancy Lynch, Alex Shvartsman, and Jennifer Welch. Geoquorums: Implementing atomic memory in mobile ad hoc networks. In *17th International Symposium on Distributed Computing (DISC)*, 2003.

[5] Shlomi Dolev, Seth Gilbert, Nancy Lynch, Alex Shvartsman, and Jennifer Welch. Geoquorums: Implementing atomic memory in mobile ad hoc networks. *Technical Report MIT-LCS-TR-900*, 2003.

[6] Shlomi Dolev, Seth Gilbert, Nancy A. Lynch, Elad Schiller, Alexander A. Shvartsman, and Jennifer L. Welch. Virtual mobile nodes for mobile ad hoc networks. In *18th International Symposium on Distributed Computing (DISC)*, pages 230–244, 2004.

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

[8] David Kiyoshi Goldenberg, Jie Lin, and A. Stephen Morse. Towards mobility as a network control primitive. In *MobiHoc '04: Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing*, pages 163–174. ACM Press, 2004.

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

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

[11] Nancy Lynch, Roberto Segala, and Frits Vaandrager. Hybrid I/O automata. *Information and Computation*, 185(1):105–157, August 2003.

[12] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufman, 1996.

[13] Sonia Martinez, Jorge Cortes, and Francesco Bullo. On robust rendezvous for mobile autonomous agents. In *IFAC World Congress*, Prague, Czech Republic, 2005. To appear.

[14] Michael Merritt, Francemary Modugno, and Mark Tuttle. Time constrained automata. In *2nd International Conference on Concurrency Theory (CONCUR)*, 1991.

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

APPENDIX

Signature:
Input
$\mathsf{receive}(m)_i$, m a client message
$\mathsf{TOBcast\text{-}rcv}(m)_i$, m a $TOBcast$ message
Output
$\mathsf{send}(m)_i$, m a client message
$\mathsf{TOBcast}(m)_i$ m a $TOBcast$ message
Internal
$\mathsf{zone\text{-}update}_i$
join_i
$\mathsf{restart}_i$
$\mathsf{init\text{-}action}(act)_i$, $act \in VN_h.sig$
$\mathsf{simulate\text{-}action}(act)_i$, $act \in VN_h.sig$

Variables:
Input
$\mathbf{x}_i \in \mathcal{B}$, current location of mobile node
$realtime \in R^{\geq 0}$
Internal
$status \in \{\mathsf{joining}, \mathsf{listening}, \mathsf{active}\}$, initially active
$h \in \mathcal{H} \cup \{\perp\}$, zone id, initially $\perp$
$val \in VN_h.states$, state of VN_h, initially $VN_h.start$
answered-joins, set of ids of answered join reqs, initially $\emptyset$
join-id, a tuple of time and a mobile node id, initially $\langle 0, i\rangle$
pending-actions, queue of $VN_h.actions$ to be simulated, initially $\emptyset$
completed-actions, queue of $VN_h.actions$ simulated, initially $\emptyset$
TOBcast-out, queue of outgoing $TOBcast$ msgs, initially $\emptyset$
local-out, queue of outgoing client messages, initially $\emptyset$

Fig. 9. Signature and variables of $VNE_{i,h}$ algorithm implementing VN_h.

Input $\mathsf{receive}(m)_i$
Effect
$TOBcast\text{-}out \leftarrow TOBcast\text{-}out \cup \{\langle \mathsf{simulate}, \langle \mathsf{receive}, m\rangle, \perp\rangle\}$

Output $\mathsf{send}(m)_i$
Precondition
$local\text{-}out \neq \emptyset \wedge m = \mathbf{head}(local\text{-}out)$
Effect
$local\text{-}out \leftarrow \mathbf{tail}(local\text{-}out)$

Internal $\mathsf{init\text{-}action}(act)_i$
Precondition
$status = \mathsf{active} \wedge \mathbf{x} \in B_h \wedge \delta(val, act) \neq \perp$
Effect
$TOBcast\text{-}out \leftarrow TOBcast\text{-}out \cup \{\langle \mathsf{simulate}, act, \langle realtime, i\rangle\rangle\}$

Internal join_i
Precondition
$status = \mathsf{idle} \wedge \mathbf{x} \in B_h$
Effect
$status \leftarrow \mathsf{joining}$
$join\text{-}id \leftarrow \langle realtime, i\rangle$
$TOBcast\text{-}out \leftarrow TOBcast\text{-}out \cup \{\langle \mathsf{join\text{-}req}, \perp, join\text{-}id\rangle\}$

Internal $\mathsf{restart}_i$
Precondition
$status = \mathsf{listening} \wedge realtime = join\text{-}id.time + 2d_p + 3\epsilon$
Effect
$TOBcast\text{-}out \leftarrow TOBcast\text{-}out \cup \{\langle \mathsf{reset}\rangle\}$

Internal $\mathsf{zone\text{-}update}_i$
Precondition
$\mathbf{x} \notin B_h$
Effect
$status \leftarrow \mathsf{idle}$
$h \leftarrow$ id of zone h' such that $\mathbf{x} \in B_{h'}$
$val \leftarrow VN_h.start$
$pending\text{-}actions \leftarrow \emptyset$

Internal $\mathsf{simulate\text{-}action}(act)_i$
Precondition
$status = \mathsf{active} \wedge \mathbf{x} \in B_h$
$\mathbf{head}(pending\text{-}actions) = \langle \mathsf{simulate}, act, oid\rangle$
Effect
$\mathbf{dequeue}(pending\text{-}actions)$
if $(\langle \mathsf{simulate}, act, oid\rangle \notin completed\text{-}actions \wedge \delta(val, act) \neq \perp)$ **then**
  $val \leftarrow \delta(val, act)$
  if $act = \langle \mathsf{send}, m\rangle$ **then**
    $local\text{-}out \leftarrow local\text{-}out \cup \{m\}$
  $completed\text{-}actions \leftarrow completed\text{-}actions \cup \{\langle \mathsf{simulate}, act, oid\rangle\}$

Input $\mathsf{TOBcast\text{-}rcv}(\langle optype, param, oid\rangle)_i$
Effect
if $optype = \mathsf{simulate}$ **then**
  if $status = \mathsf{listening}$ *or* active **then**
    $\mathbf{enqueue}(pending\text{-}actions, \langle \mathsf{simulate}, param, oid\rangle)$
if $optype = \mathsf{join\text{-}req}$ **then**
  if $(status = \mathsf{joining} \wedge oid = join\text{-}id)$ **then**
    $status \leftarrow \mathsf{listening}$
  if $(status = \mathsf{active} \wedge oid \notin answered\text{-}joins \wedge \mathbf{x} \in B_h)$ **then**
    $TOBcast\text{-}out \leftarrow TOBcast\text{-}out \cup \{\langle \mathsf{join\text{-}ack}, \langle val, completed\text{-}actions\rangle, oid\rangle\}$
if $optype = \mathsf{join\text{-}ack}$ **then**
  $answered\text{-}joins \leftarrow answered\text{-}joins \cup \{oid\}$
  if $(status = \mathsf{listening}$ **and** $oid = join\text{-}id)$ **then**
    $status \leftarrow \mathsf{active}$
    $\langle val, completed\text{-}actions\rangle \leftarrow param$
if $optype = \mathsf{reset}$ **then**
  $status \leftarrow \mathsf{active}$
  $pending\text{-}actions \leftarrow \emptyset$

Output $\mathsf{TOBcast}(m)_i$
Precondition
$TOBcast\text{-}out \neq \emptyset \wedge m = \mathbf{head}(TOBcast\text{-}out)$
Effect
$TOBcast\text{-}out \leftarrow \mathbf{tail}(TOBcast\text{-}out)$

Trajectories:
Stop when any Precondition above is satisfied

Fig. 10. Transitions and trajectories of $VNE_{i,h}$ algorithm.