Self-Stabilizing Mobile Node Location Management and Message Routing*

Shlomi Dolev[1], Limor Lahiani[1], Nancy Lynch[2], and Tina Nolte[2]

[1] Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, 84105, Israel
[2] MIT CSAIL, Cambridge, MA 02139, USA

Abstract. We present simple algorithms for achieving self-stabilizing location management and routing in mobile ad-hoc networks. While mobile clients may be susceptible to corruption and stopping failures, mobile networks are often deployed with a reliable *GPS oracle*, supplying frequent updates of accurate real time and location information to mobile nodes. Information from a GPS oracle provides an external, shared source of consistency for mobile nodes, allowing them to label and timestamp messages, and hence aiding in identification of, and eventual recovery from, corruption and failures. Our algorithms use a GPS oracle.
Our algorithms also take advantage of the *Virtual Stationary Automata* programming abstraction, consisting of mobile clients, virtual timed machines called virtual stationary automata (VSAs), and a local broadcast service connecting VSAs and mobile clients. VSAs are distributed at known locations over the plane, and emulated in a self-stabilizing manner by the mobile nodes in the system. They serve as fault-tolerant building blocks that can interact with mobile clients and each other, and can simplify implementations of services in mobile networks.
We implement three self-stabilizing, fault-tolerant services, each built on the prior services: (1) VSA-to-VSA geographic routing, (2) mobile client location management, and (3) mobile client end-to-end routing. We use a greedy version of the classical depth-first search algorithm to route messages between VSAs in different regions. The mobile client location management service is based on *home locations*: Each client identifier hashes to a set of home locations, regions whose VSAs are periodically updated with the client's location. VSAs maintain this information and answer queries for client locations. Finally, the VSA-to-VSA routing and location management services are used to implement mobile client end-to-end routing.

Keywords: virtual infrastructure, location management, home locations, end-to-end routing, hash functions, self-stabilization, GPS oracle

* Longer version available as MIT LCS Technical Report MIT-LCS-TR-999.

[1] Partially supported by IBM faculty award, NSF grant and the Israeli ministry of defense. Email: {dolev, lahiani}@cs.bgu.ac.il.

[2] Supported by DARPA contract F33615-01-C-1896, NSF ITR contract CCR-0121277, and USAF, AFRL contract FA9550-04-1-0121. Email: {lynch, tnolte}@theory.csail.mit.edu.

1 Introduction

A system with no fixed infrastructure in which mobile clients may wander in the plane and assist each other in forwarding messages is called an ad-hoc network. The task of designing algorithms for constantly changing networks is difficult. Highly dynamic networks, however, are becoming increasingly prevalent, and it is therefore important to develop and use techniques that simplify this task. In addition, mobile nodes in these networks may suffer from crash failures or corruption faults, which cause arbitrary changes to their program states. Self-stabilization [4, 5] is the ability to recover from an arbitrarily corrupt state. This property is important in long-lived, chaotic systems where certain events can result in unpredictable faults. For example, transient interference may disrupt wireless communication, violating our assumptions about the broadcast medium.

Mobile networks are often deployed with "reliable" GPS services, supplying frequent updates of real time and region information to mobile nodes. While the mobile clients may be susceptible to corruption and stopping failures, the GPS service may not be. Each of our algorithms utilizes such a reliable *GPS oracle*. Information from this oracle provides an external, shared source of consistency for nodes, and aids in identification of, and recovery from, failures.

In this paper we describe self-stabilizing algorithms that use a reliable GPS oracle to provide geographic routing, a mobile client location management service, and a mobile client end-to-end routing service. Each service is built on the prior services such that the composition of the services remains self-stabilizing [11]. To simplify the service implementations, we mask the unpredictability of mobile nodes by using a self-stabilizing *virtual* infrastructure, consisting of mobile clients, timing-aware and location-aware machines at fixed locations, called *Virtual Stationary Automata* (VSAs) [8, 9], that mobile clients can interact with, and a local broadcast service connecting VSAs and mobile clients.

Self-stabilization and *GPS oracles*. Traditionally, self-stabilizing systems are those systems that can be started from arbitrary configurations and eventually regain consistency *without external help*. However, mobile clients often have access to some reliable external information from a service such as GPS. Each algorithm in this paper uses an external GPS service (or an equivalent) as a reliable *GPS oracle*, providing periodic time and location updates, to base stabilization upon; our algorithms use timestamps and location information to tag events. In an arbitrary state, recorded events may have corrupted timestamps. Corrupted timestamps indicating future times can be identified and reset to predefined values; new events receive newer timestamps than any in the arbitrary initial state. This eventually allows nodes in the system to totally order events. We use the eventual total order to provide consistency of information and distinguish between incarnations of activity (such as retransmissions of messages).

Virtual Stationary Automata programming layer. In prior work [8, 7, 6], we 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 physical nodes. The GeoQuorums algorithm [7] proposes storing data at fixed locations; however it only supports atomic objects,

rather than general automata. A more general virtual mobile automaton is suggested in [6]. Finally, the virtual automata presented in [8, 9] (and used here) are more powerful than those of [6], providing timing capabilities.

The static infrastructure we use in this paper includes virtual machines with an explicit notion of real time, called *Virtual Stationary Automata* (VSAs), distributed at known locations [8, 9]. Each VSA represents a predetermined geographic area and has broadcast capabilities similar to those of the mobile nodes, allowing nearby VSAs and mobile nodes to communicate with one another. Many algorithms depend significantly on timing, and many mobile nodes have access to reasonably synchronized clocks. In the VSA layer, VSAs also have access to *virtual* clocks, guaranteed to not drift too far from real time. The layer provides mobile nodes with a fixed virtual infrastructure, reminiscent of better understood wired networks, with which to coordinate. An important property of the VSA layer implementation described in [8, 9] is that it is self-stabilizing. Corruptions at physical nodes can result in inconsistency in the emulation of a VSA. However, emulations recover after corruptions to correctly emulate a VSA.

Geographic/VSA-to-VSA routing. A basic service running on the VSA layer that we describe and use repeatedly is that of VSA-to-VSA (or region-to-region) routing (VtoVComm), providing a form of geocast. GeoCast algorithms [24, 3], GOAFR [19], and algorithms for "routing on a curve" [23] route messages based on the location of the source and destination, using geography to delivery messages efficiently. GPSR [17], AFR [20], GOAFR+ [19], polygonal broadcast [10], and the asymptotically optimal algorithm [20] are algorithms based on greedy geographic routing algorithms, forwarding messages to the neighbor that is geographically closest to the destination. The algorithms also address "local minimum situations", where the greedy decision cannot be made. GPSR, GOAFR+, and AFR achieve, under reasonable network behavior, a linear order expected cost in the distance between the sender and the receiver. We implement VSA-to-VSA routing using a persistent greedy depth-first search (DFS) routing algorithm that runs on top of the VSA layer's fixed infrastructure. Our scheme is an application of the classical DFS algorithm in a new setting.

Location management. Finding the location of a moving client in an ad-hoc network is difficult, much more so than in cellular networks where a fixed infrastructure of wired support stations exist (as in [16]), or sensor networks where some approximation of fixed infrastructure may exist [2]. A *location service* is a service that allows any client to discover the location of any other client using only its identifier. The paradigm for location services that we use here is that of a home location service: Hosts called *home location servers* are responsible for storing and maintaining the location of other hosts in the network [1, 14, 21]. Several ways to determine home location servers have been suggested.

The locality aware location service in [1] for ad-hoc networks is based on a hierarchy of lattice points for destination nodes, published with locations of associated nodes. Lattice points can be queried for the desired location, with a query traversing a spiral path of lattice nodes increasingly distant from the source until it reaches the destination. Another way of choosing location servers

is based on quorums. A set of hosts is chosen to be a *write* quorum for a mobile client and is updated with the client's location. Another set is chosen to be a *read* quorum and queried for the desired client location. Each *write* and *read* quorum has a nonempty intersection, guaranteeing that if a *read* quorum is queried, the results will include the latest location written to a *write* quorum. In [14], a uniform quorum system is suggested, based on a virtual backbone of quorum representatives. Geographic quorums based on focal points are suggested in [7].

Location servers can also be chosen using a hash table. Some papers [21, 15, 25] use geographic locations as a repository for data. These use a hash to associate each piece of data with a region of the network and store the data at nodes in the region. This data can be used for routing or other applications. The Grid location service (GLS) [21] maps client ids to geographic coordinates. A client C_p's location is saved by clients closest to the coordinates p hashes to.

The scheme we present is based on hash tables and built on top of the VSA layer and VSA-to-VSA routing service. VSAs and mobile clients are programmed to form a self-stabilizing distributed data structure, where VSAs serve as home locations for clients. Each client's id hashes to a VSA region, the client's home location, whose VSA is responsible for maintaining the location of the client. To tolerate crashes of a limited number of VSAs, each mobile client id actually maps to a set of VSA home locations; the hash function returns a sequence of region ids as the home locations. We can use any hash function that provides a sequence of regions; one possibility is a *permutation hash function*, where permutations of region ids are lexicographically ordered and indexed by client id.

End-to-end routing. Another important service in mobile networks is end-to-end routing. Our self-stabilizing implementation of a mobile client end-to-end communication service is simple, given VSA-to-VSA routing and the home location service. A client sends a message to another client by using the home location service to discover the destination client's region and then has a local VSA forward the message to the region using the VSA-to-VSA service.

2 Datatypes and system model

We assume the *Virtual Stationary Automata* programming abstraction [8], which includes mobile client nodes and the virtual stationary automata (VSAs) the mobile nodes emulate, as well as a local broadcast service, V-bcast, between them (see Figure 1). The network is a fixed, closed, and bounded connected region R of the 2-D plane. R is partitioned into known connected subregions called *regions*, with unique ids from the set of region ids U. We define a neighbor relation $nbrs$ on ids from U. This relation holds for any two regions u and v where the supremum distance between points in u and v is bounded by a constant r_{virt}.

2.1 Client nodes

For each physical node identifier p from P, we assume a mobile timed I/O automaton client C_p, whose location in R at any time is referred to as $loc(p)$. Mobile client speed is bounded by a constant v_{max}. Clients receive region and time information from the GPS oracle. A $\mathsf{GPSupdate}(u, now)_p$ happens every

ϵ_{sample} time at each client, indicating to the client the region u where it is located and the current time *now*. Clients accept *now* as the value of their own local clock. For simplicity, this local variable progresses at the rate of real time.

Each client C_p is equipped with a local broadcast service V-bcast (see Section 2.3), allowing it to communicate with nearby VSAs and clients with $\mathsf{bcast}(m)_p$ and $\mathsf{brcv}(m)_p$. Clients are susceptible to stopping and corruption failures. After a stopping failure, a client performs no additional local steps until restarted. If restarted, it starts again from an initial state. If a node suffers a corruption, it experiences a nondeterministic change to its program state. Additional interface actions and local state at the client are allowed. Local steps take no time.

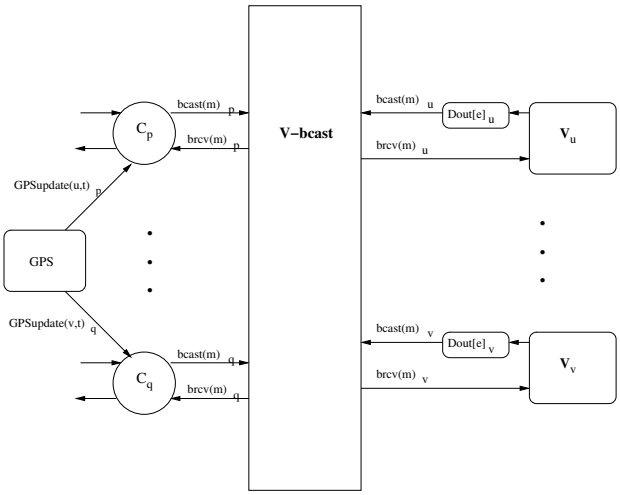


Fig. 1. VSA layer. VSAs and clients communicate with V-bcast. VSA outputs may be delayed in Dout.

2.2 Virtual Stationary Automata (VSAs)

A self-stabilizing implementation of VSAs using a GPS oracle and physical mobile nodes can be found in [8, 9]. An abstract VSA is a timing-enabled virtual machine that may be emulated by the mobile nodes in its region in the network. A VSA for region u, V_u, is a TIOA whose program is a tuple of its action signature, sig_u, valid states, $states_u$, a start state function mapping clock values to start states, $start_u$, a discrete transition function, δ_u, and a set of valid trajectories [18], τ_u. The state of V_u is referred to collectively as $vstate$ and is assumed to include a variable corresponding to real time, $vstate.now$. To guarantee we can emulate a VSA using mobile nodes, its interface must be emulatable by the nodes; a VSA V_u's external interface is restricted, including only stopping, corruption, and restart inputs, and the ability to broadcast and receive messages.

Since a VSA is emulated by physical nodes in its region, its failures are defined in terms of client failures: (1) If no clients are in the region, the VSA is crashed, (2) If no client failure occurs in an alive VSA's region over some interval, the VSA does not suffer a failure during that interval, and (3) A VSA may suffer a corruption only if a mobile client in its region suffers a corruption; the self-stabilizing implementation of a VSA in [8, 9] guarantees that starting

from an arbitrary configuration, the emulation's external trace will eventually look like that of the abstract VSA, starting from a corrupted abstract state.

Due to message delays or node failure, emulations might be behind real time by up to some time e. It is then a *delay-augmented TIOA*, an augmentation of V_u with timing perturbations, represented with buffers $\text{Dout}[e]_u$. The buffer delays messages by a nondeterministic time $[0, e]$, where e is more than V-bcast's broadcast delay, d (see Section 2.3).

2.3 Local broadcast service (V-bcast)

Communication is in the form of local broadcast V-bcast, with broadcast radius r_{virt} and message delay d. It allows communication between VSAs and clients in the same or neighboring regions. The service allows the broadcasting and receiving of message m at each port $i \in P \cup U$ through $\mathsf{bcast}(m)_i$ and $\mathsf{brcv}(m)_i$.

We assume V-bcast guarantees two properties: integrity and reliable local delivery. *Integrity* guarantees for any $\mathsf{brcv}(m)_i$ that occurs, a $\mathsf{bcast}(m)_j, j \in P \cup U$ previously occurred. *Reliable local delivery* roughly guarantees a transmission will be received by nearby ports: If port i in region u transmits a message, then every port j in region u or neighboring regions during the entire time interval starting at transmission and ending d later receives the message by the end of the interval. (Here, due to GPSupdate lag, a client is said to be "in" region u if it has just left u but has not yet received a GPSupdate with the change.)

We assume broadcast buffers are large enough that overflows do not occur in normal operation. In the event of overflow, overflow messages are lost.

3 Problem specifications

We describe the services we will build over the VSA layer: VSA-to-VSA routing, a location service, and client-to-client routing, and describe our requirement that implementations be self-stabilizing.

The following constants (explained/used shortly) are globally known: (1) $f < |U|$, a limit on "home location" VSA failures for a client, (2) h, a function mapping each client id to a sequence of $f+1$ distinct region ids, (3) $ttl_{VtoV} > d$, delivery time for the VtoVComm service, (4) $ttl_{HLS} \geq \epsilon_{sample} + 2d + 3e + 2ttl_{VtoV}$, response time of the location management service, and (5) ttl_{hb}, a refresh period. We assume the following client mobility and VSA crash failure conditions:
(1) Each client spends at least ϵ_{sample} time in a region before moving to another,
(2) At any time, each alive client's current region or a neighboring region has a non-crashed VSA that remains alive for an additional ttl_{HLS} time,
(3) For any interval of length $ttl_{VtoV} + e$, VSAs alive over the interval are connected via at least one path of non-crashed VSAs over the whole interval, and
(4) For any interval of length $ttl_{hb} + 2ttl_{VtoV} + 2e + d$, and any alive client q, at least one VSA from $h(q)$ does not crash during the interval.

3.1 VSA-to-VSA communication service (VtoVComm) specification

The first service is an inter-VSA routing service, where a VSA from some region u can send a message m through $\mathsf{VtoVsend}(v, m)_u$ to a VSA in another (potentially non-neighboring) region v. Region v's VSA later receives m through

$\mathsf{VtoVrcv}(m)_v$. The service guarantees two properties:
(1) If a VSA at region u performs a $\mathsf{VtoVsend}(v, m)$, and region u and v VSAs are alive over the interval beginning with the send and ending ttl_{VtoV} later, then the VSA at region v performs a $\mathsf{VtoVrcv}(m)$ before the end of the interval, and
(2) If a message is received at some VSA, it was previously sent to that VSA.

3.2 Location service specification

A location service answers queries from clients for the locations of other clients. A client node p can submit a query for a recent region of client node q via a $\mathsf{HLquery}(q)_p$ action. If q has been in the system for a sufficient amount of time, the service responds within bounded time with a recent region location of q, $qreg$, through a $\mathsf{HLreply}(q, qreg)_i$ action. More precisely, the service guarantees that if a client p performs a $\mathsf{HLquery}$ to find an alive client q that has been in the system longer than $\epsilon_{sample} + d + ttl_{VtoV} + e + ttl_{HLS}$ time, and p does not crash or change regions for ttl_{HLS} time, then:
(1) Within ttl_{HLS} time, client p will perform a $\mathsf{HLreply}$ with a region for q, and
(2) If p performs a $\mathsf{HLreply}(q, qreg)$, then p had requested q's location and q was either: (a) alive in region $qreg$ within the last ttl_{HLS} time, or (b) failed for at most $ttl_{hb} + ttl_{HLS} - \epsilon_{sample}$ time.

3.3 Client end-to-end routing (EtoEComm) specification

End-to-end routing is an important application for ad-hoc networks. The V-bcast service provides a local broadcast service where VSAs and clients can communicate with VSAs and clients in neighboring regions. VtoVComm allows arbitrary VSAs to communicate. End-to-end routing (EtoEComm) allows arbitrary clients to communicate: a client p sends message m to client q using $\mathsf{send}(q, m)_p$, which is received by q in bounded time via $\mathsf{receive}(m)_q$.

If clients p and q do not crash for ttl_{HLS} time, clients do not change regions for ttl_{HLS} time after a send, and q has been in the system at least $ttl_{HLS} + \epsilon_{sample} + d + ttl_{VtoV} + e$ time, then:
(1) If p sends m to q, q receives m within $ttl_{HLS} + 2d + 2e + ttl_{VtoV}$ time, and
(2) Any message received by a client was previously sent to the client.

3.4 Self-stabilizing implementations

We require implementations of the services to be self-stabilizing. A system configuration is *safe* with respect to a specification and implementation if any admissible execution fragment of the implementation starting from the configuration is an admissible execution fragment of the specification. An implementation is *self-stabilizing* if starting from any configuration, an admissible execution of the implementation eventually reaches a safe configuration. Notice if an implementation is self-stabilizing, then any long enough execution fragment of the implementation will eventually have a suffix that looks like the suffix of some correct execution of the specification, until a corruption occurs.

Each of the above services' self-stabilizing implementations will be built on top of self-stabilizing implementations of other services: VtoVComm over the VSA layer, the location service over the VSA layer and VtoVComm service, and

EtoEComm over the VSA layer, VtoVComm, and location services. Each self-stabilizing implementation uses lower level services without feedback, so lower level service executions are not influenced by the upper level services. This allows us to guarantee that higher level service implementations are still self-stabilizing through *fair composition* [11].

Our service implementations, starting from an arbitrary system configuration, stabilize within the following times: VtoVComm: $ttl_{VtoV} + d$ time after the VSA layer stabilizes, the location service: $max(ttl_{HLS}, 2e + 3ttl_{VtoV} + ttl_{hb} + 2d)$ time after VtoVComm stabilizes, and EtoEComm: $ttl_{pb} + 2d + 2e + ttl_{VtoV}$ time after the location service has stabilized.

4 VSA-to-VSA communication implementation

The VSA-to-VSA communication (VtoVComm) service allows communication of messages between any two VSAs through VtoVsend and VtoVrcv actions, as long as there is a path of non-failed VSAs between them. The VtoVComm service is built on top of the V-bcast service [8], which supports communication between two neighboring VSAs.

VSA-to-VSA communication is based on greedy DFS. When a VSA receives a message for which it is not the destination, it chooses a neighboring VSA that is on a shortest path to the destination VSA and forwards the message in a forward message to that neighbor. If the VSA does not receive an indication through a found message that the message has been delivered to the destination within some bounded amount of time, it then forwards the message to the neighboring VSA on the next shortest path to the destination VSA, and so on. This choice of neighbors is greedy in the sense that the next neighbor chosen to receive the forwarded message is the one on a shortest path to the destination VSA, excluding the neighbors associated with previous tries. The greedy DFS can turn into a flood in pathological situations in which the destination is that last VSA reached. Self-stabilization of the algorithm is ensured by the use of a real time timestamp to identify the version of the DFS. Too old versions are eliminated from the system and new versions are handled as completely new attempts to complete a greedy DFS towards the destination.

We present a simple greedy DFS that gradually expands the search until all paths are checked. This algorithm will find a path to the destination if such a path exists throughout the DFS execution. We also have a modification of the algorithm to produce a persistent version in which each VSA repeatedly tries to forward messages along previously unsuccessful paths to take advantage of recoveries of VSAs that may result in a viable path [13].

4.1 Detailed code description

The following code description refers to the code for VSA V_u^{VtoV} in Figure 2. The main state variable $DFStable$ keeps track of information for messages that are still waiting to be delivered. For each such unique message, the table stores the intermediate source of the message, the set of VSA neighbors that have yet to have the message forwarded to them, and a timeout for the neighbor currently being tried for forwarding the message.

A source VSA V_u^{VtoV} for region u sends a message m to a destination VSA in region d using $\mathsf{VtoVsend}(d,m)_u$ (line 35). If $u = d$ then V_u^{VtoV} receives m through $\mathsf{VtoVrcv}(m)_u$ (lines 37-38). Otherwise the destination VSA is another VSA and VSA V_u^{VtoV} sets the $DFStable$ mapping of an augmented version of the message, $\langle m, u, d, now\rangle$, to $\langle u, nbrs(u), now\rangle$. This enables the start of a new DFS execution to forward the message to its destination (line 39-40).

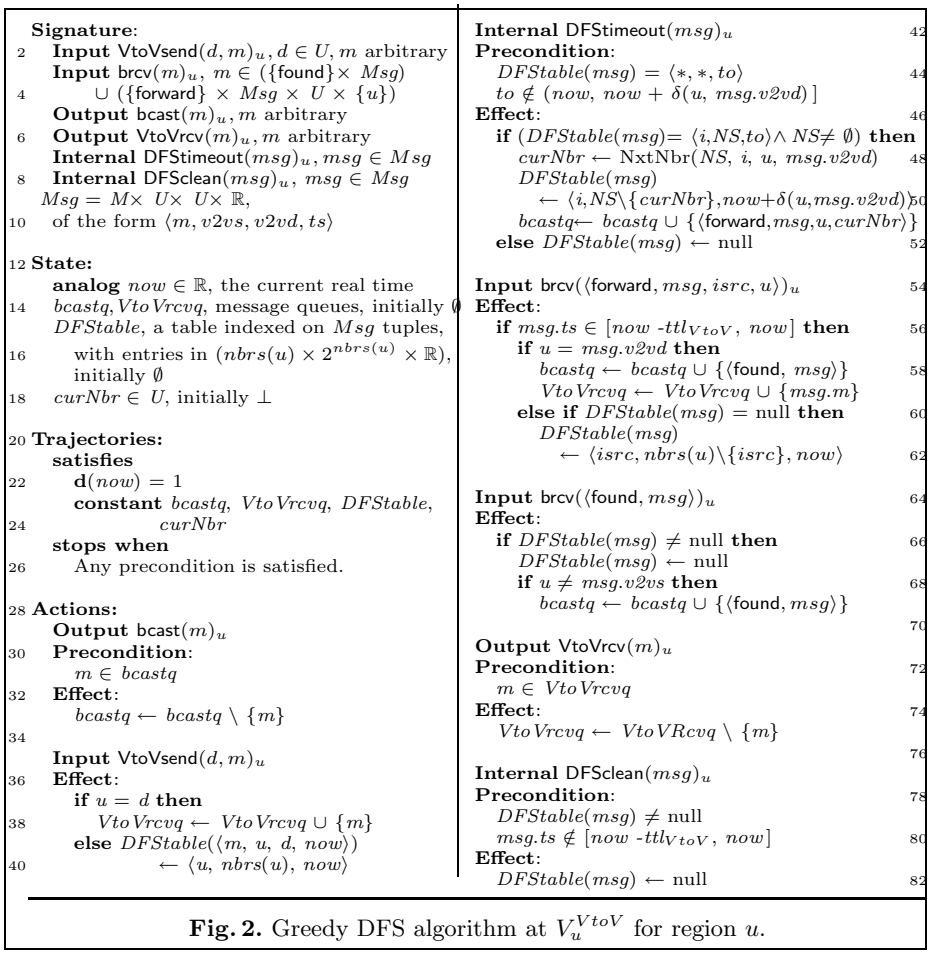


Fig. 2. Greedy DFS algorithm at V_u^{VtoV} for region u.

Whenever the forwarding of a message to a neighbor in $DFStable$ times out, it triggers forwarding to the next neighbor in the DFS, if possible. If the message hasn't yet been forwarded to all of the relevant neighbors, then the next neighbor closest to the destination VSA that has not yet had a message forwarded to it, $curNbr$, is selected and the message tuple msg is then forwarded in a forward message to it using the V-bcast service (lines 42-52). The timeout for this attempt

at forwarding is set to $\delta(curNbr, msg.v2vd)$ later, where $\delta : \{U\} \times \{U\} \rightarrow \mathcal{N}$ is a bound on the time required for a message to arrive from x to y. If the message has already been forwarded to all the relevant neighbors, then $DFStable(msg)$ is set to null, indicating that nothing more can be done.

If a tuple msg whose destination is u is received in a forward message from $isrc$, then V_u^{VtoV} broadcasts a $\langle$found, $msg\rangle$ message via the V-bcast service and VtoVrcv's the message $msg.m$. The found message notifies neighbors still participating in the DFS for msg that it has reached its destination. No forwarding is required (lines 56-59). Otherwise, if msg is not destined for V_u^{VtoV} and V_u^{VtoV} does not already have an entry in $DFStable$ for msg, then the message must be forwarded to its destination. $DFStable(msg)$ is set to $\langle isrc, nbrs(u)\backslash\{isrc\}, now\rangle$ (lines 61-62), storing the intermediate source, initializing the set of neighbors that have yet to have the message forwarded to them, and setting a timeout to now. Setting the timeout to now immediately enables the DFStimeout action for msg, triggering the forwarding of msg to one of V_u^{VtoV}'s neighbors.

When a found message is received for a message tuple msg that is mapped by $DFStable$, the entry in $DFStable$ is erased, preventing additional forwarding (line 67). If $u \neq msg.v2vs$ then VSA V_u^{VtoV} broadcasts a found message via the V-bcast service (lines 68-69), notifying neighbors that are still participating for msg that it has been delivered. Clearly, if $u = msg.v2vs$, then no found message is required and no further action needs to be taken.

4.2 Correctness

Let the source VSA be V_s^{VtoV}, the destination VSA be V_d^{VtoV}, the message sent be m, and a DFS execution exe from V_s^{VtoV} to V_d^{VtoV} be as defined above. Any non-negative wait time is sufficient for correctness. However, a wait time dependent on hop count between regions will be the most message-efficient. If no corruptions occur and the status (failed or non-failed) of every VSA doesn't change during exe, then the following holds:

Lemma 1. *If V_s^{VtoV} performs a VtoVsend(d, m) at time t, and there exists a path of non-failed VSAs between V_s^{VtoV} and V_d^{VtoV} from t to time $t + ttl_{VtoV}$, then V_d^{VtoV} performs a VtoVrcv(m) in the interval $[t, t + ttl_{VtoV}]$, for $ttl_{VtoV} \geq [e + d + (max_{u,v \in U}\delta(u, v) \cdot max_{u \in U}|nbrs(u)| - 1)] \cdot (|U| - 1)$.*

Lemma 2. *The number of times a message tuple is re-broadcast is bounded.*

Lemma 3. *Once corruptions stop and the VSA layer has stabilized, it takes up to $d + ttl_{VtoV}$ time for VtoVComm to stabilize.*

5 Home Location Service (HLS) implementation

The location service allows a client to determine a recent region of another alive client. In our implementation, called the *Home Location Service (HLS)*, we accomplish this using *home locations*. Recall that the home locations of a client node p are $f + 1$ regions whose VSAs are occasionally updated with p's region. The home locations are calculated with a hash function h, mapping a client's id to a list of VSA regions, and is known to all VSAs. These home location VSAs can then be queried by other VSAs to determine a recent region of p.

The HLS implementation consists of two parts: a client-side portion and a VSA-side portion. C_p^{HL} is a subautomaton of client p that interacts with VSAs to provide HLS. It notifies local VSAs of its region. It also handles $\mathsf{HLquery}(q)_p$ requests, by broadcasting the query to local VSAs. It translates responses from the VSAs into HLreply outputs. For the VSA-side, V_u^{HL} and V_v^{HL} are home location VSAs corresponding to regions u and v of the network; they are subautomata of VSAs V_u and V_v. V_u^{HL} takes a request from a local client for client node q's region, calculates q's home locations using the hash function, and then sends location queries to the home locations using VtoVComm. Home locations respond with region information they have for q, which is then provided by V_u^{HL} to the requesting client. V_u^{HL} also is responsible both for informing the home locations of each client p located in its region or neighboring regions of p's region, and answering queries for the regions of clients for which it is a home location.

Time and region information from the GPS oracle is used throughout the HLS algorithm, by clients and VSAs, to timestamp and label information and messages. This information is used to guarantee timeliness of replies from the HLS service, and to stabilize the service after faults. Timestamps are used to determine if information is too old or too new, while region information allows clients and VSAs to know which other clients and VSAs to interact with.

5.1 HLS client actions

Clients receive GPSupdates every ϵ_{sample} time from the GPS automaton, making them aware of their current region and the time. If a client's region has changed, the client immediately sends a heartbeat message with its id, current time and region information. The client periodically reminds its current and neighboring region VSAs of its region by broadcasting additional heartbeat messages every ttl_{hb} time, where ttl_{hb} is a known constant.

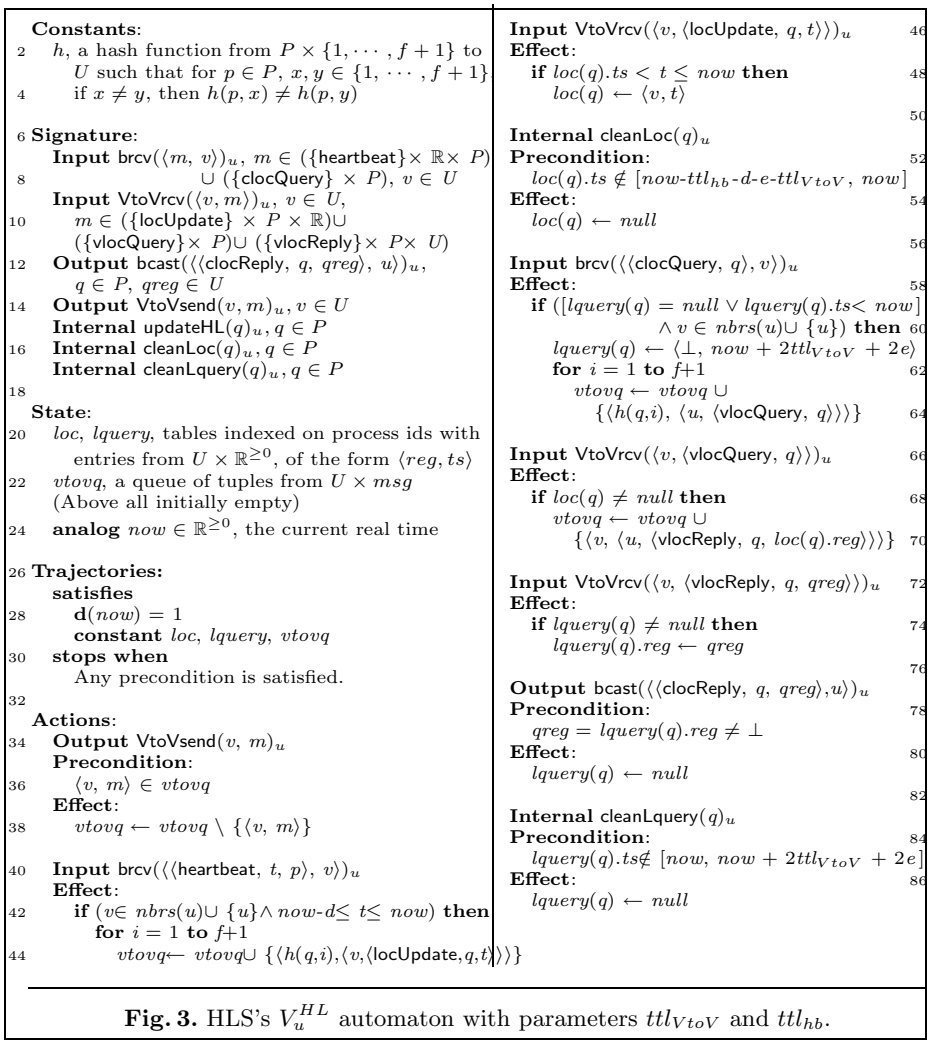
C_p^{HL} also handles the $\mathsf{HLquery}(q)$ inputs it receives. This request for q's location is stored in a *queryq* table and, once the client knows its own region, translated into a $\langle$clocQuery, $q\rangle$ message that is broadcast, together with the VSA region, to local regions' VSAs. If C_p^{HL} eventually receives a $\langle \mathsf{clocReply}, q, qreg\rangle$ message from its current or neighboring region's VSA for a client q in *queryq*, indicating that node q was in region $qreg$, it clears the entry for q in *queryq*, and outputs a $\mathsf{HLreply}(q, qreg)$ of the information. If the request goes unanswered for more than $ttl_{HLS} - \epsilon_{sample}$ time, then the request has failed and is removed.

5.2 HLS VSA actions

The code for automaton V_u^{HL} appears in Figure 3. The VSA knows of local clients through heartbeat messages. If a VSA hears a heartbeat from a client p claiming to be in its region or a neighboring region, the VSA sends a locUpdate message for p, with p's heartbeat timestamp and region, through VtoVComm to the VSAs at home locations of client p (lines 40-44); home locations are computed using a known hash function h from $P \times \{1, \cdots, f+1\}$ to U.

When a VSA receives one of these locUpdate messages for a client p, it stores both the region indicated in the message as p's current region and the attached heartbeat timestamp in its *loc* table (lines 46-49). This location information for p is refreshed each time the VSA receives a locUpdate for client p with a

newer heartbeat timestamp. Since a client sends a heartbeat message every ttl_{hb} time, which can take up to $d + e$ time to arrive at and trigger a VSA to send a locUpdate message through VtoVComm, which can take ttl_{VtoV} time to be delivered at a home location, an entry for client p is erased if its timestamp is older than $ttl_{hb} + d + e + ttl_{VtoV}$ (lines 51-55).



Constants:
2 h, a hash function from $P \times \{1, \cdots, f+1\}$ to U such that for $p \in P$, $x, y \in \{1, \cdots, f+1\}$
4 if $x \neq y$, then $h(p, x) \neq h(p, y)$

6 **Signature**:
Input brcv($\langle m, v\rangle)_u$, $m \in$ ({heartbeat}$\times \mathbb{R} \times P$)
8 $\cup$ ({clocQuery} $\times P$), $v \in U$
Input VtoVrcv($\langle v, m\rangle)_u$, $v \in U$,
10 $m \in$ ({locUpdate} $\times P \times \mathbb{R}$)$\cup$
({vlocQuery}$\times P$)$\cup$ ({vlocReply}$\times P \times U$)
12 **Output** bcast($\langle\langle$clocReply, q, $qreg\rangle$, $u\rangle)_u$,
$q \in P$, $qreg \in U$
14 **Output** VtoVsend$(v, m)_u$, $v \in U$
Internal updateHL$(q)_u$, $q \in P$
16 **Internal** cleanLoc$(q)_u$, $q \in P$
Internal cleanLquery$(q)_u$, $q \in P$
18
State:
20 loc, $lquery$, tables indexed on process ids with
entries from $U \times \mathbb{R}^{\geq 0}$, of the form $\langle reg, ts\rangle$
22 $vtovq$, a queue of tuples from $U \times msg$
(Above all initially empty)
24 **analog** $now \in \mathbb{R}^{\geq 0}$, the current real time

26 **Trajectories:**
satisfies
28 **d**$(now) = 1$
constant loc, $lquery$, $vtovq$
30 **stops when**
Any precondition is satisfied.
32
Actions:
34 **Output** VtoVsend$(v, m)_u$
Precondition:
36 $\langle v, m\rangle \in vtovq$
Effect:
38 $vtovq \leftarrow vtovq \setminus \{\langle v, m\rangle\}$

40 **Input** brcv($\langle\langle$heartbeat, t, $p\rangle$, $v\rangle)_u$
Effect:
42 **if** ($v \in nbrs(u) \cup \{u\} \wedge now\text{-}d \leq t \leq now$) **then**
for $i = 1$ **to** $f+1$
44 $vtovq \leftarrow vtovq \cup \{\langle h(q,i), \langle v, \langle$locUpdate$, q, t\rangle\rangle\rangle\}$

Input VtoVrcv($\langle v, \langle$locUpdate, $q, t\rangle\rangle)_u$ 46
Effect:
if $loc(q).ts < t \leq now$ **then** 48
$loc(q) \leftarrow \langle v, t\rangle$
50
Internal cleanLoc$(q)_u$
Precondition: 52
$loc(q).ts \notin [now\text{-}ttl_{hb}\text{-}d\text{-}e\text{-}ttl_{VtoV}, now]$
Effect: 54
$loc(q) \leftarrow null$
56
Input brcv($\langle\langle$clocQuery, $q\rangle, v\rangle)_u$
Effect: 58
if ($[lquery(q) = null \vee lquery(q).ts < now] \wedge v \in nbrs(u) \cup \{u\}$) **then** 60
$lquery(q) \leftarrow \langle \perp, now + 2ttl_{VtoV} + 2e\rangle$
for $i = 1$ **to** $f+1$ 62
$vtovq \leftarrow vtovq \cup$
$\{\langle h(q,i), \langle u, \langle$vlocQuery, $q\rangle\rangle\rangle\}$ 64

Input VtoVrcv($\langle v, \langle$vlocQuery, $q\rangle\rangle)_u$ 66
Effect:
if $loc(q) \neq null$ **then** 68
$vtovq \leftarrow vtovq \cup$
$\{\langle v, \langle u, \langle$vlocReply, q, $loc(q).reg\rangle\rangle\rangle\}$ 70

Input VtoVrcv($\langle v, \langle$vlocReply, q, $qreg\rangle\rangle)_u$ 72
Effect:
if $lquery(q) \neq null$ **then** 74
$lquery(q).reg \leftarrow qreg$
76
Output bcast($\langle\langle$clocReply, q, $qreg\rangle$,$u\rangle)_u$
Precondition: 78
$qreg = lquery(q).reg \neq \perp$
Effect: 80
$lquery(q) \leftarrow null$
82
Internal cleanLquery$(q)_u$
Precondition: 84
$lquery(q).ts \notin [now, now + 2ttl_{VtoV} + 2e]$
Effect: 86
$lquery(q) \leftarrow null$

Fig. 3. HLS's V_u^{HL} automaton with parameters ttl_{VtoV} and ttl_{hb}.

The other responsibility of the VSA is to receive and respond to local client requests for location information on other clients. A client p in a VSA's region or a neighboring region v can send a query for q's current location to the VSA. This is done via a mobile node's broadcast of a $\langle\langle$clocQuery, $q\rangle, v\rangle$ message. When the VSA at region u receives this query, if no outstanding query for q exists, it

notes the request for q in $lquery(q)$, and sends a vlocQuery message to q's $f+1$ home locations, querying about q's location (lines 57-64). Any home location that receives such a message and has an entry for q's region responds with a vlocReply to the querying VSA with the region (lines 66-70).

If the querying VSA at u receives a vlocReply in response to an outstanding location request for a client q, it stores the attached region information in $lquery(q)$ (lines 72-75), broadcasts a clocReply message with q and its region to local clients, and erases the entry for $lquery(q)$ (lines 77-81). If, however, $2ttl_{VtoV} + 2e$ time passes since a request for q's region was received by a local client and there is no entry for q's region, $lquery(q)$ is just erased (lines 83-87).

5.3 Correctness

We make the system assumptions described in Section 3. For the following two lemmas and theorem, assume the system starts in a safe configuration, and no corruptions occur.

Lemma 4. *For any VSA u, if there is a request for q's region in lquery, it was submitted through a* HLquery(q) *within the last $\epsilon_{sample} + d + 2ttl_{VtoV} + 2e$ time.*

Lemma 5. *Starting $\epsilon_{sample}+d+e+ttl_{VtoV}$ time after client p enters the system and until p fails, for each interval of length $ttl_{VtoV} + e$, all but f of p's home locations will have a non-null $loc(p)$ entry for the entire interval. If client p is alive and there is some VSA u such that $loc(p)$ is not null, p was alive and located in $loc(p).reg$ within the last $\epsilon_{sample} + d + e + ttl_{VtoV}$ time.*

Theorem 1. *Every client p searching for a non-failed client q that has been in the system longer than $ttl_{HLS} + \epsilon_{sample} + d + ttl_{VtoV} + e$ time will perform a* HLreply$(q, qreg)$ *within time ttl_{HLS}, such that q was located in region $qreg$ no more than ttl_{HLS} time ago. No reply will occur if q has been failed for more than $ttl_{hb} + ttl_{HLS} - \epsilon_{sample}$ time. Any reply is in response to a query.*

Proof sketch: By the prior lemma, once client q has been in the system for $\epsilon_{sample} + d + e + ttl_{VtoV}$ time, any queries of its home locations will succeed in producing a result. However, a new HLquery request "piggybacks" on any prior unexpired HLquery requests. Since one of these requests could have been initiated just before the client q's home locations are updated, we can only guarantee a response will be received for a new request if any outstanding requests will be answered. If the client has been in the system for this total $ttl_{HLS}+d+e+ttl_{VtoV}$ time after receiving its first GPSupdate, then any response to a query can take as much as ttl_{HLS} time: ϵ_{sample} time for the querying client to receive its first GPSupdate, d time for the query to be transmitted and received by a local VSA, $e+ttl_{VtoV}$ for the local VSA to query a home location, $e+ttl_{VtoV}$ for the response to arrive at a local VSA, e time for the local VSA to transmit the response to its requesting clients, and d time for the transmission to be received and translated into HLreplys at clients. By the prior lemma, we know that information can only be out of date by $\epsilon_{sample} + ttl_{VtoV} + e + d$ time when a home location responds to a query by another VSA. The response can take $e + ttl_{VtoV}$ time to arrive at the querying VSA, followed by $e + d$ time for the querying VSA to get the

information to the clients that prompted the query. The oldest the information could be is the total.

For the second statement, note that a failed client will not send a heartbeat message. Since $loc(p)$ entries are cleared once $ttl_{hb} + d + e + ttl_{VtoV}$ time has passed since the heartbeat message upon which it was based was broadcast, and the information from the entry can only take as much as $e + ttl_{VtoV}$ time to reach a querying VSA and $e + d$ time to reach any querying clients, the total is the maximum time a HLreply can occur after the client fails.

For the third statement, a query expires after ttl_{HLS} time. Hence, any response generated must be for a query that is not older. □

Theorem 2. *Starting from an arbitrary configuration, after VtoVComm has stabilized, it takes* $max(ttl_{HLS}, 2e+3ttl_{VtoV}+ttl_{hb}+2d)$ *time for HLS to stabilize.*

Proof sketch: Once lower levels have stabilized, most client state is made locally consistent within ϵ_{sample} time, the time for a GPSupdate. This action resets most variables if the region is updated. The remaining state is made consistent instantaneously with local correction, except for the heartbeat timer and *queryq* variables. The heartbeat timer can affect operations for at most ttl_{hb} time. The *queryq* variable can affect operations for ttl_{HLS} time, when it would be deleted.

For VSAs, there are two variables that are not instantaneously corrected: *loc* and *lquery*. The *loc* variable will be consistent within time $e+2ttl_{VtoV}+ttl_{hb}+d$. At worst, there could be a corrupted message that arrives at a VSA after ttl_{VtoV} time, adding a bad entry that takes $e + ttl_{VtoV} + ttl_{hb} + d$ time to expire. If the client referred to is in the system, it might not be until the next update after the timestamp of the corrupted message (which could have been delivered as late as ttl_{VtoV} after corruptions stopped) arrives for the information to be cleaned up. This time is exactly what the offset term for *loc* timeouts describes. Hence, the variable might not be cleaned until ttl_{VtoV} plus that offset term.

However, there may be responses based on this bad *loc* table information that were sent right at $e + 2ttl_{VtoV} + ttl_{hb} + d$, and that take $e + ttl_{VtoV}$ to arrive at the VSA. The resulting transmission (taking d time to complete) to local clients is then incorrect. However, those incorrect transmissions cease after the total time $2e + 3ttl_{VtoV} + ttl_{hb} + 2d$ elapses.

The *lquery* variable is cleaned up within ttl_{HLS} time. An entry in *lquery* only has a total of $2ttl_{VtoV} + 2e$ time in the data structure. It could be the case that a spurious request was transmitted in the beginning, which adds d time. If a region response is received it results in immediate correction of the state through erasure. Hence, the time required to be consistent is the time that it takes for a query to be accounted for. The maximum of ttl_{HLS} and $2e+3ttl_{VtoV}+ttl_{hb}+2d$ is the maximum stabilization time. □

6 Client end-to-end routing (EtoEComm) implementation

Our implementation of the end-to-end routing service, EtoEComm, uses the location service to discover a recent region location of a destination client node

and then uses this location in conjunction with VtoVComm to deliver messages. As in the implementation of the Home Location Service, there are two parts to the implementation: the client-side portion and the VSA-side portion.

A message m is sent to another client q via $\mathsf{send}(q, m)_p$. This input to client-side C_p^{E2E} results in the forwarding of the message to p's current region u's and neighboring VSAs through a local broadcast of the message with the the destination q and q's location, if q's location is known. If a recent region for q is not known, C_p^{E2E} queries HLS to determine one. A timeout for response to the location request is set for ttl_{HLS} later. Once a response is received from HLS in the form of $\mathsf{HLreply}(q, qreg)_p$, indicating q was in region $qreg$, the location of q is stored and kept for ttl_{pb} time. For each message waiting to be sent to q, the message, labeled with q and $qreg$, is forwarded to p's current and neighboring regions' VSAs through a local broadcast, as before.

Messages for client p from other clients are received from p's current region or a neighboring region v's VSA through a local broadcast from a local VSA. The message is subsequently delivered through the output $\mathsf{receive}(m)_p$.

The VSA V_u^{E2E} portion is very simple. A client may send it information to be transmitted to other VSAs, which it forwards through VtoVComm, or another VSA may send it information to be delivered at a client in its own or a neighboring region, which it forwards through V-bcast.

The receipt of a locally broadcast message m from a client p in region u or a neighboring region to q at region $qreg$ results in the subsequent forwarding of the message to the virtual automata at regions $\mathsf{calcregs}(qreg)$ and their neighboring regions, via the virtual automata VtoVComm service. The set of VSA regions $\mathsf{calcregs}(qreg)$ describes the regions that q may occupy by the time the message is delivered to it. The receipt, via VtoVComm of message m intended for client p in region u or a neighboring region results in the forwarding of the message to p through a local broadcast.

7 Concluding remarks

We described how both the GPS oracle and the VSA layer could help implement self-stabilizing geocast routing, location management, and end-to-end routing services. The self-stabilizing VSA layer provides a virtual fixed infrastructure useful for solving a variety of problems. It acts as a fault-tolerant, self-stabilizing building block for services, allowing applications to be built for mobile networks as though base stations existed for mobile clients to interact with.

The GPS oracle's frequently refreshed and reliable timing and location information made providing self-stabilization easier. The paradigm of an external service providing reliable information that can be used in a self-stabilizing service implementation is an especially important and relevant one in mobile networks. Mobile networks demonstrate many properties that naturally require self-stabilizing implementations, such as a need for self-configuration, or the possibility of unpredictable kinds of failures, but also often have access to reliable external knowledge that can act as a source of shared consistency in the network; here, accurate region knowledge allowed nodes to determine who they should be communicating with (current region and neighboring region nodes),

and time information allowed them to order messages and assess timeliness of information.

References

1. Abraham, I., Dolev, D., and Malkhi, D., "LLS: A Locality Aware Location Service for Mobile Ad Hoc Networks", *Proceedings of the DIALM-POMC Joint Workshop on Foundations of Mobile Computing (DIALM-POMC)*, pp. 75-84, 2004.
2. Arora, A., Demirbas, M., Lynch, N., and Nolte, T., "A Hierarchy-based Fault-local Stabilizing Algorithm for Tracking in Sensor Networks", *8th International Conference on Principles of Distributed Systems (OPODIS)*, pp. 207-217, 2004.
3. Camp, T., Liu, Y., "An adaptive mesh-based protocol for geocast routing", *Journal of Parallel and Distributed Computing: Special Issue on Mobile Ad-hoc Networking and Computing*, pp. 196–213, 2002.
4. Dijkstra, E.W., "Self stabilizing systems in spite of distributed control", *Communications of the ACM*, pp. 643-644, 1974.
5. Dolev, S., *Self-Stabilization*, MIT Press, 2000.
6. Dolev, S., Gilbert, S., Lynch, N., Schiller, E., Shvartsman, A., and Welch, J., "Virtual Mobile Nodes for Mobile Ad Hoc Networks", *International Conference on Principles of Distributed Computing (DISC)*, pp. 230-244, 2004.
7. Dolev, S., Gilbert, S., Lynch, N., Shvartsman, A., Welch, J., "GeoQuorums: Implementing Atomic Memory in Ad Hoc Networks", *17th International Conference on Principles of Distributed Computing (DISC)*, Springer-Verlag LNCS:2848, pp. 306-320, 2003. Also to appear in *Distributed Computing*.
8. Dolev, S., Gilbert, S., Lahiani, L., Lynch, N., and Nolte, T., "Timed Virtual Stationary Automata for Mobile Networks", Technical Report MIT-LCS-TR-979a, MIT CSAIL, Cambridge, MA 02139, 2005.
9. Dolev, S., Gilbert, S., Lahiani, L., Lynch, N., and Nolte, T., "Brief Announcement: Virtual Stationary Automata for Mobile Networks", *Proceedings of the 24th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 323, 2005.
10. Dolev, S., Herman, T., and Lahiani, L., "Polygonal Broadcast, Secret Maturity and the Firing Sensors", *Third International Conference on Fun with Algorithms (FUN)*, pp. 41-52, May 2004. Also to appear in *Ad Hoc Networks Journal*, Elseiver.
11. Dolev, S., Israeli, A., and Moran, S., "Self-Stabilization of Dynamic Systems Assuming only Read/Write Atomicity", *Proceeding of the ACM Symposium on the Principles of Distributed Computing (PODC 90)*, pp. 103-117. Also in *Distributed Computing* 7(1): 3-16 (1993).
12. Dolev, S., Pradhan, D.K., and Welch, J.L., "Modified Tree Structure for Location Management in Mobile Environments", *Computer Communications*, Special issue on mobile computing, Vol. 19, No. 4, pp. 335-345, April 1996. Also INFOCOM 1995, Vol. 2, pp. 530-537, 1995.
13. Dolev, S. and Welch, J.L.,"Crash Resilient Communication in Dynamic Networks", *IEEE Transactions on Computers*, Vol. 46, No. 1, pp.14-26, January 1997.
14. Haas, Z.J. and Liang, B., "Ad Hoc Mobility Management With Uniform Quorum Systems", *IEEE/ACM Trans. on Networking*, Vol. 7, No. 2, pp. 228-240, April 1999.
15. Hubaux, J.P., Le Boudec, J.Y., Giordano, S., and Hamdi, M., "The Terminodes Project: Towards Mobile Ad-Hoc WAN", *Proceedings of MOMUC*, pp. 124-128, 1999.
16. Imielinski, T., and Badrinath, B.R., "Mobile wireless computing: challenges in data management", *Communications of the ACM*, Vol. 37, Issue 10, pp. 18-28, 1994.
17. Karp, B. and Kung, H. T., "GPSR: Greedy Perimeter Stateless Routing for Wireless Networks", *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, pp. 243-254, SCM Press, 2000.
18. Kaynar, D., Lynch, N., Segala, R., and Vaandrager, F., "The Theory of Timed I/O Automata", Technical Report MIT-LCS-TR-917a, MIT LCS, 2004.
19. Kuhn, F., Wattenhofer, R., Zhang, Y., Zollinger, A., "Geometric Ad-Hoc Routing: Of Theory and Practice", *Proceedings of the 22nd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 63-72, 2003.
20. Kuhn, F., Wattenhofer, R., and Zollinger, A., "Asymptotically Optimal Geometric Mobile Ad-Hoc routing", *Proceedings of the 6th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (Dial-M)*, pp. 24-33, ACM Press, 2002.
21. Li, J., Jannotti, J., De Couto, D.S.J., Karger, D.R., and Morris, R., "A Scalable Location Service for Geographic Ad Hoc Routing", *Proceedings of Mobicom*, pp. 120-130, 2000.
22. Malkhi, D., Reiter, M., and Wright, R., "Probabilistic Quorum Systems", *Proceeding of the 16th Annual ACM Symposium on the Principles of Distributed Computing (PODC 97)*, pp. 267-273, Santa Barbara, CA, August 1997.
23. Nath, B., Niculescu, D., "Routing on a curve", *ACM SIGCOMM Computer Communication Review*, pp. 155-160, 2003.
24. Navas, J.C., Imielinski, T., "Geocast- geographic addressing and routing", *Proceedings of the 3rd MobiCom*, pp. 66-76, 1997.
25. Ratnasamy, S., Karp, B., Yin, L., Yu, F., Estrin, D., Govindan, R., and Shenker, S., "GHT: A Geographic Hash Table for Data-Centric Storage", *First ACM International Workshop on Wireless Sensor Networks and Applications*, pp. 78-87, 2002.