Using Virtual Nodes to Coordinate the Motion of Mobile Nodes

Nancy Lynch, Sayan Mitra, and Tina Nolte

***Abstract*— We describe how a virtual node abstraction layer could 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 closed 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*— Virtual nodes, Motion Coordination,...**

I. Introduction

Review our prior VN work. Cite work on motion coordination, e.g., from Bullo's [1] Morse's work [3].

We should also emphasize the modeling. All these systems should make sense in terms of HIOAs.

II. Physical System

Our physical model of the system consists of a finite but unknown number of communicating mobile 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 [4] (Figure 1): (1) automata MN_i to model mobile node with identifier $i \in \mathcal{I}$, (2) a "real world" automaton RW to model the physical location of all the mobile nodes and the real time, and (3) a $LBcast$ automaton that models the local broadcast communication service between the mobile nodes.

Figure 2 shows the components of automaton MN_i that are part of the physical layer model; it may have other arbitrary internal variables and actions that are not specified. For convenience, we assume that local actions take no time to complete. At each point each mobile node is either in active or inactive mode; initially finitely many nodes are active. The fail_i action sets the mode to inactive. In inactive mode, all internal and output actions are disabled, none of the input actions — except for the $\mathsf{recover}_i$ action which sets the mode to active — affect the internal and output variables, and the locally controlled variables remain constant during trajectories. We model the departure of a node from $\mathcal{B}$ as a failure, in which case the node also stops moving. In other words, our model excludes the possibility of an inactive node moving. The maximum speed of a mobile node is bounded by v_c.

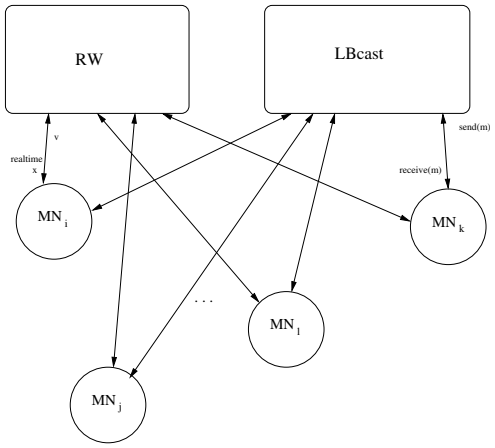

Fig. 1. Physical layer. MN automata communicate with each other through an $LBcast$ service and receive time and location updates from RW.

Signature:
Input $\mathsf{receive}(m)_i$
Input fail_i
Input $\mathsf{recover}_i$
Output $\mathsf{send}(m)_i$

State:
Input
$\mathbf{x}_i \in \mathcal{B}$
$realtime \in R^{\geq 0}$
Output
$\mathbf{v}_i \in R^2$, initially $\mathbf{0}$
Internal
$mode \in \{\mathsf{active}, \mathsf{inactive}\}$
finite set of other variables

Transitions:
Input fail_i
Effect
$\mathbf{v}_i \leftarrow 0$
$mode \leftarrow \mathsf{inactive}$
all other internal variables
set to initial values

Input $\mathsf{recover}_i$
Effect
$mode \leftarrow \mathsf{active}$

Fig. 2. Hybrid I/O Automaton MN_i

The RW automaton (see Figure 8) reads the velocity output $\mathbf{v}_i$ for each node $i \in \mathcal{I}$, and produces the position $\mathbf{x}_i$ for the MN_i and the $LBcast$ automata. RW also produces $realtime$ for all other physical layer components.

The mobile nodes 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 MN_i performs a $\mathsf{send}(m)_i$ action at some time t, the message is delivered within the interval $[t, t + d_p]$ — by a $\mathsf{receive}(m)_j$ action — to every MN_j that did not fail and was within R_p distance of i over the interval $[t, t + d_p]$.

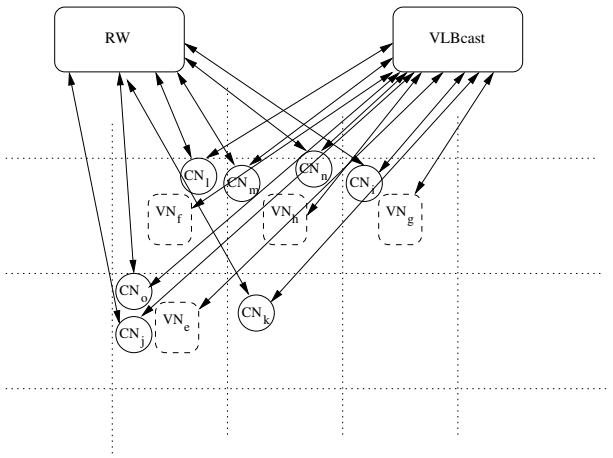

Fig. 3. Virtual Node abstraction. VNs and CNs communicate using the $VLBcast$ service.

III. Virtual System

The bounded square $\mathcal{B}$ is partitioned into a set of zones $B_h, h \in \mathcal{H}$ that is known a priori to all participating mobile nodes. For simplicity we assume this to be a square grid with sides of length b. Our virtual layer abstraction consists of stationary virtual nodes corresponding to each $h \in \mathcal{H}$, mobile client nodes, and a communication service between virtual and client nodes. Formally, the virtual layer model consists of four types of HIOA (Figure 3): (1) a VN_h automaton for each $h \in \mathcal{H}$, (2) a CN_i automaton for each mobile node with identifier $i \in \mathcal{I}$, (3) an automaton RW to model the physical location of all the client nodes and the real time, and (4) a virtual $LBcast$ communication service, $VLBcast$, for the VNs and the CNs.

Abstractly, a VN_h, $h \in \mathcal{H}$ automaton is a restricted kind of HIOA. In particular, it has no realtime clock variable and is an MMT automaton whose enabled locally controlled actions are guaranteed to occur within d' time after becoming enabled. The value of d' is further discussed in Section VI. It is located at the center $\mathbf{o}_h$ of the zone B_h and communicates with other VNs and CNs using the $VLBcast$ service through send_h and $\mathsf{receive}_h$ actions. A CN_i, $i \in \mathcal{I}$ automaton has the same external interface as a VN automaton. In addition, it has input variables $realtime$ and $\mathbf{x}_i$ from the RW automaton and an output vector velocity $\mathbf{v}_i$ to the RW automaton. For the purpose of this paper, the VN and CN automata implement a distributed motion coordination algorithm using several other internal state variables and actions (see Figures 6 and 4) which will be discussed in detail in Section V. With respect to failures, an automaton CN_i behaves the same way as MN_i. An automaton VN_h is more robust; it fails only when all the mobile nodes that are in the zone B_h fail or leave and it recovers as soon as some mobile node enters B_h.

The RW automaton in the virtual layer is the same as the one described in the physical layer. The $VLBcast$ service provides the same guarantees as the $LBcast$ service described in of the physical layer, with parameters $R_v = \sqrt{5}b$ and $d_v = 2d_p + 1$. A VN_h communicates only with its neighboring VNs and with the CNs that are in B_h.

IV. The Motion Coordination Problem

In this section we formally state the objective of our motion coordination algorithms. A differentiable parameterized curve Γ is a differentiable map $P \to \mathcal{B}$, 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 $s(a,b) = \int_a^b |\Gamma'(p)|dp$. Γ is said to be parameterized by arc length if for every $p \in P$ $|\Gamma'(p)| = 1$.

For a given point $\mathbf{x} \in \mathcal{B}$, if there exists a $p \in P$ 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 $\mathbf{x} \in \Gamma$. Since Γ is a simple curve, the inverse function Γ^{-1} is well defined. That is, if $\mathbf{x} \in \Gamma$, then $p = \Gamma^{-1}(\mathbf{x})$ is unique.

Definition 1: 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 $p_1, \ldots, p_n$, such that for each i, $1 \leq i \leq n$, $\Gamma(p_i) = \mathbf{x}_i$, and for each $1 < i < n$ $p_i - p_{i-1} = p_{i+1} - p_i$.

Consider a simple, regular, and differentiable target curve Γ parameterized by arc length, and an algorithm $\mathcal{A}$ running on N mobile nodes initially located in $\mathcal{B}$. $\mathcal{A}$ is said to solve the motion coordination problem on Γ, if after finite time the position of $N - c$ of the mobile nodes are on Γ, where c is a constant. $\mathcal{A}$ is said to solve the problem *with even spacing* if in addition, (eventually?) the position of $N - c$ of the mobile nodes are evenly spaced on Γ.

V. Solution Using Virtual Node Layer

We an algorithm that solves the motion coordination problem for target curve Γ such that most of the participating mobile nodes are eventually evenly spaced on Γ. In our algorithm each virtual node $h \in \mathcal{H}$, uses only local information about the target curve Γ. This is desirable in many practical settings, for example, where the curve is dynamically changing, or where the mobile nodes in B_h, receive periodic updates only about the nature of the curve in zone B_h from an external source. We could model this scenario by having the RW automaton communicate the local nature of Γ to the relevant mobile nodes, and our motion coordination algorithm would still be correct, however, for convenience we use a simplified model where all the mobile nodes have prior knowledge of the complete curve Γ.

Let $P_h = \{p \in P | \Gamma(p) \in B_h\}$ be the domain of Γ in zone $B_h \subset \mathcal{B}$. We assume that P_h is convex for every zone $B_h \subseteq \mathcal{B}$; it may be empty for some B_h. The length of the curve in zone B_h is $|P_h|$. The local part of the curve Γ in zone B_h is the restriction $\Gamma_h : P_h \to B_h$.

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

Signature:
Input receive$(m)_h$
Output send$(m)_h$

State:
Internal
$M : \mathcal{I} \to \mathcal{H} \times B \times R$, a partial function mapping client ids to assigned zone id, current location, last update time, initially $\emptyset$.
Accessors: *assigned*, *pos*, *ts*.
M', same type as M, initially $\emptyset$.
$V : \mathcal{H} \to Z \times R \times R$, a partial function mapping virtual node ids to current number, density, last timestamp, initially $\{\langle g, \langle 0, 0, 0\rangle\rangle\}$ for all $g \in \mathbf{N} \cup \{h\}$.
Accessors: *num*, *density*, *ts*.
$target : \mathcal{I} \to \mathcal{H} \times \mathcal{B}$, a partial function mapping client ids to assigned zone id, target location, initially $\emptyset$.
Accessors: *assigned*, *pos*.
send-buffer, a queue of messages to be sent, initially $\emptyset$
logical-time $\in R$, initially 0
next-vn-update $\in R$, initially Δ
next-cn-update $\in R$, initially $\Delta + d$
seq, a list of $\langle P, \mathcal{I}\rangle$ pairs sorted by P, initially $\emptyset$.
Accessors: *pos*, *id*.

Derived
$\mathbf{N}$ = set of ids of neighboring virtual nodes, including h
$Y = |\{i \in id(M) \ | assigned(M(i)) = h\}|$
$\mathbf{o}_g \in \mathcal{B}$, for each $g \in \mathcal{H}$, center of zone B_g
In $= \{\ g \in \mathbf{N} \ ||P_g| \neq 0\ \}$
Out $= \mathbf{N} \setminus In$
height$(g) \triangleq$ **if** $h \in$ *In* **then** *density*$(V(g))$, **else** *num*$(V(g))$
frac$(g) \triangleq$ **if** $h \in$ *In* **then** $|P_g|$, **else** 1
Low = **if** $h \in$ *In* **then** $\{g \in$ *In* $|$*height*$(g) \leq$ *height*$(g)\}$,
else $\{g \in$ *Out* $|$*height*$(g) \leq$ *height*$(g)\}$,

Transitions:
Input receive$(m)_h$
Effect
if $m = \langle$client-update, *id*, *assigned*, $\mathbf{x}$, *sent*$\rangle$ **then**
if $\mathbf{x} \in B_h$ **then**
if *logical-time* $= 0 \wedge$ *sent* > 0 **then**
next-vn-update $\leftarrow \lceil sent/\Delta \rceil \cdot \Delta$
next-cn-update $\leftarrow$ *next-vn-update* + *d*
logical-time $\leftarrow$ *max*(*logical-time*, *sent*)
if *sent* $\leq$ *next-cn-update -d* **then**
$M \leftarrow M \cup \{\langle$*id*, $\langle$*assigned*, $\mathbf{x}$, *sent*$\rangle\rangle\}$
else $M' \leftarrow M' \cup \{\langle$*id*, $\langle$*assigned*, $\mathbf{x}$, *sent*$\rangle\rangle\}$
else if $m = \langle$vn-update, *id*, *n*, *dens*, *sent*$\rangle$ **then**
if $id \in \mathbf{N}$ **then**
if *logical-time* $= 0 \wedge$ *sent* > 0 **then**
next-vn-update $\leftarrow \lceil sent/\Delta \rceil \cdot \Delta$
next-cn-update $\leftarrow$ *next-vn-update* + *d*
logical-time $\leftarrow$ *max*(*logical-time*, *sent*)
V(*id*) $\leftarrow \langle$*n*, *dens*, *sent*$\rangle$
for each $i \in id(M)$
if *ts*$(M(i)) \neq$ *next-cn-update -d* **then**
$M \leftarrow M \setminus \{\langle i, M(i)\rangle\}$
$V(h) \leftarrow \langle Y, Y/(|P_h|,$ *logical-time*$\rangle$
if *logical-time* $>$ *next-vn-update* + d_v **then**
send-buffer $\leftarrow$ *send-buffer* $\cup$
$\{\langle$vn-update, *h*, *Y*, *density*$(V(h))$, *logical-time*$\rangle\}$
next-vn-update $\leftarrow \lceil$*logical-time*$/\Delta \rceil \cdot \Delta$
if *logical-time* $>$ *next-cn-update* **then**
$M \leftarrow$ assign(M, V)
target $\leftarrow$ targetupdate(M, Γ_h)
send-buffer $\leftarrow$ *send-buffer* $\cup \{\langle$target-update, *h*, *target*$\rangle\}$
for each $g \in id(V) \cap \mathbf{N}$
$V(g) \leftarrow \langle 0, 0, 0\rangle$
$M \leftarrow M'$
$M' \leftarrow \emptyset$
next-cn-update $\leftarrow$ *next-vn-update* + *d*

Output send$(m)_h$
Precondition
nonempty(*send-buffer*) $\wedge\, m =$ *head*(*send-buffer*)
Effect
send-buffer $\leftarrow$ *tail*(*send-buffer*)

Fig. 4. VN_h IOA implementing motion coordination algorithm with parameters: safety e, update period δ, Δ, and update offset d.

Functions:
function assign$(M, V) =$
if $Y \leq e$ **then**
return M; **continue**
if $h \in Out \wedge In \neq \emptyset$
if $Y > e$
for each $g \in In$
$ra \leftarrow \lfloor (Y - e)/|In| \rfloor$
update M *by reassigning* ra *nodes from* h *to* g
else
$average \leftarrow \sum_{g \in Low} height(g)/|Low|$
$change \leftarrow \rho_2(height(h) - average)$
$nra \leftarrow min(\lfloor change \cdot frac \rfloor, Y - e)$
$sum \leftarrow \sum_{g \in Low}(height(h) - height(g))frac(g)$
if $nra > 0$
for each $g \in$ *Low*, $g \neq h$
if $h \in$ *In*
$ra \leftarrow min(\lfloor nra(height(h) - height(g))frac(g)/sum \rfloor,$
$\lfloor \frac{1}{4}(height(h) - height(g))frac(g)/frac(h) \rfloor)$
else
$ra \leftarrow \lfloor nra \cdot (height(h) - height(g))/sum \rfloor,$
update M *by reassigning* ra *nodes from* h *to* g
return M

function targetupdate$(M, \Gamma_h) =$
seq $\leftarrow \perp$; $target \leftarrow \emptyset$
for each $i \in id(M) \wedge assigned(M(i)) = h \wedge \mathbf{x}_i \in \Gamma_h$
seq $\leftarrow$ *insertsort*(*seq*, $\langle \Gamma_h^{-1}(\mathbf{x}), i\rangle$)
for each $i \in id(M)$
if $assigned(M(i)) \neq h$ **then**
let $g =$ *assigned*$(M(i))$
temp $\leftarrow \langle g, \mathbf{o}_g\rangle$
else if $\mathbf{x}_i \notin \Gamma_h$ **then**
let $\mathbf{x}^*$ **choose** $\{argmin_{\mathbf{y} \in \Gamma_h}\{dist(\mathbf{y}, \mathbf{x}_i)\}\}$
temp $\leftarrow \langle h, \mathbf{x}^*\rangle$
else let $p = \Gamma_h^{-1}(\mathbf{x})$, $seq(k) = \langle p, i\rangle$
if $k =$ *first*(*seq*) **then**
temp $\leftarrow \langle h, \Gamma_h(inf(P_h))\rangle$
else if $k =$ *last*(*seq*) **then**
temp $\leftarrow \langle h, \Gamma_h(sup(P_h))\rangle$
else
$p' \leftarrow [pos(seq(k-1)) + pos(seq(k+1))]/2$
temp $\leftarrow \langle h, \Gamma_h(p + \rho_1(p' - p))\rangle$
target $\leftarrow$ *target* $\cup \{\langle i,$ *temp*$\rangle\}$
return *target*

Fig. 5. Control functions of VN_h automaton with parameters: damping ρ_1, ρ_2, safety e.

broadcasting target waypoints for the CNs: After collecting information from clients in its zone and exchanging this information with its neighbors, a VN_h periodically sends out a target-update message which contains a target point for each client node "assigned" to zone h. Informally, each virtual node performs two tasks when setting the target points: (a) it re-assigns some of the client nodes that are assigned to itself to neighboring VNs, and (b) it sends a target position on Γ to each client node that is assigned to itself. The objective of (a) is to prevent neighboring virtual nodes from failing and to achieve a distribution of the client nodes over the zones that is proportional to the length of Γ in each zone. The objective of (b) is to space the nodes evenly on Γ in each zone. A client node, in turn, receives its current position information from RW and continuously

computes the velocity vector that would take it to its latest target point as indicated in a target-update message.

A. Virtual Node Algorithm

We define the *density* of virtual node h as $\frac{(|M_{A,h}|-e)}{(|P_h|+\epsilon)}$ where $M_{A,h}$ is the number of client nodes assigned to zone h, $e > 0$ is a parameter representing the minimal number of mobile nodes considered safe to implement a virtual node, and $\epsilon > 0$ is a small positive constant.

In designing the motion coordination algorithm we make use of synchrony in the system, that is, the common $realtime$ clock variable that is available to all client nodes. VNs do not have access to the $realtime$ variable but instead rely on second-hand clock information from messages to construct a *logical-time* clock to determine when enough time has passed to trigger required actions. Whenever a VN receives a message with a timestamp of $sent$, it adopts $max(logical-time, sent)$ as its current $logical-time$.

Intuitively, the structure of the VN algorithm (Figure 4) follows a *round* structure, where each round is of length Δ. A round is comprised of an integer number of smaller δ time periods. Each active CN_i synchronously broadcasts a client-update message every δ time through a send-vn$_i$ action. This message contains information about its id, assigned zone, current location, and the current $realtime$. At the beginning of each Δ round, each VN_h HIOA collects these messages from CNs in its region, aggregating the information and using the message timestamps to approximate the $realtime$ clock. Once a VN's $logical-time$ is larger than d_v after the start of the round, indicating that any CN's client-update message broadcast at the beginning of the round has been received, the VN_h tallies and computes the current number of nodes assigned to it and its current density and sends this information via a vn-update message to all of VN_h's neighboring zone VNs.

Once VN_h has waited until its $logical-time$ is greater than $d \geq \delta + d_v + d' + d_v$ past the beginning of the round, it knows it has collected all neighboring vn-update messages that were broadcast for the round: a neighboring VN might be delayed in sending its vn-update for the round until the next δ client-update occurs, when it is guaranteed to receive a newer message timestamp. It can then take an additional d_v time for the message to arrive, d' time for the resulting triggered vn-update to be executed, and an extra d_v time for the vn-update to be received at VN_h.

VN_h then uses the node count and density information from its neighbors to calculate how many of the CNs assigned to its zone should be reassigned and to which neighboring zones (through the assign function, Figure 5). This information is then used to calculate new target waypoints for each CN in the zone (through the targetupdate function, Figure 5). VN_h broadcasts the new target waypoints through a target-update message to its zone's CNS. Any CN in the region receiving this broadcast looks in the message to determine its new target waypoint and starts moving towards the new waypoint. Given a speed of v_c for each client mobile node and the maximum distance from a point in one region to a point in a neighboring region when following a target waypoint at the center of the neighboring zone, it will take up to $d_v + \frac{\sqrt{10/9}b}{v_c}$ time for any client node reassigned to a new zone to discover it has been reassigned and to move into that new zone. In addition, the target zone might not yet have a virtual node. In this case, the client must restart the virtual node, taking up to a known d_r time (explained further in Section VI).

For simplicity, we assume that the Δ time period is large enough for a reassigned client node to arrive in its new zone and restart the virtual node (if it is empty) before a new round begins. This introduces the restriction that the integer number of δ time periods comprising Δ be at least $\lceil \frac{d+d_v+(\sqrt{10/9}b/v_c)+d_r}{\delta} \rceil$. 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.

[[doesn't match code: VN_h may perform a classified$_h$ action d time after every vn-update message is sent. This action computes the reassignment of client nodes that are assigned to h. A classified$_h$ action occurs only if the number of client nodes assigned to h, $|M_{A,h}|$ exceeds the safety critical number e, which is a parameter of the algorithm, and one of the following conditions hold: (1) There exists a set N_h^0 of VNs in the neighborhood of h, each of which has fewer than e assigned client nodes. In this case, some of the extra $|M_{A,h}| - e$ nodes that are currently assigned to h are reassigned to the VNs in this set N_h^0. (2) None of the neighboring VNs of h have fewer than e CNs, but there exists a set N_h^- of neighboring VNs that have a much lower client node density than h. In this case, h lowers its density by a factor of ρ_2, by reassigning $(1-\rho_2).|M_h^h|$ CNs that are currently assigned to itself (making sure that $|M_h^h|$ does not fall below e). In either case, the reassignment of the CNs currently assigned to h to the VN g in N_h^0 (or in N_h^-) is in proportion to $|density(g)|$ (or $|density(h) - density(g)|$). The output of classified$_h$ is the partial function M that maps CN identifiers to VNs that they are assigned to.

The classified$_h$ action triggers the set-target$_h$ action, which assigns to every client node a target point in $\mathcal{B}$ to move to. This action first computes seq, a sequence of pairs of type $(P_h, \mathcal{I})$. For every CN node i in M_h, if i is assigned to h and $\mathbf{x}_i$ is on the curve Γ_h, i.e., $\Gamma_h(p) = \mathbf{x}_i$ for some $p \in P_h$, then a pair (p, i) is inserted in seq. The list seq is sorted on p. The target point for CN i is computed as follows: If i is not assigned to h then its target is the center of the VN it is assigned to; if it is not on the curve Γ_h then it is directed to the nearest point on the curve (nondeterministically choosing amongst the set of nearest points); if $\mathbf{x}_i$ is already on the curve and is one of the end points of Γ_h, then it is not moved; if $\mathbf{x}_i$ is on the curve but is not an end point, then it is moved to the mid-point of the

Signature:
  Input receive$(m)_i$
  Output send$(m)_i$
  Internal send-vn$_i$

State:
  Input
    $\mathbf{x}_i \in \mathcal{B}$
    $realtime \in R^{\geq 0}$
  Internal
    $\mathbf{x}^* \in \mathcal{B}$, target point, initially same as $\mathbf{x}$
    next-update $\in R$, initially $\lceil realtime/\delta \rceil \cdot \delta$
    send-buffer, queue of messages, initially $\emptyset$
    assigned $\in \mathcal{H}$, initially $h \in H$ **such that** $\mathbf{x}_i \in B_h$
  Output
    $\mathbf{v}_i \in R^2$, velocity vector, initially $null$

Transitions:
  Internal send-vn$_i$
   Precondition
    realtime $=$ *next-update*
   Effect
    send-buffer $\leftarrow$ *send-buffer* $\cup$
      $\{\langle$client-update, *i*, *assigned*, $\mathbf{x}_i$, *realtime*$\rangle\}$
    next-update $\leftarrow$ *next-update* + δ

  Output send$(m)_i$
  Precondition
    nonempty(*send-buffer*)
    $m =$ *head*(*send-buffer*)
  Effect
    send-buffer $\leftarrow$ *tail*(*send-buffer*)

  Input
   receive$(m)_i$
   Effect
    if $m = \langle$target-update, *h*, *target*$\rangle$ **then**
      if *assigned* $=$ *h* **then**
        assigned $\leftarrow$ *assigned*(*target*(*i*))
        $\mathbf{x}^* \leftarrow pos(target(i))$

Trajectories:
  Evolve
    if $\mathbf{x}_i = \mathbf{x}^*$ **then** $\mathbf{v}_i = \mathbf{0}$
    else if $\mathbf{x}_i \notin \Gamma$ *or* $\mathbf{x}^* \notin \Gamma$ **then**
      $\mathbf{v}_i = v_c.(\mathbf{x}^* - \mathbf{x}_i)/||\mathbf{x}^* - \mathbf{x}||$
    else let $\mathbf{x}_i = \Gamma(p), \mathbf{x}^* = \Gamma(p^*)$
      if $p < p^*$ **then** $\mathbf{v}_i = v_c.\Gamma'(p)$
      else $\mathbf{v}_i = -v_c.\Gamma'(p)$
  Stop when
    next-update $=$ *realtime*

Fig. 6. CN_i automaton with update parameter δ

preceding and the succeeding CNs on the curve. For this last computation the sorted list seq is used.]]

B. Client Node Algorithm

Here we describe the client node CN HIOA (Figure 6) with parameter δ. The CN automaton periodically (every δ time) sends a client-update message to update the VN it is located in, informing it of its id, current location in $\mathcal{B}$, assigned VN, and current $realtime$ value. It processes only one type of message, namely the target-update messages sent by the VN to which it is currently assigned. The contents of this message determine the new target location $\mathbf{x}_i^*$ for CN_i, and possibly, an assignment to a different VN. Client node i computes its velocity vector $\mathbf{v}_i$, based on its current position $\mathbf{x}_i$ and its target position $\mathbf{x}_i^*$, as follows: If either $\mathbf{x}_i$ or $\mathbf{x}_i^*$ are not on Γ, then it moves in a straight line with constant velocity towards $\mathbf{x}_i^*$; otherwise, it moves along Γ with constant speed, its velocity vector being determined by the tangent on Γ at $\mathbf{x}_i$.

C. Correctness and Performance

As described before, our motion coordination algorithm has two control "processes" implemented by the two functions assign and targetupdate of each VN. The first process (1) moves all except e CNs from the virtual nodes in Out, to the virtual nodes in In, and (2) moves the CNs that are already in the VNs in In, to achieve a distribution of CNs that is proportion to the length of Γ in each VN. This process of the algorithm is said to converge in round t if (1) the set of CNs in each virtual nodes in $h \in In$, $id(M_h(t))$, remain constant at every round after t. The second process of the algorithm is executed only by the VNs in In, and it moves the CNs within each VN to position them evenly along the portion of the curve Γ in B_h. This process converges in round t if the position of all CNs remain unchanged in all successive rounds.

The convergence of the first process is necessary for the convergence of the second. In our analysis, we show that if there are no failures or recoveries in the system after a certain round, then the first process converges in a finite number of rounds, and the second process converges eventually, after the first process has converged. For the convergence of the first process, we assume that a virtual node may be assigned a fraction of a client node and therefore ignore the $\lfloor \cdot \rfloor$ operators in the pseudocode of Figure 5. This simplifies the analysis as well as the presentation. With the integrality constrains in place, convergence is still guaranteed, although with a slightly different criterion. The details of this will be presented in a full version of this paper. We define the following quantities at round t,

$$
\begin{aligned}
In(t) &\triangleq \{h \in \mathcal{H} \mid |P_h| > 0\} \\
Out(t) &\triangleq \mathcal{H} \setminus In(t) \\
Border(t) &\triangleq \{h \in Out(t) \mid \exists g \in \mathbf{N}_h, g \in In(t)\} \\
C_{in}(t) &\triangleq \{i \in \mathcal{I} \mid i \in id(M_h(t)), h \in In(t)\} \\
C_{out}(t) &\triangleq \{i \in \mathcal{I} \mid i \in id(M_h(t)), h \in Out(t)\} \\
C(t) &\triangleq C_{in}(t) \cup C_{out}(t) \\
P(t) &\triangleq \{P_h \mid h \in In(t)\} \\
peak(t) &\triangleq MAX_{h \in In(t)} density(h)(t), \\
peaksize(t) &\triangleq \{P_h \mid density(h) = max(t)\}
\end{aligned}
$$

In, Out, and $Border$, are the sets of VNs that are alive (have some mobile node in their respective zones) at round t. C_{in} (resp. C_{out}) is the set of CNs that are alive at round t and are assigned to some VN in In (resp. Out). $P(t)$ is the length of the curve Γ that goes through zones of live

VNs at round t. $peak$ is the maximum density of any VN in In, and $peaksize$ is the length of Γ in peak zones.

Lemma 1: If there are no failures after round t, then for all $t' \geq t$, $In(t) \subseteq In(t'), Out(t) \subseteq Out(t')$, $C_{in}(t') \geq C_{in}(t)$, $C_{out}(t') \leq C_{out}(t')$, and $Y_h(t) \geq e$ for some $h \in \mathcal{H}$, implies $Y_h(t') \geq e$.

Lemma 2: If client node i is in some virtual node $h \in In(t)$ at round t, then in all successive round $t' \geq t$, $i \in id(M_g^g(t'))$ for some virtual node in $g \in In(t')$.

Lemma 3: At any round t, $peak(t) = C_{in}(t)/|P(t)|$ iff $peaksize(t) = P(t)$.

Proof: Part(a): Let $peak(t) = C_{in}(t)/|P(t)|$. Suppose, for the sake of contradiction, that $peaksize(t) \subset P(t)$. There is a non-zero subset of intervals, $low(t) = P(t) \setminus peaksize(t)$, such that for every $P_g \in low(t)$, $density(g)(t) < max(t)$. Then, $peak(t) \cdot |peaksize(t)| + \sum_{P_g \in low(t)} density(g)(t) \cdot |P_g| = C_{in}(t)$. LHS is strictly less than $peak(t)(|peaksize(t)| + \sum |P_g|)$, which leads to a contradiction. The other direction is trivial. ■

Lemma 4: If there are no failures after round t, then there exists a round $T \geq t$, such that for all $t' \geq T$, and for all $h \in In$, $M_h^h(t') = M_h^h(T)$.

Proof: We claim that the above result follows if we show that there exists a round T such that in every successive round t', for every $h \in In(t')$, the number of client nodes assigned to h, $Y_h(t)$ remains unchanged . From lines 12–13 we see that the number of CNs reassigned by VN h depends only on the $density$ of the VNs with strictly lower density, in the neighborhood of h. The densities remain constant because by our assumption Y remains constant. Therefore, the same number of CNs are reassigned by each VN in all successive rounds. Suppose some VN in In reassigns non-zero client nodes. As there cannot be a cycle in the density ordering of the VNs and CNs (from Lemma 2) cannot leave the set of VNs in In, the density of some VN in In must be increasing, which contradicts our previous assumption. So, the number of reassignments in every successive round by every VN in In, must be zero. Which implies that the set of CNs assigned to each VN remains unchanged.

Now we show that indeed there exists a round T such that in every successive round t', for every $h \in In(t')$, $Y_h(t)$ remains unchanged. First, we consider the energy function at round t, $E(t) = (C_{out}(t), peak(t), peaksize(t))$. Observe that if there are no failures, then in any round t, $E(t)$ is at least $E_0(t) = (e|Out(t)|, (|C(t)| - e|Out(t)|)/|P(t)|, |P(t)|)$. If there are no failures after round t_0, then in any round $t \geq t_0$, the minimum value $E_0(t)$ is bounded from below by $(e|Out(t_0)|, (|C(t_0)| - e|Out(t_0)|)/|P|, |P|)$. If this minimum value $E_0(t)$ is achieved in some round t, then $P(t) = P$ and every VN h in $In(t)$ has $density(h)(t) = peak(t) = C_{in}(t)$. That is, in every successive round $Y_h(t)$ remains unchanged. We show that at any round t, if every $E(t) > E_0(t)$, then $E(t+1) < E(t)$.

By Lemma 1, $C_{out}(t+1) \leq C_{out}(t)$. If $C_{out}(t+1) = C_{out}(t)$, then $C_{in}(t+1) = C_{in}(t)$. We claim that $peak(t+1) \leq peak(t)$. If this is not true, then there exists a VN $h \in In(t)$, such that $density(h)(t+1) > peak(t)$. If must be that $density(h)(t) < peak(t)$, otherwise h would not have been assigned any CNs by its neighbors (see line 12) in round t. $density(h)(t+1) - density(h)(t) \leq 4\rho_2(peak(t) - density(h)(t))$.

3(a). ■

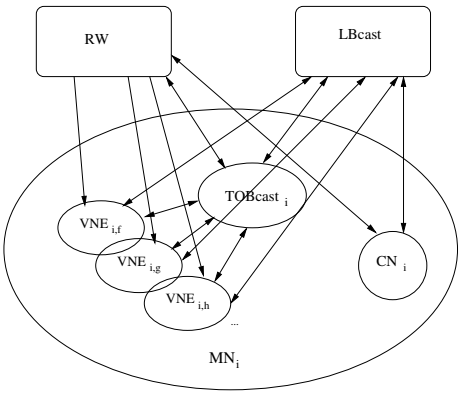

Fig. 7. MN_i's subautomata: A mobile node runs several programs, including VNE and TOBcast automata as well as a CN automaton.

Claim 1: If there exists a round T_{in} such that for every $t \geq T_{in}$, $M_h^h(t) = M_h^h(T_{in})$ and there are no failures, then the locations of the CNs in B_h are eventually evenly spaced on Γ_h.

Proof: (Sketch) Let t be the time when the assignments "stabilize". Then, within $\frac{\sqrt{2}b}{v_c} + O(\delta)$ time units after t, every CN in B_h is assigned to VN h, and every CN that is assigned to VN h is in B_h. Within another $\frac{2\sqrt{2}b}{v_c} + O(\delta)$ time units, every CN is on Γ and the endpoints are fixed at $\Gamma(inf(P_h))$ and $\Gamma(sup(P_h))$. At this point suppose $seq_h = \langle p_0, i_{(0)} \rangle, \ldots, \langle p_{n+1}, i_{(n+1)} \rangle$. We can show that (if nodes do not fail and if no new nodes enter B_h) then eventually the $p's$ in seq are equidistant. ■

VI. Implementing the Virtual Node Layer

In this section we present a sketch of our implementation of the virtual layer. Our implementation is an adaptation of techniques from [2] to emulate a virtual mobile node. The only substantive changes made in our current implementation are in (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 client node discovers it is in a failed virtual node's zone.

In addition to its client CN_i, a real mobile node i in zone B_h runs a $TOBcast_i$ service and a $VNE_{i,h}$ algorithm (Figure 7) to help implement each virtual node VN_h and the $VLBcast$ service of the virtual layer. We use a standard replicated state machine approach to implement

robust virtual nodes that takes advantage of the $TOBcast$ service to ensure that all VNEs in a region receive the same messages in the same order. Using the $LBcast$ service of the real nodes and common knowledge about $realtime$, a totally ordered broadcast service, $TOBcast$ for each B_h can be implemented as follows: At the time of sending each message is tagged with a timestamp which is the current value of $realtime$ and the sender's identifier. Assuming that a real node does not make multiple broadcasts at the same point in time, the tags define a total order on the sent messages. Before delivering a message a node waits until $d_p + 1$ time has elapsed since it was sent, ensuring that all earlier messages were received.

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 mobile 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 includes broadcasting suggestions to receive messages on behalf of the VN that were actually received by $VNE_{i,h}$. When an action suggestion is received by $VNE_{i,h}$, it is saved in a $pending - action$ queue. Actions are removed from a $pending - action$ 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 $completed - action$ queue. This queue is referenced by $VNE_{i,h}$ to prevent reprocessing of completed actions. The $TOBcast$ necessary for each simulated action introduces the requirement that $d' \geq d_p + 1$, meaning that enabled actions of an emulated VN will occur after the time necessary for the transmission of the action to be completed. Given that mobile node actions are assumed to take no time, we can guarantee that $d' = d_p + 1$.

When a mobile node enters a zone, it executes a join protocol to get the state of the zone's VN. The join protocol begins by using $TOBcast$ to send a join-req message. Whenever a node receives its own join-req message, it starts saving messages to process in its $pending - action$ queue. If a node 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 node receives the join-ack, it copies the included VN state and starts processing the actions in its $pending - action$ queue. If no mobile nodes are in a region to emulate the VN, the VN fails. If a real node enters a failed region its join-req will not be answered in $2d_p + 3$ time, and the real node will reset the VN by using $TOBcast$ to send a reset message. When a node receives a reset message $d_p + 1$ time later, it restarts the VN state at its initial state, clears the $pending - action$ queue, and starts simulating the VN as before. This total time of $3d_p + 4$ time required for a joining node to succeed in restarting a VN is the constant d_r.

VII. Putting the Pieces Together

[[Nancy: Say something about what it all looks like when it's combined. What actually runs on each PN. A piece representing the client node and a piece doing the emulation. Does it all fit together right? It's kind of mind-boggling.]]

VIII. Conclusions

What have we done? What have we learned?

Future work/extensions: We can consider a problem extension where the curve is moving. 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 mobile 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.

Probabilistic guarantees for VN survival: Given a probability of mobile node failure p in a span of Δ time, we can say with high probability that a zone's VN remains alive; would be based on an expression for the number e of mobile nodes considered the minimum for safe VN replication.

References

[1] 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.

[2] 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.

[3] 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.

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

APPENDIX

State:
  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. 8. RW automaton

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

State:
  Input
    $\mathbf{x}_i \in \mathcal{B}$, current location of mobile node
    realtime $\in R^{\geq 0}$
  Internal
    status $\in$ {joining, listening, 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 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. Signatures and States of VNE$_{i,h}$ algorithm implementing VN_h

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

Output send$(m)_{h,i}$
Precondition:
  $m \in$ *local-out*
Effect:
  local-out $\leftarrow$ *local-out* / $\{m\}$

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

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

Input reset$_i$
Effect:
  TOBcast-out $\leftarrow$ *TOBcast-out* $\cup$ $\{\langle$reset$\rangle\}$

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

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

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

Input TOBcast-rcv($\langle$*optype*, *param*, *oid*$\rangle$)$_i$
Effect:
  if (*optype* $=$ simulate) **then**
    if (*status* $\neq$ listening *or* active) **then continue**;
    else Enqueue(*pending-actions*, $\langle$simulate, *param*, *oid*$\rangle$)
  else if (*optype* $=$ join-req) **then**
    if ((*status* $=$ joining) **and** (*oid* $=$ *join-id*))
      then *status* $\leftarrow$ listening
    if ((*status* $=$ active) **then**
      if (*oid* $\in$ *answered-joins*)) **then continue**;
      else if ($\mathbf{x} \in B_h$)
        then TOBcast($\langle$join-ack, $\langle$*val*, *completed-actions*$\rangle$, *oid*$\rangle$)
  else if (*optype* $=$ join-ack)
    answered-joins $\leftarrow$ *answered-joins* $\cup \{oid\}$
    if (*status* $=$ listening) **then**
      if (*oid* $=$ *join-id*)) **then**
        status $\leftarrow$ active
        $\langle$*val*, *completed-actions*$\rangle \leftarrow$ *param*
  else if (*optype* $=$ reset) **then**
    status $\leftarrow$ active; *pending-actions* $\leftarrow \emptyset$

Output TOBcast$(m)_i$
Precondition:
  $m \in$ *TOBcast-out*
Effect:
  TOBcast-out $\leftarrow$ *TOBcast-out* / $\{m\}$

Trajectories:
  Stop when *any precondition above is satisfied*

Fig. 10. Transitions of VNE$_{i,h}$ algorithm