Advance Access publication on 5 September 2012 doi:10.1093/comjnl/bxs117

Decentralized Monitoring of Moving Objects in a Transportation Network Augmented with Checkpoints

ALAN BOTH[1,*], MATT DUCKHAM[1], PATRICK LAUBE[2], TIM WARK[3] AND JEREMY YEOMAN[1]

[1]*Department of Infrastructure Engineering, University of Melbourne, Melbourne, Australia*
[2]*Department of Geography, University of Zürich, Zürich, Switzerland*
[3]*CSIRO ICT Center, QCAT Brisbane, Brisbane, Australia*
**Corresponding author: aboth@student.unimelb.edu.au*

This paper examines efficient and decentralized monitoring of objects moving in a transportation network. Previous work in moving object monitoring has focused primarily on centralized information systems, like moving object databases and geographic information systems. In contrast, in this paper monitoring is in-network, requiring no centralized control and allowing for substantial spatial constraints to the movement of information. The transportation network is assumed to be augmented with fixed checkpoints that can detect passing mobile objects. This assumption is motivated by many practical applications, from traffic management in vehicle *ad hoc* networks to habitat monitoring by tracking animal movements. In this context, this paper proposes and evaluates a family of efficient decentralized algorithms for capturing, storing and querying the movements of objects. The algorithms differ in the restrictions they make on the communication and sensing constraints to the mobile nodes and the fixed checkpoints. The performance of the algorithms is evaluated and compared with respect to their scalability (in terms of communication and space complexity), and their latency (the time between when a movement event occurs, and when all interested nodes are updated with records about that event). The conclusions identify three key principles for efficient decentralized monitoring of objects moving past checkpoints: structuring computation around neighboring checkpoints; taking advantage of mobility diffusion and separating the generation and querying of movement information.

Keywords: geosensor network; decentralized spatial computing; network-constrained movement; moving objects; environmental monitoring

Received 14 December 2011; revised 27 June 2012
Handling editor: Jacob Beal

1. INTRODUCTION

When monitoring objects moving within a transportation network, two alternative perspectives immediately present themselves: monitor locations with change over time or monitor times over change in location. The former perspective is exemplified by using *trajectories* to represent the motion of objects, where at fixed times the location of the moving object is recorded (e.g. by sampling global positioning system coordinates every minute). The latter perspective is exemplified by using *checkpoints*, where the time at which moving objects pass fixed locations is recorded. Figure 1 summarizes these two perspectives, trajectory, and checkpoints, in the case of a single object moving through a transportation network.

This paper examines the design of algorithms for monitoring and querying of moving objects passing checkpoints. More specifically, we restrict the focus in this paper to movement within a transportation network (although the principles and algorithms developed can also be adapted to movement in unconstrained planar spaces). Following terminology used in the transportation literature, in this paper transportation networks augmented with checkpoints are referred to as *cordon-structured networks*.

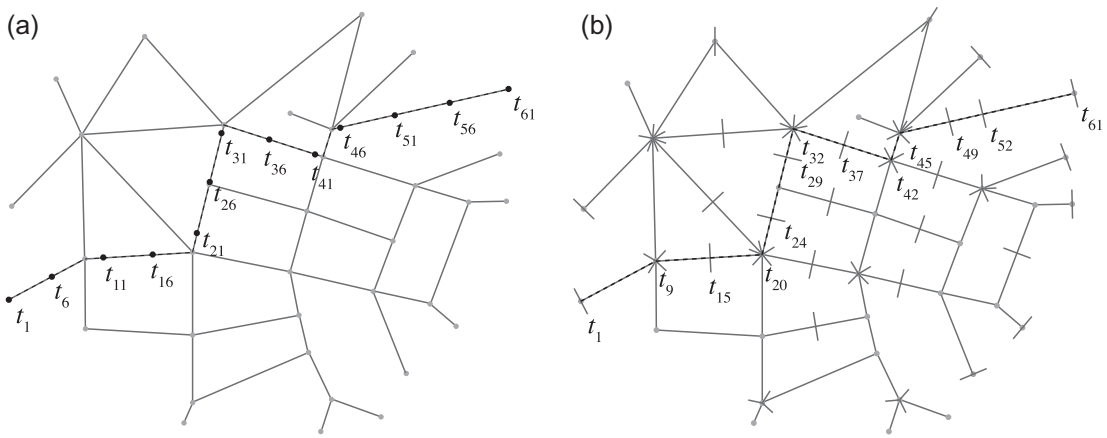

FIGURE 1. Movement of an object: (**a**) as a trajectory (sequence of time-stamped locations) and (**b**) past checkpoints (small bisecting lines) in a transportation network.

In addition to the novelty of examining cordon-structured networks, rather than more traditional trajectory-based approaches, this paper also focuses on efficient *decentralized* algorithms, which operate with no centralized information store or control. Decentralized algorithms are an active area of research in spatial computing and spatial information science (e.g. [1–4]), in part because they are well suited to use with new technologies like wireless sensor networks and vehicle *ad hoc* networks (VANETs). Using a decentralized algorithm enables queries about spatiotemporal events to be satisfied partly or wholly *in the network*, without the need to communicate and collate information about object movements within a single, centralized information system. Decentralization may also have important implications for protecting privacy, as the information about the movements of individual nodes is smeared across the entire network rather than at a single centralized node, potentially making invasions of privacy harder to engineer.

This paper defines and tests decentralized algorithms for scalable and efficient monitoring of moving objects in cordon-structured networks, as well as defining mechanisms for querying this information. Examples of applications for our algorithms include as follows:

(i) Monitoring the movement of vehicles around road networks, where vehicle locations are tracked by fixed road network infrastructure, like electronic tolling gantries, traffic cameras and inductive loops (e.g. [5]).
(ii) Monitoring the movement of emergency personnel through indoor spaces during an emergency relief effort, where personnel are tracked moving past checkpoints at key points within a building (e.g. [6]).
(iii) Monitoring the movement of animals in a network, such as native fish species tagged with radio transmitters passing logging stations placed at key positions around a river network (e.g. [7]).

Following a discussion of the related work (Section 2) and the precise definition of the underlying structures used in our model (Section 3), this paper presents a family of related decentralized algorithms for maintaining and querying information about movement events in cordon-structured networks (Sections 4 and 6). The key idea behind the decentralized algorithms in this paper is to distribute records about observed movement events across nodes in the network, storing the times when mobile objects pass cordons. Specifically, we store at every cordon information about the movement events on the edges that are incident with that cordon. This approach is designed to support a range of queries over these networks, from queries about the location and movement of mobile objects, through queries about the loads and flows on transportation network edges, to queries about fault detection, for example, inferring when cordons are not working correctly.

We show that our approach can adapt to increasingly restrictive assumptions about the spatial constraints to the movement of information made by the communication network, from communication between nearby cordons, to a largely disconnected network, where only near-coincident cordons and moving objects have the opportunity to communicate. In all cases, the spatial structure of the network, as well as movement through the network, is used to minimize communication and reduce redundancy in stored information. The different options are evaluated and compared experimentally, based on their scalability and latency (Sections 5 and 6). The evaluation demonstrates that the approach is highly scalable, both in terms of communication and storage of information, even for sparsely and infrequently connected networks. The conclusions in Section 7 summarize the findings and the avenues for ongoing research.

2. RELATED WORK

This paper lies at the intersection of two established research topics: monitoring mobility in cordon-structured networks and decentralized spatial computing.

2.1. Trajectories and checkpoints

The term 'trajectory' in this paper refers specifically to a 'set of n moving point objects whose locations are known at t *consecutive time steps* [emphasis added]' [8], as opposed to the more general definition of a trajectory as 'polyline in three-dimensional space (two-dimensional geography, plus time), represented as a sequence of points (x, y, t)' [9]. This latter definition would also include movement data from checkpoints. However, we argue that it is the former definition, a sequence of locations at consecutive time steps, that most researchers have in mind when using the term 'trajectory'.

The distinction between representing movement using trajectories or checkpoints has its roots in the fundamental differences between the Eulerian and the Lagrangian views of movement [10]. The trajectory perspective is akin to the Lagrangian view, which considers changes in a moving object's location. The checkpoint perspective is closer to the Eulerian view, where the movement is described as changes in location relative to known, fixed points in space. A similar distinction is made in [11] in the context of continuous and discrete models of space.

We use the term 'cordon-structured network' to refer to a transportation network augmented with checkpoints. The term 'cordon' in this sense is adopted from the transportation literature (cf. [12–15]). Whereas the focus in the transportation literature is on the *regions* enclosed by checkpoints, in this paper our focus is slightly different, on the relative network locations of the checkpoints themselves and the movement of objects past these locations.

We often intuitively associate the cordon-structured networks with low spatial precision, caused by irregular and wide spacing between checkpoints (like fixed traffic cameras or inductive loops in traffic monitoring applications). However, it should be noted that this is not a feature of Eulerian (checkpoint) or Lagrangian (trajectory) perspectives on moving objects; indeed in some applications, closely spaced 'checkpoints' (like radio-frequency identification tag readers) have been used for highly accurate and precise positioning (e.g. [16, 17]).

Conversely, even if imprecise, movement data from cordon-structured networks are frequently more *accurate* than trajectory data. The locations of checkpoints can usually be known with a high degree of certainty, and objects moving past these checkpoints can often be unambiguously and reliably identified (e.g. an electronic road tolling system may have low spatial precision, dependent on the spacing of traffic gantries, but high spatial accuracy, only very rarely failing to correctly identify vehicles passing a gantry). Further, in cordon-structured networks, the checkpoints are usually located at structurally or semantically important locations, like intersections. As a result, the data generated by cordon-structured networks are typically more concise. In contrast, removing redundant points from voluminous trajectory data is an increasingly important research problem [18].

Related research in the area of monitoring moving objects typically assumes one or more of the following: trajectory-based movement data (e.g. [19]); movement in planar space, unconstrained by transportation networks (e.g. [20–22]) or centralized storage and processing of movement data (e.g. most work in the area of moving object databases [23]). Such assumptions are not well suited to many emerging applications, including those indicated above (e.g. where tagged fish are monitored moving through a river network by fixed logging stations in remote locations with restricted communication capabilities).

2.2. Decentralized spatial computing

The key feature of spatial computing (as distinct from computing with spatial information) is that there exist spatial constraints to the *movement* of information [11]. These constraints may arise for a variety of reasons, including resource limitations (e.g. limited energy for communication in untethered sensor nodes); the need to avoid information overload (e.g. through in-network filtering of low value or relevance data) and the desire for increased scalability and decreased operational latency (e.g. in a sensor/actuator network) [24].

In turn, the spatial constraints to movement of information motivate the interest in *decentralized* algorithms for spatial computing environments. Decentralized algorithms are a special case of distributed algorithms where no single node in a distributed system possesses global knowledge of the system's state [25]. Recent years have seen substantial activity in exploring fundamental decentralized algorithms to support spatial computing, for example: leader election [26, 27], localization [28, 29] and coordination across the network [30–32].

A feature of decentralized algorithms is that they are designed to operate on all nodes identically (with the same computation and data storage requirements) regardless of the network size. A fundamental result in distributed systems theory has proved that no matter how many different types of nodes and behaviors are required by a heterogeneous network, it is always possible to define a single, homogeneous protocol that can satisfy those requirements [27]. This simplifies the process of decentralized algorithm design, allowing networks to scale organically by adding nodes with identical protocols as required.

In the context of decentralized algorithms for monitoring and querying moving objects, one of the research areas that has the most in common with the assumptions behind this paper is VANETs (transportation applications based on wireless vehicle-to-vehicle and vehicle-to-infrastructure local area network communications, [33]). VANET research is explicitly concerned with movement through a transportation network, and does already consider decentralized approaches to communication and information processing (e.g. [34–36]). Decentralization is necessary due to the highly variable nature

of VANET connectivity, where no consistent communication coordinator can be assumed [37]. However, unlike the approach in this paper, such studies do assume trajectory, rather than cordon-structured movement data.

A key assumption behind our algorithms is that each node has a unique identity. This is a common assumption for decentralized algorithms where coordination between neighbors occurs (e.g. [25, 27]), and is frequently found in practice (e.g. in VANETs, communication typically relies on a media access control layer that includes an addressing mechanism; similarly, unique identities are usually available in our motivating example of environmental monitoring of fish movements within a river network [7]).

Our latter two algorithms do rely on *data mules* to physically move information toward sinks in the absence of communication network connectivity [38]. This idea of using the movement of objects in this way has already received much attention in the literature, also termed the *mobility diffusion effect* [39], *participatory data transfer* [40] and is closely related to *opportunistic data dissemination* [41, 42].

3. FORMAL MODEL

Before specifying our algorithms, it is necessary to precisely define the network and information structures we assume exist. We assume three interrelated types of network: a *communication* network (of cordons and mobile objects); a *transportation* network (through which objects are moving) and a *connectivity* network, which models the relative network locations of the cordons (in terms of their direct connectivity) in the transportation network.

3.1. Communication network

A sensor network comprises a set of sensor nodes and the direct (peer-to-peer) one-hop communication links between those nodes. Our model assumes two types of sensor nodes, modeled as two disjoint sets: F, the set of moving objects (termed in this paper 'fish'); and C, the set of immobile cordons (checkpoints) at known locations in the transportation network through which the mobile objects (fish) are moving.[1]

To represent the potential for direct (one-hop) communication between cordons and/or fish, we use a time-varying, undirected graph $G_{\mathrm{m}}(t) = (V, E(t))$, termed the *communication graph*, where we require $V = F \cup C$; a discrete set of times T and a time-varying set of edges, $E(t)$, such that, for some time $t \in T$, $E(t) \subseteq V \times V$ is the set of potential one-hop communication links between nearby nodes. The exact communication links for a network will depend on the specific application and technologies used. However, as is common in sensor networks, we assume a relatively sparse graph, where the number of neighbors for a node is relatively small compared with the number of nodes in the network.

3.2. Transportation and connectivity network

The transportation network is modeled as a graph, $G_{\mathrm{t}} = (I, E_{\mathrm{t}})$, where E_{t} is the set of transportation network edges connecting intersections I. Although for simplicity, the transportation network is assumed to be static and undirected, extensions of this model to time-varying and directed networks are straightforward.

In addition, we introduce the concept of a *connectivity network* to represent the relative network locations of cordons in terms of the transportation network connectivity between cordon locations. This relative location is modeled using a *connectivity graph* $G_{\mathrm{c}} = (C, E_{\mathrm{c}})$, where $\{c_1, c_2\} \in E_{\mathrm{c}}$ if and only if there exists a path through the transportation network between cordons c_1 and c_2 that does not traverse any other $c \in C$, where $c \neq c_1 \neq c_2$. Figure 2 illustrates the relationship between the cordon-structured transportation graph (shown in Fig. 2a) and the induced connectivity graph (Fig. 2b). Note that it is not a requirement that cordons be coincident with intersections on the transportation network.

3.3. Sensing capabilities

Cordons are assumed to have the capability of sensing the movement of fish past their location. This models the situation, for example, where riverside receivers track signals from tagged fish, or roadside gantries monitor the passing of a tagged vehicle. To track the direction, sensors would be placed across each edge of the cordon so that both the origin and the destination of the fish can be logged. Alternatively, camera-based systems could be used to monitor the passing of vehicles [44] and log their license plates. Movement events are represented as a function $\mathrm{sense_c} : C \times T \rightarrow \mathbb{N} \times \mathbb{N} \times \mathbb{N} \cup \{\varnothing\}$. Informally, the $\mathrm{sense_c}$ function identifies for each cordon and time any movement of a fish past that cordon, from one incident edge to another.

For example, $\mathrm{sense_c}(v, t) = (1, 102, 103)$ indicates that cordon v detected at time t the fish with ID 1 coming from the direction of the cordon with ID 102 toward the direction of the cordon with ID 103. The distinction between *nodes* (e.g. $v \in V$) and the *identity* of nodes (without loss of generality represented as the set of natural numbers, $1, 2, 3, \ldots \in \mathbb{N}$) is an important one for decentralized spatial algorithms. In a decentralized system, any individual node v can never have direct access to information about another node v' unless that information has previously been explicitly communicated to v. Thus, making a clear distinction between a node and information about its identity helps to avoiding errors in decentralized

[1] We use the term 'fish' as shorthand for our moving objects, because this research was initiated in response to problems faced by a real river health monitoring system deployed in the Murray River, Australia, where the movement of native fish species is tracked using radio frequency transmitters implanted into fish, and monitored using riverside cordons [43].

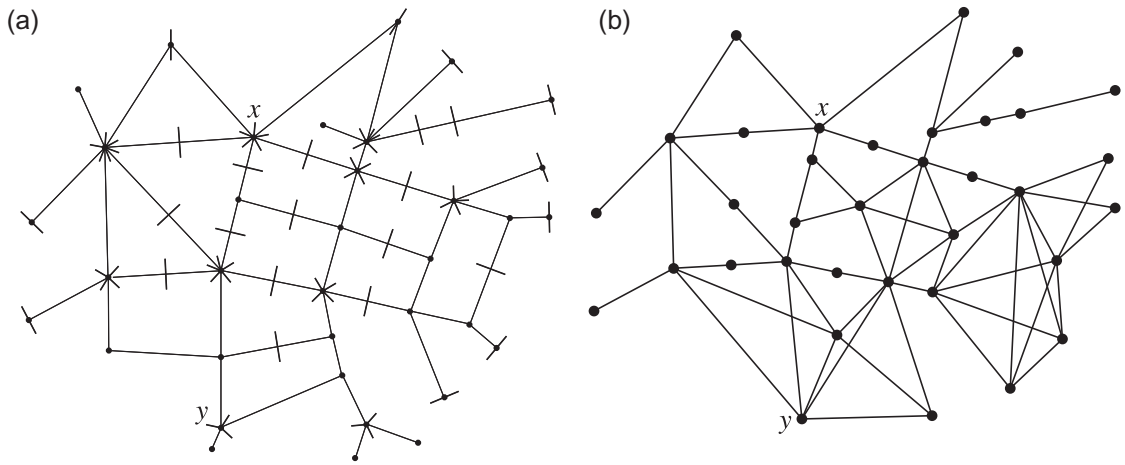

FIGURE 2. Example cordon-structured transportation graph (where small bisecting lines represent checkpoints) (**a**) and induced connectivity graph (**b**), highlighting two cordons x and y.

algorithm design. In cases where we need to highlight this *local* knowledge available to a node, we use the overdot notation (e.g. $\dot{\text{sense}}_c(t)$—said 'local' or 'my' sense_c of t—where $\dot{s}(a)$ is equivalent to $s(v, a)$ and the (local) node $v \in V$ is clear from the context).

$\text{sense}_c(v, t) = \varnothing$ indicates that cordon v detects no movement events at time t. Without loss of generality, the model assumes that at most one fish moves through a cordon at any particular time t. The function $\text{id} : V \to \mathbb{N}$ is used to represent the relationship between nodes and their identities.

In our work, we assume that nodes have unique identities (i.e. the function id is an injection). However, relaxations of this assumption are possible, and in some cases desirable. For instance, in our motivating example of fish monitoring in the Murray Darling River, technical limitations mean that, on occasion, fish may not have unique identifiers. Additionally, errors in a cordon's sensor can lead to incorrect or incomplete records (e.g. in the case of a road network, the camera fails to correctly identify a car's license plate). Such issues are considered in more detail in Section 6.

Note that the sense_c function assumes that as a fish moves past a cordon, that cordon can detect both the edge in the connectivity network the fish is arriving from and the edge it is departing to (in other words, the previous and next cordons a fish has passed and will pass, respectively). In some cases, this may be a reasonable assumption. For example, in the primary motivating application for this paper, monitoring fish movement, cordons are specifically placed at locations in the transportation network that enable them to determine exactly this information (for example, see cordon x in Fig. 2, where there is a 1:1 mapping between out-edges of x in the transportation network and out-edges of x in the connectivity network). However, in other cases, this assumption may be too strong, for example, in many transportation applications. (For example, see cordon y in Fig. 2, where there is no 1:1 mapping between out-edges in the transportation network and in the connectivity network.) The former more restrictive assumption has been made in this paper because it simplifies the algorithm exposition. However, it is straightforward to also support the latter, more general assumption with limited modifications of the approach (specifically the capability of fish to identify cordon IDs as they pass, and communicate this information to the next cordon they pass).

Additionally, note that the transportation network may extend far beyond the extent of the monitoring cordons. Fish that pass beyond the perimeter of the cordons can be accounted for by the introduction of a 'virtual' node to the connectivity graph which is connected to every edge that leads outside the network. In this way, fish entering or leaving the network will be recorded as swimming toward or from this virtual node.

Finally, it may also be that the fish too have the capability of sensing other fish in close proximity (for example, detecting a short range radio handshake or ultrasound 'ping'). The capability of fish of sensing when they move past each other can be represented as a function $\text{sense}_f : F \times T \to \mathbb{N} \cup \{\varnothing\}$. For example, a specific application of this function, $\text{sense}_f(v, t) = 1$, indicates that fish v detected the fish with ID 1 moving past it at time t.

4. DECENTRALIZED ALGORITHMS FOR MONITORING MOVEMENT

Our approach relies on two stages—first maintaining records about movement events at the cordons or fish near where those events occurred; and subsequently querying those records. Thus, the algorithms in this section only provide the capability to maintain decentralized records about movement events; querying is addressed later in Section 6.

4.1. Algorithm 1: communication graph contains connectivity graph

The first algorithm assumes that all cordons that are connected in the *connectivity* network are also connected in the *communication* network (although the converse is not required, not all nodes connected in the communication network need to be connected in the transportation network). Formally, we assume that, for all times $t \in T$, $E_c \subseteq E(t)$ (recall: the communication graph is $G_m(t) = (F \cup C, E(t))$, and the connectivity network is $G_c = (C, E_c)$). This may in some cases be a realistic assumption (e.g. where traffic gantries in a road network enjoy wired communication links with nearby neighbors, or where all cordons are connected by a wide area network). Later algorithms relax this strong assumption, allowing for lesser communication capabilities and greater restrictions to the movement of information. In this simplest case, it is straightforward for any cordon that detects a movement event to store that locally, and communicate this information to its two cordon neighbors (one at the opposite end of the edge the moving object came from, and one at the opposite end of the edge the moving object is now moving on).

As a result, Algorithm 1 is relatively straightforward to construct. Nevertheless, the algorithm is included here because it helps to introduce the algorithm specification style used, based on the approach of Santoro [27, 45]. In short, this approach defines the behavior of individual nodes by specifying a protocol for interaction between nodes, using four components: restrictions, events, actions, states. Restrictions are listed in the header of the algorithm, and we define the assumptions made about the computing environment in which all the nodes operate. Two types of events may occur to nodes. First, trigger events (indicated with the keyword *When*) occur when a node detects the activation of some trigger, such as new sensed data. Second, communication events (indicated with the keyword *Receiving*) occur when a node receives a message via direct communication from a one-hop neighbor. Actions are sequences of operations (i.e. 'programs'), which a node executes in response to an event. Actions are atomic in the sense that they cannot be interrupted by other events. Finally, nodes may respond with different actions to the same event depending on that node's state. A node's state may also change through the course of actions that occur in response to events. For each possible event/state pair, we may define a different action. Event/state pairs that do not appear in an algorithm are assumed to have empty actions (i.e. 'do nothing'). Figure 3 illustrates how trigger events caused by the passing of fish lead to actions that both store event data and initiate communication events in neighboring cordons. These

Algorithm 1 Basic algorithm, where all cordons are directly connected in the communication network to cordon neighbors in the transportation network.

1: Restrictions: Set of nodes $V = F \cup C$, where F is set of mobile fish, C is set of static cordons; sensor function $sense_c : C \times T \to \mathbb{N} \times \mathbb{N} \times \mathbb{N} \cup \{\varnothing\}$; identifier function $id : V \to \mathbb{N}$; connectivity graph $G_c = (C, E_c)$; communication graph $G(t) = (V, E(t))$, where for all t, $E_c \subseteq E(t)$
2: State transition system: $\langle\{$CORD, FISH$\}, \varnothing\rangle$
3: Initialization: All cordons in state CORD, all fish in state FISH
4: Local variables: Table $e = \langle fid : \mathbb{N}, enter : T, exit : T, edge : \mathbb{N}\rangle$, initialized with zero records.

CORD
5: *When* $\hat{sense}_c(now) \neq \varnothing$
6: **let** $(f, c_p, c_n) = \hat{sense}_c(now)$
7: UPDATE e SET $exit = now$ WHERE $fid = f$ AND $exit =$ NULL AND $edge = c_p$
8: INSERT INTO e VALUES $(f, now,$ NULL$, c_n)$
9: **send** (entr, $f, now, \mathring{id}$) to node with ID c_n
10: **send** (exit, $f, now, \mathring{id}$) to node with ID c_p
11: *Receiving* (entr, f, t, i)
12: INSERT INTO e VALUES $(f, t,$ NULL$, i)$
13: *Receiving* (exit, f, t, i)
14: UPDATE e SET $exit = t$ WHERE $fid = f$ AND $exit =$ NULL AND $edge = i$ AND $enter \neq t$

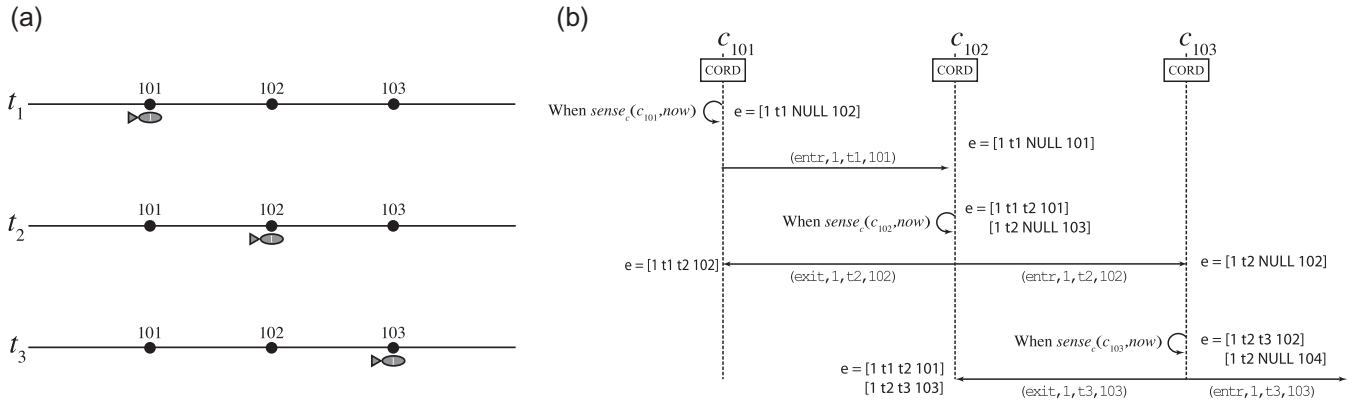

FIGURE 3. Example sequence diagram (**b**) for Algorithm 1, showing how the data change with fish movements (**a**).

communication events in turn lead to actions that also store event data.

4.2. Algorithm 2: disconnected communication graph

Algorithm 1 assumes cordons that are neighbors in the transportation network are also neighbors in the communication network. In many cases, the distances between cordons may be large, requiring wide area network connectivity and this capability may not be available. Our second algorithm relaxes this assumption, instead only assuming communication links between cordons and fish when and where a cordon senses a fish. Restating this formally, we assume that, for any time $t \in T$, cordons $c, c_p, c_n \in C$ and fish $f \in F$, then $\text{sense}_c(c, t) = (\text{id}(f), \text{id}(c_p), \text{id}(c_n))$ implies $\{c, f\} \in E(t)$ and $\{\{c_n, c\}, \{c, c_p\}\} \subseteq E_c$.

In this scenario, the approach of Algorithm 2 is to recruit fish as 'data mules' to physically carry relevant data to the cordon at the opposite end of the edge in the transport network. Thus, unlike Algorithm 1, the fish in Algorithm 2 are active participants in the communication. Informally, there are a number of steps that occur following a movement event, as a fish f passes a cordon c, listed below. These steps are further highlighted by Fig. 4 which shows a simple example execution of this algorithm.

(1) The cordon c sends an exit message to the fish heading toward cordon c_n about all other fish recently exited from their shared edge (stored in table p), before expunging these records.
(2) The cordon c updates its own event table e with the entry record for the fish f for the edge to c_n.
(3) The cordon c stores the new exit record in active table a that the fish f exited the edge to c_p.
(4) On receiving an exit message, the fish f updates its own record of its entry and exit from the edge (stored in table e).
(5) The fish f then sends an exit message back to the cordon containing its own entry and exit record from c_p (stored in table e) along with any exit records (from c_p) it is carrying as a data mule.
(6) The fish f then deletes all data from tables e and a, storing new exit records received from the cordon for c_n in a and its own new entry record in e.
(7) On receiving an exit message from the fish f, the cordon c updates its own event table e with data from the exit records carried by the fish from c_p.

4.3. Algorithm 3: disconnected communication graph with fish–fish communication

Algorithm 3 goes one step further than Algorithm 2, enabling fish to communicate on edges in order to discover new knowledge before either of the cordons. In the case of a VANET, this models the situation where vehicles can communicate with

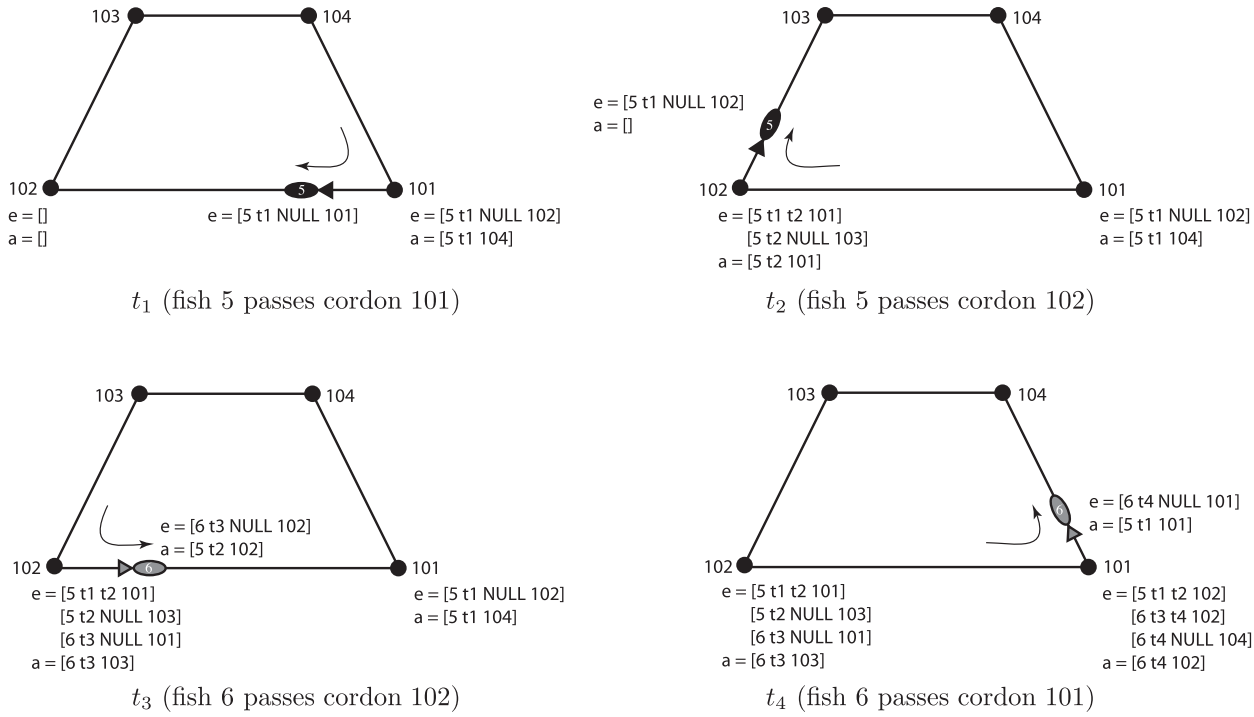

FIGURE 4. Example execution of Algorithm 2 showing how the data tables evolve in response to fish movements. For clarity, only the data tables of cordons 101 and 102, in addition to fish passing these cordons, are shown.

Algorithm 2 Mule algorithm, where fish transport exit records back to cordon.

1: Restrictions: Set of nodes $V = F \cup C$, where F is set of mobile fish, C is set of static cordons; sensor function $sense_c : C \times T \to \mathbb{N} \times \mathbb{N} \times \mathbb{N} \cup \{\varnothing\}$; identifier function $id : V \to \mathbb{N}$; transportation graph $G_c = (C, E_c)$; communication graph $G(t) = (V, E(t))$, where $sense(c,t) = (id(f), x, y)$ implies $\{c, f\} \in E(t)$ and $\{\{c_n, c\}, \{c, c_p\}\} \subseteq E_c$
2: State transition system: $\langle\{$CORD, FISH$\}, \varnothing\rangle$
3: Initialization: All cordons in state CORD, all fish in state FISH
4: Local variables: Event table $e = \langle fid : \mathbb{N}, enter : T, exit : T, edge : \mathbb{N}\rangle$, initialized with zero records; active table $a = \langle fid : \mathbb{N}, exit : T, edge : \mathbb{N}\rangle$.

CORD

5: *When* $\hat{sense}_c(now) \neq \varnothing$
6: **let** $(f, c_p, c_n) = \hat{sense}_c(now)$
7: **let** $p :=$ SELECT *fid*, *exit*, $\mathring{id}$ FROM a WHERE $edge = c_n$
8: **send** (`exit`, p, $\mathring{id}$, *now*) to fish with ID f
9: DELETE FROM a WHERE $edge = c_n$
10: INSERT INTO a VALUES (f, now, c_p)
11: INSERT INTO e VALUES $(f, now,$ NULL$, c_n)$
12: *Receiving* (`exit`, e', a')
13: INSERT INTO e SELECT * FROM e'
14: **for** each (f, t, c) in a' **do**
15: UPDATE e SET $exit = t$ WHERE *fid* $= f$ AND *edge* $= c$ AND *exit* $=$ NULL

FISH

16: *Receiving* (`exit`, p, i, t)
17: UPDATE e SET $exit = t$
18: **send** (`exit`, e, a) to cordon with ID i
19: DELETE FROM e
20: DELETE FROM a
21: INSERT INTO a SELECT * FROM p
22: INSERT INTO e VALUES $(\mathring{id}, t,$ NULL$, i)$

one another as they pass. Formally, the capability of fish to sense when they move past each other is represented as the function $\text{sense}_f : F \times T \to \mathbb{N} \cup \{\varnothing\}$ (see Section 3.3).

Previously, the records for which fish were on an edge in the connectivity graph were stored at the two cordons that bounded that edge. Algorithm 3 additionally generates completed records on fish *in the edge* (which are then later updated at cordons as in Algorithm 2). It does so by expanding each fish's event table, which in Algorithm 2 carried a single record, to carry additional incomplete records for the current edge which can be completed by exchanging records from the active tables of fish they pass. In short, Algorithm 3 operates just as Algorithm 2, except that it additionally allows data mules (i.e. fish—with apologies for the mixed metaphors) to communicate directly with each other, generating complete records about movement events as they move along the edge (i.e. before reaching a cordon), and so reducing latency.

Because these additional movement records are located along the edges of the network, their use may seem limited. This,

Algorithm 3 Extended mule algorithm, where fish transport and exchange exit records.

1: Restrictions: Set of nodes $V = F \cup C$, where F is set of mobile fish, C is set of static cordons; sensor function $sense_c : C \times T \to \mathbb{N} \times \mathbb{N} \times \mathbb{N} \cup \{\varnothing\}$; sensor function $sense_f : F \times T \to \mathbb{N} \cup \{\varnothing\}$; identifier function $id : V \to \mathbb{N}$; transportation graph $G_c = (C, E_c)$; communication graph $G(t) = (V, E(t))$, where $sense(c,t) = (id(f), x, y) \to \{c, f\} \in E(t) \wedge \{\{c_n, c\}, \{c, c_p\}\} \subseteq E_c$, and $sense_f(f,t) = (id(f')) \to sense_f(f', t) = (id(f)) \wedge \{f, f'\} \in E(t)$
2: State transition system: $\langle\{$CORD, FISH$\}, \varnothing\rangle$
3: Initialization: All cordons in state CORD, all fish in state FISH
4: Local variables: Event table $e = \langle fid : \mathbb{N}, enter : T, exit : T, edge : \mathbb{N}\rangle$, initialized with zero records; active table $a = \langle fid : \mathbb{N}, exit : T, edge : \mathbb{N}\rangle$.

CORD

5: *When* $\hat{sense}_c(now) \neq \varnothing$
6: **let** $(f, c_p, c_n) = \hat{sense}_c(now)$
7: **let** $p :=$ SELECT *fid*, *exit*, $\mathring{id}$ FROM a WHERE $edge = c_n$
8: **let** $q :=$ SELECT *fid*, *enter*, *exit*, *edge* FROM e WHERE $edge = c_n$ AND $exit =$ NULL
9: **send** (`exit`, $p, q, \mathring{id}, now$) to fish with ID f
10: DELETE FROM a WHERE $edge = c_n$
11: INSERT INTO a VALUES (f, now, c_p)
12: INSERT INTO e VALUES $(f, now,$ NULL$, c_n)$
13: *Receiving* (`exit`, e', a')
14: INSERT INTO e SELECT * FROM e'
15: **for** each (f, t, c) in a' **do**
16: UPDATE e SET $exit = t$ WHERE *fid* $= f$ AND *edge* $= c$ AND *exit* $=$ NULL

FISH

17: *Receiving* (`exit`, p, q, i, t)
18: **let** $e' :=$ SELECT *fid*, *enter*, t, *edge* FROM e WHERE *fid* $= \mathring{id}$ AND *exit* $=$ NULL
19: **send** (`exit`, e', a) to cordon with ID i
20: DELETE FROM e
21: DELETE FROM a
22: INSERT INTO a SELECT * FROM p
23: INSERT INTO e VALUES $(\mathring{id}, t,$ NULL$, i)$
24: INSERT INTO e SELECT * FROM q
25: *When* $\hat{sense}_f(now) \neq \varnothing$
26: **let** $f = \hat{sense}_f(now)$
27: **send** (`meet`, a) to fish with ID f
28: *Receiving* (`meet`, a')
29: UPDATE e SET $exit =$ (SELECT $a'.exit$ FROM a' WHERE $e.fid = a'.fid$ AND $e.edge = a'.edge$) WHERE EXISTS (SELECT $a'.exit$ FROM a' WHERE $e.fid = a'.fid$ AND $e.edge = a'.edge$)

however, depends on the specific usage scenario. In cases, for example, where the communication range is shorter than assumed by Algorithm 1 but still large enough to cover large sections of a cordon's edges, records from a cordon and its adjacent fish can be combined to provide data that are more complete. In other cases, queries may be injected directly in the network, for example, by users that happen to be nearby in-edge

fish. Indeed, this case is not uncommon in our motivating example of fish tracking in the Murray River, where fish may linger in a stretch of river between cordons for some time, with queries potentially issued from boats or overflying aircraft.

5. EXPERIMENTAL EVALUATION OF MOVEMENT MONITORING

This section briefly evaluates experimentally the three algorithms presented in Section 4, using simulated moving objects moving through a randomized cordon-structured network. All the algorithms were programmed in the NetLogo simulation system [46]. Assuming reliable communication (i.e. that messages are never dropped or corrupted) and reliable sensing (i.e. that fish are never able to pass a cordon, or each other, without being detected), all the algorithms are guaranteed to correctly track movements. As already argued, reliable communication and sensing may be reasonable in many scenarios (e.g. in electronic road tolling). Thus, given that the algorithms are expected to be accurate, the performance of the algorithms was evaluated with respect to the two remaining features of primary interest: *latency* and *scalability*.

5.1. Experiment #1: scalability of algorithms

Our first experiment concerns the comparative scalability of the three algorithms. When considering the scalability of a decentralized algorithm, communication complexity (the amount of communication as a function of input size) is of overriding importance. The most important measures of communication are the total *number* and *length* of messages sent (or received), both for individual nodes (termed load balance) and for the network as a whole.

In this first experiment, the movement patterns of the fish are kept constant and simple. Each fish is assumed to move at constant speed along an edge, and randomly selects with uniform distribution the next edge to follow at a cordon from among all the edges incident with that cordon. Later experiments explore more complex movement patterns, where speeds vary across different fish and for the same fish across different times. The number of fish in the network is expected to be the most important factor in governing the communication complexity (as opposed to the number of cordons), because the system generates communication in response to fish movements. Indeed, preliminary work experimentally confirmed this expectation (Fig. 5). The figure shows how, for a fixed number of fish but increasing numbers of cordons, the number of messages generated by the algorithms remains approximately constant, or even decreases asymptotically in the case of Algorithm 3 (since increasing the size of the network makes fish–fish interaction and communication less likely).

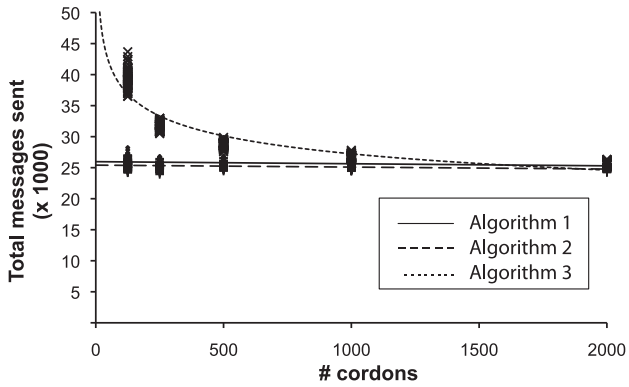

FIGURE 5. Scalability of communication in terms of total number of messages sent with change in numbers of cordons for Algorithms 1–3 (500 fish in all cases).

Thus, it is expected that the communication complexity of algorithms scales linearly with the total number of fish in the network, $O(|F|)$, for Algorithms 1 and 2 (hypotheses H1A and H1B). However, since Algorithm 3 requires fish–fish as well as fish–cordon communication, in the worst case the communication complexity of Algorithm 3 is expected to scale with the square of the total number of fish, $O(|F|^2)$ (hypothesis H1C).

To investigate these hypotheses, each of the three different algorithms was simulated on a randomized transportation network with 50 cordons for 1000 time steps, at each of five different sizes of sets of fish (125, 250, 500, 1000 and 2000 fish, with random initial location). Each experimental run was repeated 100 times, leading to a total of $3 \times 5 \times 100 = 1500$ individual experimental runs. For Algorithm 1, neighboring cordons in the connectivity network were also connected in the communication network (see Section 4.1). For Algorithms 2 and 3, the radius for communication was reduced to near zero, so only fish and cordons that were almost co-located (i.e. as a fish passes a cordon or another fish) could communicate.

Figure 6 shows the results of experiment #1 with overall communication complexity, in terms of the total number of messages sent. As expected, the results support hypotheses H1A and H1B, with a linear regression of the total number of messages communicated using Algorithms 1 and 2 having $R^2 > 0.98$. Similarly, a regression of results for Algorithm 3 indicates a good fit ($R^2 > 0.99$) where the total messages scale approximately in proportion to $|F| + |F|^2$. As expected, this is because Algorithm 3 requires all the messages of Algorithms 1 and 2 (linear in the number of fish) plus fish–fish communication (polynomial in the number of fish), lending support to hypothesis H1C.

When considering the load balance, the maximum communication load of any node in the network provides the worst-case communication complexity. Figure 7 shows the maximum load of any node for 100 randomized runs for each algorithm. All

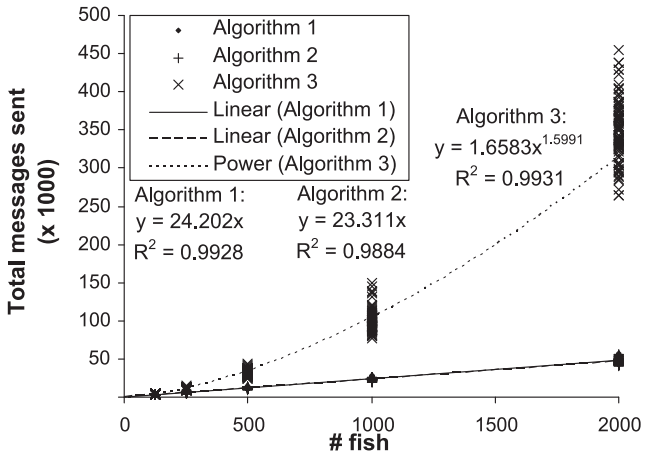

FIGURE 6. Scalability of communication in terms of the total number of messages sent with change in numbers of fish for Algorithms 1–3.

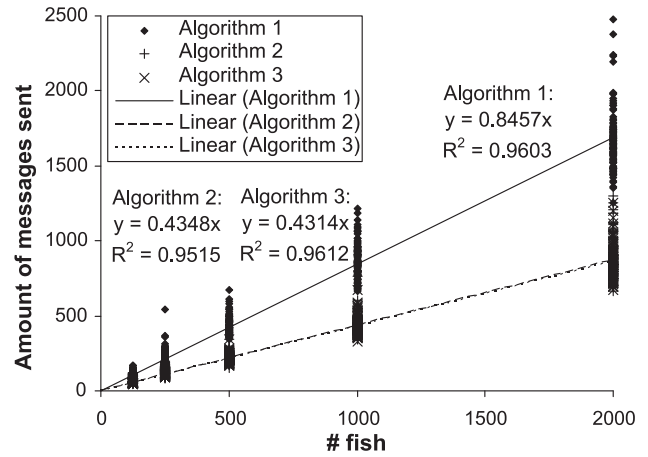

FIGURE 7. Scalability of communication in terms of worst case (maximum) load (number of messages sent) for any node during a run of Algorithms 1–3.

three algorithms exhibit linear $O(|F|)$ load balance with similar constant factors ($R^2 > 0.95$).

It is noticeable that the load balance of Algorithm 3 is no worse than that of Algorithms 1 and 2, even though the overall scalability of Algorithm 3 is worse. This is because the additional fish–fish communication required by Algorithm 3 is shared evenly between all mobile objects; in contrast the fish–cordon communication will depend on the transportation network connectivity/centrality of the cordon.

5.1.1. Scalability of data storage

One further feature of the algorithms is worth highlighting: computational efficiency with respect to data storage. Although scalability of communication is of overriding interest in decentralized spatial computing, the three algorithms do implicitly assume unlimited data storage on cordons and fish.

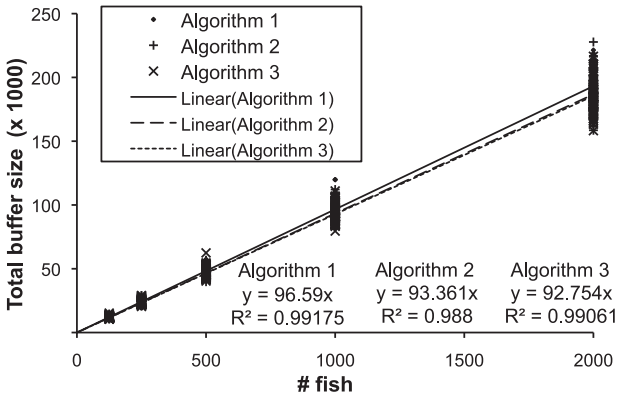

FIGURE 8. Scalability of data storage in total terms of the length of stored data records (buffer size) at cordons for Algorithms 1–3.

In many cases, this assumption may be unreasonable, especially where low-cost, embedded and mobile devices are used as fish. For these cases, it would be trivial to assume a fixed buffer size and then discard information based on spatiotemporal relevance (e.g. the oldest and most spatially distal records are discarded first).

With Algorithm 1, these data are stored in the event tables of the cordons but for Algorithms 2 and 3, data are stored in event and active tables for both cordons and fish. Unlike the event tables of the cordons which increase in size with time, the active tables remove data when they are no longer needed, keeping the size approximately the same. Assuming an arbitrary fixed 1:1 ratio of lengths of the two data types (i.e. the number of bits required to encode the node identifiers $n \in \mathbb{N}$ equals the number of bits required to encode the timestamps $t \in T$) enabled the comparison of the overall amount of data stored in the event and active tables. Because the algorithms distribute stored data with minimal redundancy across the network (each record is stored at exactly two cordons), they are expected to be highly efficient with respect to space complexity at the cordons. Indeed, experiments demonstrated that the total volume of data stored at cordons (in terms of the numbers of stored records) across the network increases linearly with the number of fish (Fig. 8) as might be expected. The worst case load balance for cordons is similarly linear in the number of fish ($R^2 > 0.98$ for all algorithms).

In terms of the scalability of data storage at the fish, the algorithm is also highly scalable, since fish only ever transport data between cordons (and then discard old records). Figure 9 shows the overall space complexity of Algorithms 2 and 3 (Algorithm 1 is omitted as it does not store any information at the fish) in terms of the total number of records stored at the fish. The number of records stored increases linearly with the number of fish for Algorithm 2, but more rapidly for Algorithm 3 (since fish may exchange a longer list of events in-edge). However, this load is relatively evenly spread across the fish in the network, with on average in experiments constant $O(1)$ records stored

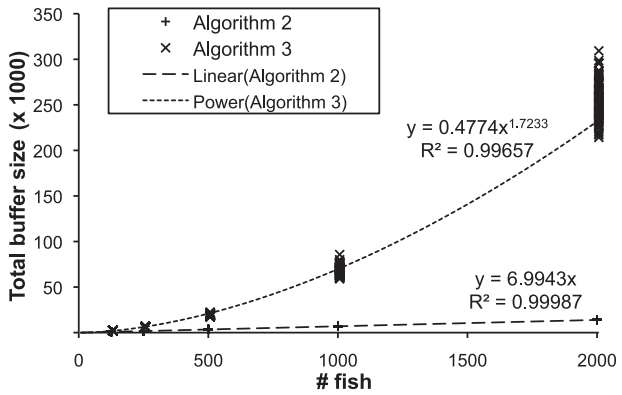

FIGURE 9. Scalability of data storage in terms of the total length of stored data records (buffer size) at fish for Algorithms 2–3.

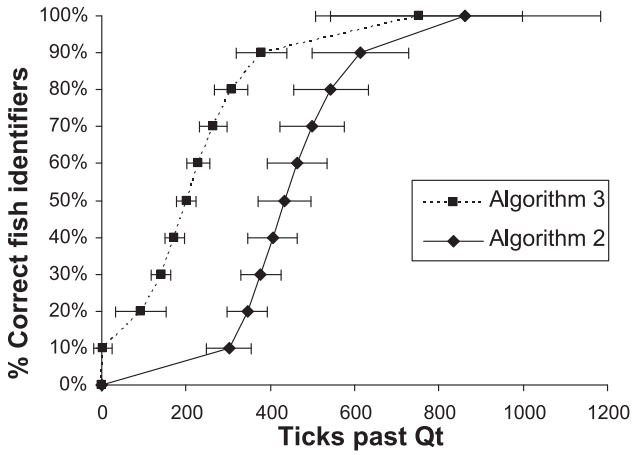

FIGURE 10. Movement latency for Algorithms 2 and 3.

at any fish for Algorithm 2, and $O(|F|^{1.7})$ records stored at any single fish in Algorithm 3.

5.2. Experiment #2: latency of algorithms

Latency concerns the length of the delay between when an event *occurs*, and when that event is correctly *detected* by an algorithm, updating the information stored in all interested nodes (i.e. the cordons at each end of the edge). Specifically, the events of interest are any changes to the identities of fish on an edge. An event is considered to have been detected by an entity (cordon or fish) when it correctly records that change. Note that as our system is decentralized, it is never intended to be the case that *all* entities record an event.

There are two primary causes of the latency associated with our algorithms. The first, termed here *initialization latency*, is caused by the time taken for the algorithm to initialize, i.e. for each fish to have passed at least *one* cordon. Initialization latency is governed primarily by the movement patterns of the fish, and is not a function of the algorithm itself (i.e. any algorithm without prior knowledge of the locations of fish can only start to operate correctly after fish have been detected).

The second cause of latency, termed *movement latency*, is the lag in detecting movement events in Algorithms 2 and 3 resulting from using fish (data mules) to physically transport data to adjacent cordons (recall that *both* cordons that bound an edge of the connectivity graph need to be updated with information about an event, which only occurs when fish have traversed the edge in both directions). Since initialization latency is independent of the algorithm used, only movement latency is used in comparing the algorithms in this section. To discount initialization latency, each individual run of the experiments in this section was allowed to initialize for 250 time steps, found by experimentation to be longer than the maximum initialization latency.

Consider the query 'What fish are on each edge in the network at time t_q?' Because our system includes no predictive capability, this query can be submitted at or after time t_q. Thus, the latency for Algorithms 1 and 2 can be measured as the proportion of fish on an edge that are correctly identified by an algorithm at time $t_q + \delta$, where $\delta \geq 0$. As δ increases, the physical mobility of data mules is expected to lead to a steady increase in this proportion.

Algorithm 1 always has zero latency (i.e. correctly identifies 100% of the fish on any edge at time t_q), and so must necessarily have lesser latency than the other two algorithms. In the worst case, where no fish meet on an the edge, Algorithm 3 will operate exactly like Algorithm 2. However, when fish can exchange events on the edge in Algorithm 3, they should be able to improve on the latency of Algorithm 2.

Thus, our expectation is that movement latency is ordered by algorithm such that Algorithm 1 $<$ Algorithm 3 $\leq$ Algorithm 2. To test this hypothesis (H2, that the latency associated with Algorithm 3 is less than or equal to that associated with Algorithm 2), simulations with 500 fish on a randomized network of 50 cordons were run for each algorithm. At a randomly chosen time t_q, between 0 and 100 time steps after initialization, the query 'What fish are on each edge in the network at time t_q' was tested against the sensor network. The time taken was measured for the proportion of fish correctly identified by this query to reach 10, 20 ... 100%. Figure 10 plots the results of this experiment, comparing the movement latency for Algorithms 2 and 3, using 100 repetitions of each experimental run to generate 95% confidence intervals (see error bars).

Figure 10 confirms our expectation that Algorithm 3 outperforms Algorithm 2 in terms of movement latency, lending support to hypothesis H2.

5.3. Further experiments

Two further variables are of secondary importance to algorithm performance: the movement behavior of fish; and the structure

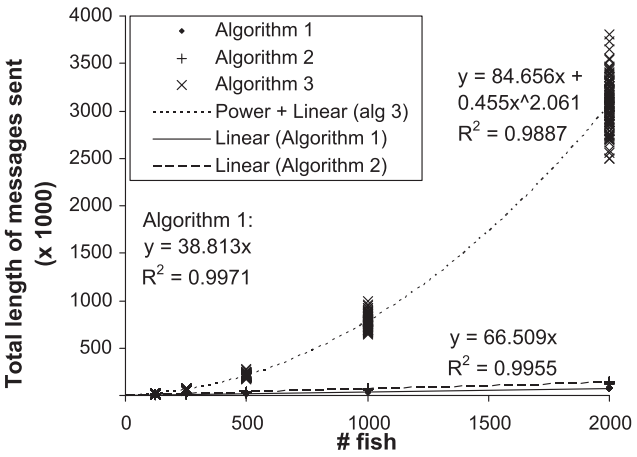

FIGURE 11. Communication complexity in terms of the total number of messages sent for Algorithms 1–3, using objects moving along Lévy walks as opposed to random walks (cf. Fig. 6).

of the transportation network. Since Algorithms 2 and 3 rely on fish as data mules, mobility patterns are also expected to affect the latency associated with these algorithms. Similarly, since the structure of the transportation network constrains the movement of data mules, sparser networks than the unit disk graph (UDG) used in this experiment are expected to affect both efficiency and latency. Without space for a full exposition of the experiments in this area, we summarize two key findings:

(i) Sparser transportation network structures, like the planar transportation networks, substantially decrease the latency of both Algorithms 2 and 3. The decreased transportation network connectivity means that each cordon has a smaller number of network neighbors, and so there is a greater chance of important information being transported more directly to those neighbors.

(ii) More 'realistic' movement patterns, like Lévy walks [47], do not change the overall orders of scalability, but do tend to increase the computational complexity of Algorithms 1 and 2 by a constant factor, primarily due to occasional fast moving objects leading to additional fish–fish interactions. For example, Fig. 11 shows an example of scalability, this time in terms of message length, for the three algorithms with object movement governed by Lévy walks.

6. DECENTRALIZED ALGORITHMS FOR QUERYING MOVEMENT

The algorithms described above provide decentralized mechanisms to *maintain* basic spatiotemporal information about the movements of fish at cordons in the network. Specifically, records for fish on an edge of the connectivity graph are stored at the two cordons that bound that edge (and in Algorithm 3 are also generated in-edge by fish, in advance of reaching both cordons). This information can then be used as the basis for decentralized queries of the network. However, the specific details of any query will depend on a number of factors, including: whether the query is *long-running* (continuously resident in the network), or snapshot (one-off query); whether the query is injected inside the network (e.g. by a human user in close proximity to a particular cordon) or through some gateway node (e.g. initiated remotely via a sensor web); whether the query response is required external to the sensor network (e.g. again, via a sensor-web gateway) or used inside the network (e.g. a sensor/actuator network for controlling river flow based on fish movements) and whether the communication network is connected (e.g. as assumed for Algorithm 1) or may be disconnected, requiring query dissemination using data mules (e.g. as assumed for Algorithms 2 and 3).

As a result, it is beyond the scope of this paper to explore the full range of the different queries that can be supported by our approach, and the design of decentralized algorithms for satisfying these queries. Instead, this section illustrates our approach to such queries, by classifying four main types of queries of interest. A number of existing categorizations of moving object queries already exist (see [23]). For example, categorizations have been proposed based on instantaneous, continuous and persistent queries [48] and distinguishing between location queries (range and nearest neighbor queries) versus trajectory queries (similarity or shape queries) [49]. However, for the purposes of defining queries over our distributed database it is not so much the type of movement that is important, but where the information required is expected to be stored. Consequently, we separate our queries into four types based on (a) whether individual or groups of nodes together hold the information to respond; and (b) whether one single node or multiple nodes are required to coordinate the response:

Q1 *One node can respond individually*: where the information required to satisfy a query is contained entirely in a known node's local database, e.g. 'What fish was on edge (c_1, c_2) over time period $[t_1, t_2]$'?

Q2 *All nodes can respond individually*: where the information required to satisfy a query may be contained in any known node's local database, e.g. 'Which edge had the highest throughput of fish over time period $[t_1, t_2]$'?

Q3 *One node can coordinate a response*: where the information required to satisfy a query is contained in the local databases of a known set of nodes, e.g. 'Which fish followed a known path p through the network'?

Q4 *All nodes must coordinate a response*: where the information required to satisfy a query is contained in the local databases of an unknown set of nodes, e.g. 'Which fish remained together (i.e. followed the same path)'?

Because the information required to satisfy queries of type Q1 is contained in a node's local databases, such queries can be satisfied simply by routing a message from the query source to the known node, and if required routing the query response back to the source. For example, in cases where the communication network is connected and the coordinates of cordons are known, queries of type Q1 can typically be satisfied by using georouting to route the query and response between the query source and known satisfaction node (e.g. [50]).

Similarly, queries of type Q2 can be satisfied using efficient routing structures over the entire network [like tiny aggregation (TAG)d, [51]]. TAG routing requires that a tree overlay graph be added to the network. A range of algorithms for constructing an overlay network routing tree already exist, but result in each node discovering the identifiers of its parent and children in the tree. Answering 'Which edge had the highest throughput of fish over time period $[t_1, t_2]$?' begins with cordons that have no children calculating the local edge with the highest throughput and sending the results to their parents. All other cordons wait until they have received results from all of their children and calculate the throughput of fish on their local edges. The edge with the highest throughput out of the received results and local results is then sent to the parent node. This ensures that the amount of messages sent is proportional to $|C| - 1$. This type of routing would also be useful for summarizing the event tables of the cordons to a central location.

The more challenging queries are those of type Q3 and Q4, where no single node contains the information necessary to satisfy the query. However, in most cases it is to be expected that the inherently autocorrelated structure of geographic space and movement will allow efficient mechanisms for query satisfaction to be defined. For example, Algorithm 4 provides a procedure for satisfying the example query above: 'Which fish followed a specified path p through the network?' Given a path through the connectivity network (specified as a table p of cordon IDs and associated sequence number for each cordon ID), Algorithm 4 simply routes a message from the beginning of that path through to the penultimate node in the path, at each step filtering that node's local movement event database to determine which fish IDs match the movement pattern. Algorithm 4 assumes that cordons connected in the connectivity network are also connected in the communication network (as in Algorithm 1). However, more sophisticated adaptations could also be devised to use fish as data mules for these queries, along similar lines to Algorithms 2 and 3. The number of messages required for the query in Algorithm 4 is $\Theta(|p|)$, where $|p|$ is the length of the path specified in the query (in addition to the application-dependent cost of routing a query or response to some gateway or query node).

A simple strategy to satisfy the type Q4 query 'Which fish remained together?' is to adapt Algorithm 4 to filter fish IDs that match the query constraints starting from *every* node (rather than only over a specified path). Satisfying this query requires additional information, leading to the procedure specified by

Algorithm 4 Type Q3 query to determine the IDs of the set of fish which have traveled some known path $p = \langle cid : \mathbb{N}, o : \mathbb{N} \rangle$, where cid is the cordon ID and o is the order of that cordon in the path (starting from 1).

1: Algorithm extends Algorithm 1, including Restrictions, State transition system, Initialization, and Local variables

CORD
2: *Receiving* (rqst, p)
3: **if** SELECT *count*(*) FROM p WHERE $o = 1$ AND $cid = \mathring{id} > 0$ **then**
4: **let** p_n :=SELECT cid FROM p WHERE $o = 2$
5: **let** f := SELECT *fid, exit* FROM e WHERE $edge = p_n$
6: **send** (path, p, f, 3) to cordon with ID p_n
7: *Receiving* (path, p, f', h)
8: **let** p_n :=SELECT cid FROM p WHERE $o = h$
9: **let** f := SELECT $e.fid, e.exit$ FROM e, f' WHERE $e.fid = f'.fid'$ AND $enter = f'.exit$ AND $edge = p_n$
10: **if** $h + 1 <$ SELECT *count*(*) FROM p **then**
11: **send** (path, p, f, $h + 1$) to cordon with ID p_n
12: **else**
13: **return** f as IDs of fish that followed the path p

Algorithm 5 Type Q4 query to determine what groups of $\geq n$ fish followed the same path of length $\geq l$ cordons with a time lag of $\leq t$.

1: Algorithm extends Algorithm 1, including Restrictions, State transition system, Initialization, and Local variables
2: Local variables: Group table, $g = \langle c : \mathbb{N}, time : \mathbb{T}, fids :$ *(list of fish ids)*$\rangle$, initialized with zero records

CORD
3: *Receiving* (rqst, n, l, t)
4: **let** $p.g$:= SELECT $(fid_1, enter_2, edge_2)$ FROM e, e WHERE $fid_1 = fid_2$ AND $exit_1 = enter_2$
5: **for all** $p.g$ **do**
6: **let** tmp := SELECT * FROM $p.g$ WHERE $enter_2 \geq enter_1$ AND $enter_2 \leq (enter_1 + t)$ AND $edge_1 = edge_2$
7: **if** *count*(*) FROM $tmp \geq n$ **then**
8: **let** $fids$:= SELECT fid FROM tmp
9: **let** $g.n := (\mathring{id}, enter_1, fids)$
10: INSERT INTO g VALUES $(edge_1, enter_1, fids)$
11: DELETE tmp records FROM $p.g$
12: **send** (path, $(g.n), n, l$) to cordon with ID $edge_1$
13: *Receiving* (path, j, n, l)
14: **let** $last.j$:= LAST * FROM j
15: **for all** g **do**
16: **let** tmp := SELECT $fids$ FROM $last.j$ INTERSECT SELECT $fids$ FROM g_i
17: **if** *count*(*) FROM $tmp \geq n$ AND (SELECT $time$ FROM $g_i >$ SELECT $time$ FROM $last.j$) **then**
18: INSERT INTO j VALUES $(\mathring{id}, time_2, fids_2)$
19: **send** (link, j, n, l) to cordon with ID (SELECT c FROM g_i)
20: **if** *count*(*) FROM $j \geq l$ **then**
21: **return** j as record of group movement

Algorithm 5. In this algorithm, each cordon stores groups of at least n fish passing in the same direction within the time period t. This group table also includes the time fish passed the cordon and the ID of the cordon they were swimming toward. Each record is then routed along the network following the same path of the group. Cordons receiving one of these messages will check their group table to find a match with the last entry of the journey table j. This match must have at least n fish in common and the local record must occur after the journey record. A positive match will then be inserted into the j table and sent along to the cordon the group was swimming toward. If the j table has more than l records, then it has passed enough cordons to be considered a group. Like Algorithm 4, by allowing the query to follow the path groups of fish have taken, the algorithm makes use of the inherently autocorrelated structure of the network to efficiently filter out irrelevant data and reduce the amount of messages sent.

The experiments of Section 5 assume reliable sensing and communication, which may not always be the case. Algorithm 6 is a type Q4 query which identifies sensor failure in the cordons. It begins by having each cordon send the event records for each edge to its corresponding neighbor. Cordons receiving this data will compare it with their own and if they are missing some records will declare a sensor failure. Because each cordon must send a message to each of its neighbors, the number of messages required for this query is $\Theta(2|E|)$, where $|E|$ is the number of edges on the network. In the specific example of our target application of river health monitoring, queries such as this can help to detect where and when cordons may be faulty as well as to detect where floods have occurred (enabling fish to establish new transportation links).

Common to all previously discussed algorithms is the assumption that fish possess unique identities. These identities are used to correlate records between cordons (and fish in the case of Algorithms 2 and 3) for both the storage of movement data and the querying of movement data.

Algorithm 6 Type Q4 query to determine if a cordons' sensor has failed.

1: Algorithm extends Algorithm 1, including Restrictions, State transition system, Initialization, and Local variables

CORD

2: *Receiving* (rqst)
3: **for all** $v \in \mathring{nbr}$ **do**
4: **let** $data :=$ SELECT $(fid, enter, exit, \mathring{id})$ FROM e WHERE $edge = v_i$
5: **send** (ping, $data, \mathring{id}$) to cordon with ID v_i
6: *Receiving* (ping, $data', x$)
7: **let** $local.data :=$ SELECT * FROM e WHERE $edge = x$
8: **let** $compare :=$ SELECT * FROM $local.data$ INTERSECT SELECT * FROM $data'$
9: **if** COUNT (*) $compare <$ COUNT (*) $data'$ **then**
10: **return** Sensor failure detected at cordon $\mathring{id}$

Relaxing this assumption would reduce Algorithms 1–3 to a single algorithm that simply records at cordons the time a fish passes a cordon and the direction it is heading toward (i.e. table $e = \langle enter : T, edge : \mathbb{N}\rangle$). Although such data may seem of limited use, certain queries from all query types are still possible, although the information obtained will necessarily be about the *number* of fish passing cordons and not their specific *identities*.

An example type Q1 query would be 'How many fish passed cordon c over time period $[t_1, t_2]$?'. An example type Q2 query would be 'Which cordon had the highest throughput of fish over time period $[t_1, t_2]$?'. A simple type Q3 query would be 'How many fish entered edge (c_1, c_2) over time period $[t_1, t_2]$?'.

For an example type Q4 query, Algorithm 5 can be modified to detect groups identified by the number of fish in the group, leading to Algorithm 7. In this algorithm, groups are assumed to be of constant sizes, and different sizes to all other groups (i.e., the size of a group is taken as a proxy for its identity). As such, the presence of groups which are of equal size or groups which change their size is likely to lead to errors in the number of groups detected by Algorithm 7. The example illustrates how a lack of identity information for nodes can necessitate stronger assumptions while at the same time as potentially reducing algorithm accuracy.

Algorithm 7 Identity restricted type Q4 query to determine what groups of $\geq n$ fish followed the same path of length $\geq l$ cordons with a time lag of $\leq t$.

1: Algorithm assumes that cordons record passing fish in an event table
2: Local variables: Event table, $e = \langle enter : T, edge : \mathbb{N}\rangle$, initialized with zero records; Group table, $g = \langle c : \mathbb{N}, time : \mathbb{T}, count : \mathbb{N}\rangle$, initialized with zero records

CORD

3: *Receiving* (rqst, n, l, t)
4: **for all** e **do**
5: **let** $tmp :=$ SELECT * FROM e WHERE $enter_2 \geq enter_1$ AND $enter_2 \leq (enter_1 + t)$ AND $edge_1 = edge_2$
6: **if** $count$(*) FROM $tmp \geq n$ **then**
7: **let** $g.n := (\mathring{id}, enter_1, count(*))$
8: INSERT INTO g VALUES $(edge_1, enter_1, count(*))$
9: DELETE tmp records FROM $p.g$
10: **send** (path, $(g.n), l$) to cordon with ID $edge_1$
11: *Receiving* (path, j, l)
12: **let** $last.j :=$ LAST * FROM j
13: **for all** g **do**
14: **if** (SELECT $count$ FROM $last.j$ = SELECT $count$ FROM g_i) AND (SELECT $time$ FROM $g_i >$ SELECT $time$ FROM $last.j$) **then**
15: INSERT INTO j VALUES $(\mathring{id}, time_2, count_2)$
16: **send** (link, j, l) to cordon with ID (SELECT c FROM g_i)
17: **if** $count$(*) FROM $j \geq l$ **then**
18: **return** j as record of group movement

6.1. Experimental evaluation of querying

Queries of the type Q4 are expected to be the most complex and computationally expensive to satisfy. For example, Fig. 12 shows the efficiency in terms of messages sent by the Q4-type query shown in Algorithm 5. The number of messages sent is plotted against the number of groups found. Owing to the way this query traverses the network for results, there exists a high positive correlation between the number of fish that satisfy the query and the number of messages required to compute the query response. When only a small number of fish match the query, the filter quickly results in a few or even no messages being required.

Figure 12 also shows the results of operating the query over three different communication network structures connecting cordons (UDG, Gabriel graph and shortest path tree) and for two different types of movement patterns, 'simple' (random walk) and 'turning angle' (a correlated random walk where at each cordon, a moving object randomly chooses the direction of the next cordon to head toward using a Gaussian distribution with its mean centered on the current direction). In general, network structures that are sparser lead to more fish following the same path, and higher numbers of fish that match the filter, and so increased communication overheads. Similarly, movement patterns that tend to more correlated movement (i.e. correlated random walk), and so to more fish matching the filter criteria, are generally less efficient. Figure 12 also shows the Pearson correlation coefficient for each of the six different data experimental runs, indicating moderately high levels of correlation ($r > 0.72$) in all cases except the UDG/simple random walk (where the network structure and movement pattern led to almost no fish matching the query, and so all data points clustered close to the origin). Lévy walks were omitted as movement patterns from this experiment because they also generate very few patterns that satisfy this particular query, and so require almost zero messages.

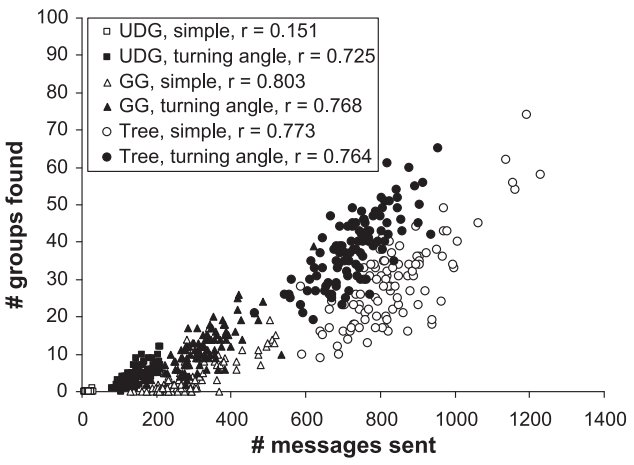

FIGURE 12. Efficiency of Algorithm 5 where $n = 3$, $l = 3$ and $t = 10$. The graph combines the results for two different movement patterns and three different communication graph structures.

7. DISCUSSION AND CONCLUSIONS

This paper has introduced and explored an important and under-researched class of spatial computing problems: tracking and querying moving objects in cordon-structured networks. Such network structures are particularly amenable to emerging monitoring systems, like sensor networks, where tracking stations at fixed locations can be used to monitor the movement of objects in a transportation network.

The paper has shown how the basic movement information required to satisfy spatiotemporal queries about these moving objects can be captured in the network itself, using a decentralized approach. Decentralization is fundamental to sensor networks for several reasons, including where long-range communication in the network is inefficient or impossible. More specifically, our approach stores information about movement events at the cordons that bound sections of the transportation network.

Based on a computational analysis of the algorithms, the experiments presented in Section 5 have illustrated and verified the following properties of the three algorithms for maintaining information about movement events in the network:

(i) Algorithm 1 provides the best performance in terms of both scalability and latency, but only in cases where adjacent cordons in the connectivity network are directly connected;
(ii) Using mobile objects as data mules, Algorithm 2 can operate in networks where cordons are disconnected and cannot communicate directly. Algorithm 2 offers comparable computational efficiency to Algorithm 1, but at the cost of increased movement latency and
(iii) Algorithm 3 can also operate in disconnected cordon-structured networks. By exchanging selected information between mobile objects, Algorithm 3 can improve on the movement latency of Algorithm 2. However, this performance comes at the cost of increased overall computational complexity when compared with Algorithms 1 and 2, although with comparable load balance.

This work has highlighted the trade-offs that must occur in any decentralized approach to monitoring moving objects in cordon-structured networks. Increased communication network connectivity can help reduce latency and increase scalability. However, in cases where the connectivity of the communication network is limited, decentralized algorithms can still be defined based on trade-offs between latency and scalability, with decreases in latency possible only at the cost of some computational efficiency.

Further, the work has demonstrated some of the queries that can be satisfied using this approach. Mechanisms for two major

types of query, where the information required to satisfy a query is contained within the local databases of individual nodes, are trivial to construct, as they only rely on existing routing strategies. Two further types of query, where the information required to satisfy a query is contained within the local databases of multiple nodes, are more challenging to satisfy. However, the examples used show how the inherent spatiotemporal structure of movement can provide an efficient structure for organizing the necessary computation.

Current work is developing further efficient decentralized algorithms for the spatiotemporal query types discussed in Section 6. Longer term objectives are to be able to compute decentrally meaningful movement patterns, such as *flocks* [52, 53], *convoys* [54] or *leadership* [55]. These works employ trajectory-based data but can be adapted to work with data from cordon-structured networks. Future work will further address the potential effects of interactions between mobile objects [56], and of changing transportation network characteristics (such as network *capacity* or *impedance*), which may also influence movement [14].

A range of avenues for further work are suggested by this research, including the following:

(i) *Effects of constrained storage capacity*: Because the algorithms distribute stored information efficiently across the network, with minimal replication (e.g. storing movement events only at cordons incident with an edge of the connectivity network; see Section 5.1.1), storage space at each node is not expected to be a major constraint in most applications. However, future work might also investigate instances where substantial constraints to storage capacity do exist. A natural mechanism for dealing with storage constraints is to assume a fixed buffer size, and then discard information based on spatiotemporal relevance (e.g. the oldest and most spatially distal records are discarded first). Depending on the severity of storage constraints, such constraints are expected to impact both the latency and accuracy of queries.

(ii) *Further query mechanisms*: In addition to investigating a wider range of queries across the different types, further research might also address the broader range of issues identified in Section 6 (e.g. long-running versus snapshot queries, query origin and query destination). For example, as in Algorithm 1, the query mechanism in Algorithm 4 assumes that the connectivity network is a subgraph of the communication network. Similar approaches to those used in Algorithms 2 and 3 (i.e. using fish as data mules) might also be applied in querying data, adapting Algorithm 4 to stronger constraints to the movement of information.

(iii) *Predictive capabilities*: The work in this paper is concerned purely with historical queries, and does not address the prediction of future movement patterns. Given that Algorithms 4 and 5 can track the movements of individual fish and groups, respectively, data from these algorithms could be used to extrapolate future movement patterns. Potentially, predictive capabilities might also have computational implications (for example, improving communication scalability by directing communication resources toward known or likely future movement patterns and network configurations). For example, probabilistic models, such as Bayesian networks, are important for modeling uncertain future states.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the helpful support and input from Jarod Lyon and Adrian Kitchingman at the Arthur Rylah Institute (ARI), Melbourne, Australia, and from Harvey Miller, University of Utah, for highlighting links to the transportation literature. The authors are also grateful for the constructive comments of the anonymous reviewers.

FUNDING

This work was supported by an Australian Research Council (ARC) Future Fellowship [grant number FT0990531], an ARC Discovery Project [grant number DP120103758] and a Commonwealth Scientific and Industrial Research Organization (CSIRO) OCE research scholarship [grant number LEX17098].

REFERENCES

[1] Duckham, M. and Reitsma, F. (2009) Decentralized environmental simulation and feedback in robust geosensor networks. *Comput. Environ. Urban Syst.*, **33**, 256–268.

[2] Laube, P., Duckham, M. and Wolle, T. (2008) Decentralized Movement Pattern Detection Amongst Mobile Geosensor Nodes. In Cova, T., Beard, K., Goodchild, M. and Frank, A.U. (eds), *Geographic Information Science*, Lecture Notes in Computer Science 5266, pp. 199–218. Springer, Berlin.

[3] Umer, M., Kulik, L. and Tanin, E. (2008) Kriging for Localized Spatial Interpolation in Sensor Networks. In Ludäscher, B. and Mamoulis, N. (eds), *Proc. 20th Int. Conf. Scientific and Statistical Database Management (SSDBM)*, Berlin, Lecture Notes in Computer Science 5069, pp. 525–532. Springer.

[4] Bachrach, J., Beal, J. and McLurkin, J. (2010) Composable continuous-space programs for robotic swarms. *Neural Comput. Appl.*, **19**, 825–847.

[5] Ban, X., Herring, R., Margulici, J. and Bayen, A.M. (2009) Optimal Sensor Placement for Freeway Travel Time Estimation. In Lam, W.H.K., Wong, S.C. and Lo, H.K. (eds), *Transportation and Traffic Theory 2009: Golden Jubilee*, pp. 697–721. Springer.

[6] Giudice, N.A., Walton, L.A. and Worboys, M. (2010) The Informatics of Indoor and Outdoor Space: A Research Agenda.

Proc. 2nd ACM SIGSPATIAL Int. Workshop on Indoor Spatial Awareness, New York, pp. 47–53. ACM.

[7] Murray Darling Basin Authority (2011) *Fish 'n' Chips: Why Do We Tag Fish? Native Fish Strategy Fact Sheet*. http://www.mdba.gov.au/files/publications/MDBA-13057-Fish-n-Chips-FS_web.pdf (accessed August 27, 2012)

[8] Gudmundsson, J., van Kreveld, M. and Speckmann, B. (2007) Efficient detection of patterns in 2D trajectories of moving points. *GeoInformatica*, **11**, 195–215.

[9] Trajcevski, G., Wolfson, O., Zhang, F. and Chamberlain, S. (2002) The Geometry of Uncertainty in Moving Objects Databases. In Jensen, C., Saltenis, S., Jeffery, K., Pokorny, J., Bertino, E., Böhn, K. and Jarke, M. (eds), *Advances in Database Technology (EDBT 2002)*, Lecture Notes in Computer Science 2287, pp. 145–161. Springer.

[10] Turchin, P. (1998) *Quantitative Analysis of Movement: Measuring and Modelling Population Redistribution in Animals and Plants*. Sinauer, Sunderland, MA.

[11] Beal, J. and Schantz, R. (2010) A Spatial Computing Approach to Distributed Algorithms. *45th Asilomar Conf. Signals, Systems, and Computers*, Pacific Grove, CA

[12] Miller, H.J. (1999) Potential contributions of spatial analysis to geographic information systems for transportation (gis-t). *Geogr. Anal.*, **31**, 373–399.

[13] Ho, H., Wong, S., Yang, H. and Loo, B.P. (2005) Cordon-based congestion pricing in a continuum traffic equilibrium system. *Transp. Res. A: Policy Pract.*, **39**, 813–834.

[14] Zhang, X. and Yang, H. (2004) The optimal cordon-based network congestion pricing problem. *Transp. Res. B: Methodol.*, **38**, 517–537.

[15] Goh, M. (2002) Congestion management and electronic road pricing in singapore. *J. Transp. Geogr.*, **10**, 29–38.

[16] Mehmood, M.A., Kulik, L. and Tanin, E. (2008) Autonomous Navigation of Mobile Agents Using RFID-Enabled Space Partitions. *Proc. 16th ACM SIGSPATIAL Int. Conf. Advances in Geographic Information Systems*, New York, pp. 21:1–21:10. ACM.

[17] Lim, A. and Zhang, K. (2006) A Robust 'ID-Based Method for Precise Indoor Positioning. In Ali, M. and Dapoigny, R. (eds), *Advances in Applied Artificial Intelligence*, Lecture Notes in Computer Science 4031, pp. 1189–1199. Springer.

[18] Schmid, F., Richter, K.-F. and Laube, P. (2009) Semantic Trajectory Compression. In Mamoulis, N., Seidl, T., Pedersen, T., Torp, K. and Assent, I. (eds), *Advances in Spatial and Temporal Databases*, Lecture Notes in Computer Science 5644, pp. 411–416. Springer, Berlin/Heidelberg.

[19] Pfoser, D. (2002) Indexing the trajectories of moving objects. *IEEE Data Eng. Bull.*, **25**, 3–9.

[20] Trajcevski, G., Bischof, Z. and Scheuermann, P. (2009) Range Queries for Mobile Objects in Wireless Sensor Networks. *Proc. 17th ACM SIGSPATIAL Int. Conf. Advances in Geographic Information Systems*, New York. ACM.

[21] Xiong, X., Elmongui, H., Chai, X. and Aref, W. (2007) PLACE*: A Distributed Spatio-Temporal Data Stream Management System for Moving Objects. *Proc. IEEE Int. Conf. Mobile Data Management (MDM)*, Mannheim, Germany, pp. 44–51. IEEE.

[22] Gedik, B. and Liu, L. (2004) MobiEyes: Distributed Processing of Continuously Moving Queries on Moving Objects in a Mobile System. In Bertino, E., Christodoulakis, S., Plexousakis, D., Christophides, V., Koubarakis, M., Böhm, K. and Ferrari, E. (eds). *Advances in Database Technology – EDBT 2004*. Lecture Notes in Computer Science. Vol. 2992, pp. 523–524. Springer, Berlin/Heidelberg.

[23] Güting, R.H. and Schneider, M. (2005) *Moving Objects Databases*. Morgan Kaufmann, San Francisco.

[24] Duckham, M. and Bennett, R. (2009) Ambient Spatial Intelligence. In Gottfried, B. and Aghajan, H. (eds), *Behavior Monitoring and Interpretation—BMI*, pp. 319–335. IOS Press, Amsterdam, Netherlands.

[25] Lynch, N. (1996) *Distributed Algorithms*. Morgan Kaufmann, San Mateo, CA.

[26] Dulman, S., Havinga, P. and Hurink, J. (2002) Wave Leader Election Protocol for Wireless Sensor Networks. *Proc. 3rd Int. Symp. Mobile Multimedia Systems and Applications*, Delft, Netherlands, pp. 43–50. Citeseer.

[27] Santoro, N. (2007) *Design and Analysis of Distributed Algorithms*. Wiley, NJ.

[28] Nagpal, R., Shrobe, H. and Bachrach, J. (2003) Organizing a Global Coordinate System from Local Information on an Ad Hoc Sensor Network. *Information Processing in Sensor Networks*, Palo Alto, CA, pp. 553–553. Springer.

[29] Dil, B., Dulman, S. and Havinga, P.J.M. (2006) Range-Based Localization in Mobile Sensor Networks. In Römer, K., Karl, H. and Mattern, F. (eds), *Proc. 3rd European Workshop on Wireless Sensor Networks (EWSN)*, Lecture Notes in Computer Science 3868, Zurich, Switzerland, pp. 164–179. Springer.

[30] Skraba, P., Fang, Q., Nguyen, A. and Guibas, L. (2006) Sweeps Over Wireless Sensor Networks. *Proc. 5th Int. Conf. Information Processing in Sensor Networks (IPSN)*, Nashville, TN, pp. 143–151. ACM.

[31] Beal, J., Bachrach, J., Vickery, D. and Tobenkin, M. (2008) Fast Self-healing Gradients. *Proc. 2008 ACM Symp. Applied Computing*, Fortaleza, Brazil, pp. 1969–1975. ACM.

[32] Beal, J., Bachrach, J., Vickery, D. and Tobenkin, M. (2009) Fast Self-stabilization for Gradients. *IEEE Int. Conf. Distributed Computing in Sensor Systems*, Los Angeles, CA, pp. 15–27. Springer.

[33] Chen, W. and Cai, S. (2005) Ad hoc peer-to-peer network architecture for vehicle safety communications. *IEEE Commun. Mag.*, **43**, 100–107.

[34] Little, T. and Agarwal, A. (2005) An Information Propagation Scheme for VANETs. *Proc. IEEE Intelligent Transportation Systems*, September, Vienna, Austria, pp. 155–160. IEEE.

[35] Nadeem, T., Shankar, P. and Iftode, L. (2006) A Comparative Study of Data Dissemination Models for VANETs. *Proc. 3rd Annual Int. Conf. Mobile and Ubiquitous Systems—Workshops*, July, San Jose, CA, pp. 1–10. IEEE.

[36] Kesting, A., Treiber, M. and Helbing, D. (2010) Connectivity statistics of store-and-forward intervehicle communication. *IEEE Trans. Intell. Transp. Syst.*, **11**, 172–181.

[37] Hartenstein, H. and Laberteaux, K. (2008) A tutorial survey on vehicular ad hoc networks. *IEEE Commun. Mag.*, **46**, 164–171.

[38] Shah, R.C., Roy, S., Jain, S. and Brunette, W. (2003) Data MULEs: modeling and analysis of a three-tier architecture for sparse sensor networks. *Ad Hoc Netw.*, **1**, 215–233.

[39] Grossglauser, M. and Vetterli, M. (2006) Locating mobile nodes with EASE: learning efficient routes from encounter histories alone. *IEEE/ACM Trans. Netw.*, **14**, 457–469.

[40] Shirani-Mehr, H., Banaei-Kashani, F., Shahabi, C. and Zhang, L. (2011) A Case Study of Participatory Data Transfer for Urban Temperature Monitoring. In Tanaka, K., Frölich, P. and Kim, K.-S. (eds), *Proc. W2GIS*, Berlin, Lecture Notes in Computer Science 6574, pp. 50–63. Springer.

[41] Wolfson, O. and Xu, B. (2004) Opportunistic Dissemination of Spatio-Temporal Resource Information in Mobile Peer-to-Peer Networks. *Proc. 1st Int. Workshop on P2P Data Management, Security and Trust (PDMST'04), DEXA Workshops 2004*, Zaragoza, Spain, pp. 954–958. IEEE.

[42] Nittel, S., Duckham, M. and Kulik, L. (2004) Information Dissemination in Mobile Ad-Hoc Geosensor Networks. In Egenhofer, M., Freksa, C. and Miller, H. (eds), *Geographic Information Science*. Lecture Notes in Computer Science 3234, Washington, MD, pp. 206–222. Springer, Berlin.

[43] Koehn, J., Nicol, S., McKenzie, J., Lieschke, J., Lyon, J. and Pomorin, K. (2008) Spatial ecology of an endangered native Australian Percichthyid fish, the trout cod Maccullochella macquariensis. *Endang. Species Res.*, **4**, 219–225.

[44] Lee, S. and Baik, H. (2006) Origin-Destination (od) Trip Table Estimation Using Traffic Movement Counts from Vehicle Tracking System at Intersection. *32nd Annual Conf. IEEE Industrial Electronics, IECON 2006*, Paris, France, pp. 3332–3337. IEEE.

[45] Duckham, M., Nussbaum, D., Sack, J.-R. and Santoro, N. (2011) Efficient, decentralized computation of the topology of spatial regions. *IEEE Trans. Comput.*, **60**, 1100–1113.

[46] Willensky, U. (1999) *Netlogo. Center for Connected Learning and Computer-Based Modeling*. Northwestern University, Evanston, IL. http://ccl.northwestern.edu/netlogo/.

[47] Sims, D.W. *et al.* (2008) Scaling laws of marine predator search behaviour. *Nature*, **451**, 1098–1102.

[48] Sistla, A., Wolfson, O., Chamberlain, S. and Dao, S. (1997) Modeling and Querying Moving Objects. *Proc. 13th Int. Conf. Data Engineering*, Birmingham, UK, pp. 422–432. IEEE.

[49] Agarwal, P.K. *et al.* (2002) Algorithmic issues in modeling motion. *ACM Comput. Surv.*, **34**, 550–572.

[50] Karp, B. and Kung, H.T. (2000) GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. *Proc. 6th Annual Int. Conf. Mobile Computing and Networking (ACM/IEEE MobiCom)*, Boston, MA. ACM.

[51] Madden, S., Franklin, M.J., Hellerstein, J.M. and Hong, W. (2002) TAG: A Tiny Aggregation Service for Ad-Hoc Sensor Networks. *5th Symp. Operating System Design and Implementation (OSDI)*, Boston, MA.

[52] Benkert, M., Gudmundsson, J., Hübner, F. and Wolle, T. (2008) Reporting flock patterns. *Comput. Geom.*, **41**, 111–125.

[53] Laube, P., van Kreveld, M. and Imfeld, S. (2004) Finding REMO: Detecting Relative Motion Patterns in Geospatial Lifelines. In Fisher, P.F. (ed.), *Developments in Spatial Data Handling, Proc. 11th Int. Symp. Spatial Data Handling*, Leicester, UK, pp. 201–214. Springer, Berlin.

[54] Jeung, H., Yiu, M.L., Zhou, X., Jensen, C.S. and Shen, H.T. (2008) Discovery of convoys in trajectory databases. *Proc. VLDB Endow.*, **1**, 1068–1080.

[55] Andersson, M., Gudmundsson, J., Laube, P. and Wolle, T. (2008) Reporting leaders and followers among trajectories of moving point objects. *GeoInformatica*, **12**, 497–528.

[56] Camp, T., Boleng, J. and Davies, V. (2002) A survey of mobility models for ad hoc network research. *Wirel. Commun. Mob. Comput.*, **2**, 483–502.