Pursuit Control over Wireless Sensor Networks using Distance Sensitivity Properties

V. Kulathumani*, A. Arora† and S. Ramagiri*
*Dept. of Computer Science and Electrical Engineering, West Virginia University
Email: Vinod.Kulathumani@mail.wvu.edu, sramagir@mix.wvu.edu
†Dept. of Computer Science and Engineering, Ohio State University
Email: anish@cse.ohio-state.edu

Abstract

In this paper, we focus on a control based surveillance application using a wireless sensor network in which information from the network is used to actively guide a mobile agent leading to eventual pursuit of one or more evaders in a large region. We adopt a co-design approach in which the application strategy for pursuit control is designed hand-in-hand with the network protocols resulting in guaranteed system performance. We exploit distance sensitivity as a locality concept in designing a scalable pursuit control system. Specifically, we show that eventual pursuit is satisfied if information about an evader is available to the pursuing agent with error, latency and frequency that decrease linearly with distance from the evader. Then, we design network algorithms for delivering snapshots of the system that satisfy these distance sensitivity properties. Finally, we close the loop by showing how the network snapshot service can be instantiated to satisfy the requirements of the pursuit control application.

I. Introduction

Over the past decade, wireless sensor networks have been used in many surveillance applications for collaboratively tracking objects of interest in a large secured area [1]–[4]. Most of these surveillance applications have been monitoring based, where information conveyed by the network is used to observe the activities of tracked objects and classify them into different types. In this paper we focus on a control based surveillance application in which information from the network is used to actively guide a mobile agent leading to eventual pursuit [5]–[8]. Specifically, we consider a distributed tracking application where one or more pursuer agents are required to *eventually* catch one or more evaders in a large region. An underlying sensor network is deployed in the region to detect and track the pursuers and evaders in the network. The tracking application executes on the pursuer object and uses the sensor network to get the desired information about the evader objects.

Designing pursuit control applications using a sensor network is a challenging task because the target track information has to be acquired and communicated over multiple hops on an unreliable wireless medium prone to collision and fading effects. Therefore information can be error-prone, can have unpredictable delays or even be lost. To address this challenge, we adopt a co-design approach in which the application strategy for pursuit control is designed hand-in-hand with the network protocols resulting in guaranteed system performance. Such a co-design is needed because there exists a tension between the application requirements and what the network can supply. In case of the eventual pursuit application, if the pursuer agents have perfect information

about the entire network instantaneously, designing the control strategy becomes simple. However imposing such a requirement on the wireless network will result in a lot of contention and therefore end up decreasing the overall system performance. Therefore, the application needs to identify weaker network requirements that still result in provable convergence properties. These conditions impose a specification for the network layer in terms of network abstractions and these abstractions have to be implemented using appropriate middleware services. In this paper we follow this co-design approach for the eventual pursuit tracking application and make the following contributions.

Contributions: (1) Given that obtaining perfect information about all objects is infeasible using a wireless sensor network, we first determine sufficient conditions on the error, latency and rate of information about the evader being tracked in order to satisfy *eventual* catch. Specifically, we show that *eventual* catch is satisfied if the error in the estimate of distance to the evader decreases linearly with distance between pursuer and the evader, if the rate at which this information is supplied to the evader decreases linearly with distance and if the staleness in the information supplied decreases linearly with distance, where the constants of proportionality depend on the relative speeds of the pursuer and the evader. (2) We capture the requirements on latency, rate and error imposed by the application on the network as three different network abstractions namely distance sensitive -latency, -rate and -error respectively. In order to implement these abstractions, we design a middleware service that periodically delivers the global snapshot of the system to all nodes in the network where the snapshots satisfy the required *distance sensitivity* properties. (3) We complete our co-design by showing how system performance can be maximized by adapting the network snapshot service to changing application requirements.

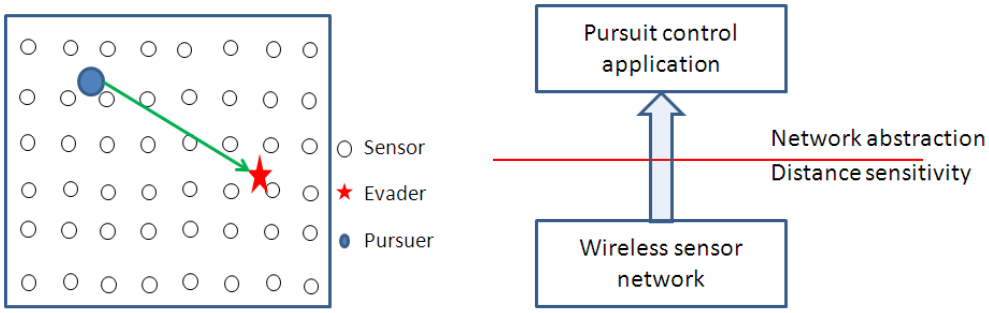


Fig. 1. (a)The objective of the pursuit control system is for the pursuer object to *catch* one or more evader objects in the system. The pursuer object is guided towards the evader using information provided by a multi-hop wireless sensor network. (b) The pursuit control application runs on the pursuer object and is co-designed with the wireless sensor network. Sufficient conditions for information delivery by the wireless sensor network are determined that satisfy pursuit control requirements. These conditions are represented by a network abstraction, namely distance sensitivity. A wireless sensor network service is then designed to implement this abstraction.

Related work: The mathematical theory of differential games has been applied to pursuit-evasion games and have been extensively studied over the past several decades [9]–[13], starting with a seminal work by Issac [14]. Pursuit-evasion games have been traditionally modeled as

continuous state perfect information systems in which the global state of the game is available to the players with no delays. By way of contrast, in this paper we study pursuit control in a network with communication constraints.

In [15] and [16], pursuit-evasion game applications using sensor networks have been explored and a series of algorithms are devised to coordinate the pursuers so as to minimize the time-to-capture of all evaders. Sensor network measurements are assumed to be fused at local stations to produce track information [17]. Evader assignment and pursuer control strategy are calculated at the base station and then communicated to the pursuer agents. However network effects such as latency and loss in communicating this information to the pursuer agents are not considered.

Cao *et al* [6], [7] have characterized the conditions on network latency and information update periods required to achieve an optimal tracking by modeling the application as a differential game and obtaining Nash equilibrium conditions for pursuer objects to *catch* evaders as far away from an asset as possible. In this paper, apart from latency and update periods, we also take into account the impact of error or loss in resolution of the information on pursuit control.

There has been a lot of interest in Graph Laplacian based techniques for distributed estimation and tracking using sensor networks [18], [19]. The focus of these papers has been on designing distributed Kalman filters for estimation in sensor networks by local exchange of messages. The building blocks for these Kalman filters are consensus filters that calculate the average sensor measurement after every iteration which is then fed into the Kalman filter [18]. A significant difference in our model for tracking objects is that we do not seek consensus. In our model for tracking, pursuer objects use the global state to make their decision and execute their next step; however they do not wait for a synchronized, unique global state. Each pursuer receives a global snapshot of the system in which the staleness (and error) in the state of each node is different. Yet we obtain sufficient conditions for the tracking application to meet its requirements.

Communicating periodic global state snapshots is a well studied problem in distributed systems [20] and consistency, timeliness and reliability have been the main design considerations in those studies. But efficiency becomes essential when considering periodic snapshots for resource constrained wireless sensor networks. To the best of our knowledge algorithms for delivering periodic snapshots across a wireless sensor network have not been studied before. Recently, fractional cascading has been used for sensor networks as an efficient storage mechanism [21], [22]. Data is first stored at multiple resolutions across the network, which is then used to efficiently answer aggregate queries about a range of locations without exploring the entire area. In contrast, we have considered a model where information is generated and consumed on an ongoing basis. Accordingly we describe push based services that regularly deliver to subscribers snapshots of the network in a pipelined manner. By providing snapshots with not just distance sensitive -error but also -latency and -rate, we achieve compression and thereby efficiency.

Outline of the paper: In Section 2, we derive the sufficient conditions for successful pursuit (that result in eventual catch) and translate these to network abstractions. In Section 3, we describe

a snapshot service for wireless sensor networks that implements these network abstractions. In Section 4, we show how the snapshot service can be adapted to pursuit control requirements. In Section 5, we highlight the robustness of our snapshot service using simulations in JProwler. We conclude in Section 6 and point directions for future work.

II. Pursuit control design

Application model: One or more pursuer objects are required to *eventually* catch all evader objects spread over a bounded region. The pursuer objects are assisted by a wireless sensor network that provides the state (location) of evader objects to the pursuer objects. We assume that every pursuer is assigned to track at most one evader at a time. The pursuit controller resides in each pursuer object and uses the evader location provided by the sensor network to converge on the evader's location. In our analysis, we consider the case where one pursuer p has been assigned to an evader e and this assignment holds until the evader is caught. Note that an eventual catch in this scenario is sufficient to show that all evaders will be eventually caught. A *catch* is said to occur when when the distance d_{pe} between a pursuer p and an evader e assigned to p is smaller than a constant ϵ.

We assume the existence of a reliable object detection and association service that assigns a unique identifier to every object in the network. The problem of detecting objects in the network and uniquely associating them with previous detections is orthogonal to the problem of supplying this information in a timely and reliable manner for pursuit control and guaranteed convergence, which is the focus of this paper. Detection and association services can be implemented in a centralized [5] or distributed [23] fashion; the latter approach would suit integration with the distributed pursuit control strategy that we discuss in this paper.

System dynamics: Let the pursuer and evader speeds be constant, denoted by v_p and v_e respectively. Let $X_p(t)$ denote the location $[x_p(t), y_p(t)]^T$ of a pursuer at any time t. Let $u_p(t)$ denote the control action executed by a pursuer at time t that dictates the direction of motion at time t (since speed is constant). Thus, we have

$$\dot{X}_p(t) = u_p(t) \tag{1}$$

Similarly let $X_e(t)$ denote the location $[x_e(t), y_e(t)]^T$ of a pursuer at any time t. Let $y_p(t)$ denote the state of the evader that is supplied to the pursuer at time t by the underlying sensor network service. In our model, these updates are provided only at certain instants of time and let $\{T_k\}$ denote the set of times when a new update is available to a pursuer. At all times that an update is available, note that y_p provides an estimate of the evader's location at that time because the network delivers the state with certain latency and error. At other times, y_p reflects the evader's location at the last time that an update was available. Thus, we have,

$$y_p(t_k) = \hat{X}_e(t_k) \qquad (\forall t_k \in \{T_k\}) \tag{2}$$

$$y_p(t_j) = \hat{X}_e(t_{j-}) \qquad (\forall t_j \notin \{T_k\}) \tag{3}$$

In Eq. 3, t_{j-}, corresponds to the largest timestamp, less than or equal to t_j, that belongs to $\{T_k\}$. The pursuit control strategy ($u_p(t)$) is simply to move along a straight line towards the most recent available location of the evader with a speed v_p. At all times $t_k \in \{T_k\}$, the pursuer changes its trajectory towards $\hat{X_e}(t_k)$ and at other times continues on the straight line towards the previous estimate. Let $< \overrightarrow{P,Q} >$ denote the unit vector in the direction from point P to point Q. Thus, we have:

$$u_p(t_k) = v_p(< \overrightarrow{X_p(t_k), \hat{X}_e(t_k)} >) \qquad (\forall t_k \in \{T_k\}) \tag{4}$$

$$u_p(t_j) = v_p(< \overrightarrow{X_p(t_j), \hat{X}_e(t_{j-})} >) \qquad (\forall t_j \notin \{T_k\}) \tag{5}$$

Let $d_{pe}(t)$ denote the distance between the pursuer and evader at time t. Let $\delta(t)$ denote the staleness in the state of e supplied to p at time t. Let $I(t)$ denote the maximum interval after time t at which location of e can be provided to p. Let $\alpha = \frac{v_p}{v_e}$, where $\alpha > 1$.

Theorem II.1. *Evader e will be eventually caught by pursuer p if there exists constant $k > \frac{\alpha+1}{\alpha-1}$ and time T_o such that the following conditions hold at all $t > T_o$:*

$G1: \quad |X_e(t) - \hat{X}_e(t)| < \frac{d_{pe}(t)}{k}$

$G2: \quad \delta(t) < (\frac{d_{pe}(t)}{v_e})(1 - \frac{\alpha+k+1}{k\alpha})$

$G3: \quad I(t) < d_{pe}(t)(\frac{k+1}{kv_p})$

Proof:

Consider time $t > T_o$, such that $t \in \{T_k\}$. If the maximum error in the location of the evader is denoted as $\frac{d_{pe}}{k}$ (where conditions on k are yet to be derived), the actual location X_e of evader e at time t is within a ball of radius $\frac{d_{pe}(t)}{k}$ around $\hat{X_e}(t)$. Therefore, the maximum distance between $X_p(t)$ and $\hat{X_e}(t)$ is bounded by the following inequality.

$$dist(X_p(t), \hat{X}_e(t)) < d_{pe}(t)(\frac{k+1}{k}) \tag{6}$$

At time t, the action of the pursuer is to move towards $\hat{X_e}(t)$. Let us assume that next information about the evader is available to p only after reaching $\hat{X_e}(t)$. It follows using Eq. 6 that this interval ($I(t)$) is equal to the maximum time taken to travel from $X_p(t)$ to $\hat{X_e}(t)$ and is bounded by the following inequality.

$$I(t) < d_{pe}(t)(\frac{k+1}{kv_p}) \tag{7}$$

Note that there is a staleness of $\delta(t)$ in the information about the evader available at time t. Additionally, a time $I(t)$ is taken to travel towards the estimated location. During these times, the evader can change its location with a speed of v_e. Using Eq. 7, we have the following inequality.

$$d_{pe}(t + I(t)) < v_e d_{pe}(t)(\frac{k+1}{kv_p}) + \frac{d_{pe}(t)}{k} + v_e\delta(t) \tag{8}$$

In order for eventual catch, we require that $d_{pe}(t+I(t)) < d_{pe}(t)$. Using this, we get the following inequality.

$$\delta(t) < (\frac{d_{pe}(t)}{v_e})(1 - \frac{\alpha + k + 1}{k\alpha}) \tag{9}$$

Note that for $\delta(t) > 0$, we require that $\alpha + k + 1 > k\alpha$. Using this we get the following equation:

$$k > \frac{\alpha + 1}{\alpha - 1} \tag{10}$$

■

Thus, we have shown that conditions $G1$, $G2$ and $G3$ of Theorem II.1 and Eq. 10 are sufficient for the distances between pursuer and evader to monotonically decrease at each successive time instant belonging to set $\{T_k\}$ and to eventually reach ϵ. We now determine the time required for eventual *catch*.

Lemma II.2. *Let D_0 denote the initial distance between the pursuer and evader at time t_0 : $t_o \in \{T_k\}$. If conditions for eventual* catch *specified in Theorem II.1 are satisfied, then the time, τ_c, required by a pursuer to eventually catch an evader is bounded by the following inequality.*

$$\tau_c < \frac{D_0(k+1)}{kv_p} log_\theta(\frac{D_0}{\epsilon}) \tag{11}$$

Proof: Let θ denote the ratio of distance between pursuer and evader in two successive time instants that belong to set $\{T_k\}$. Using Eq. 8 and Eq. 9, we have:

$$\theta = \frac{d_{pe}(t)}{d_{pe}(t + I(t))} > \frac{1}{\frac{k+1}{k\alpha} + \frac{1}{k} + \frac{1}{v_e} - \frac{\alpha+k+1}{\alpha k v_e}} \tag{12}$$

Recall that a *catch* is said to occur when when the distance between pursuer and evader reduces to ϵ. Hence $log_\theta(\frac{D_0}{\epsilon})$ such update intervals will be needed to reduce D_0 to ϵ. Noting from Eq. 7 that $I(t_0) < \frac{D_0(k+1)}{kv_p}$, we get Eq. 11. ■

III. Distance sensitive snapshot service

In this section, we design a network middleware service that can be used for supplying information about evader objects to pursuer objects while meeting the sufficient conditions for pursuit control stated in Theorem II.1.

A. Distance sensitive snapshots

Definition III.1 (Snapshot S^V)**.** *A snapshot $S^V : V \rightarrow \Re \times \Re$ of a set of nodes V is a mapping from each node $i \in V$ to a state value ($X \in \Re$) and a timestamp ($t \in \Re$) associated with that state value.*

Let $X_i(S^V)$ denote the state of node $i \in V$ in snapshot S^V and let $t_i(S^V)$ denote the timestamp of the state of node $i \in V$ in snapshot S^V. A consistent snapshot [20] is one where the timestamps associated with the state of each node in the network are the same. In order to be feasible to implement in a resource constrained wireless sensor network, we relax the consistency requirement for a snapshot S^V along the dimensions of latency, error and periodicity of delivery.

Let τ denote the current time. The *staleness* ($\gamma_i(S^V)$) of the state of node i in snapshot S^V is defined as: $\gamma_i(S^V) = \tau - t_i(S^V)$, where $i \in V$. We now consider a generalization where state values do not necessarily correspond to the same instant of time but their staleness enjoys a distance sensitive property. Let $d(i,j)$ denote the distance between nodes i and j in the network.

Definition III.2 (Snapshots with distance sensitive latency)**.** *A snapshot S^V received by a node j has* distance sensitive latency *if* $\gamma_i(S^V) = O(d(i,j)) \quad \forall i : i \in V$

Definition III.3 (Snapshots with distance sensitive error)**.** *A snapshot S^V received by a node j has* distance sensitive error *if* $e_i(S^V) = O(d(i,j)) \quad \forall i : i \in V$*, where ($e_i(S^V)$) is the error of the state of node i in snapshot S^V.*

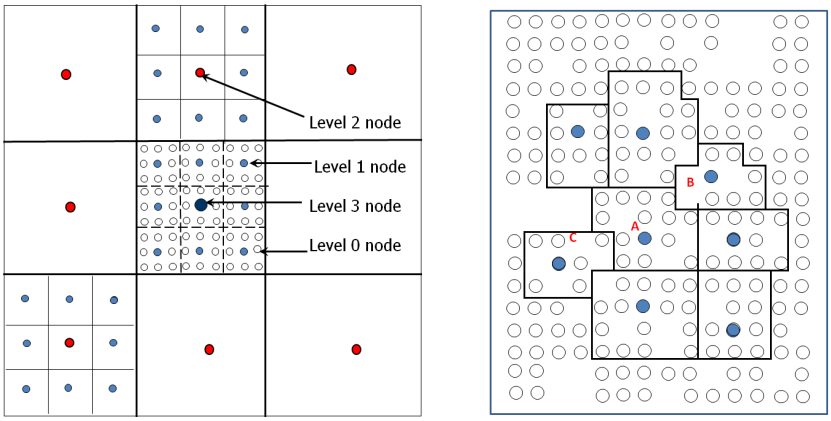
Definition III.4 (Snapshots with distance sensitive rate)**.** *A node j receives a snapshot S^V with* distance sensitive rate *if* $\zeta_i = O(d(i,j)) \quad \forall i : i \in V$ *where ζ_i is the rate at which state of node i is updated in the snapshot S^V is received by j.*

Note that the concept of distance sensitive rate is orthogonal to that of distance sensitive latency. In the latter staleness in the state received decreases with distance but fresh information arrives at the same rate at all nodes, where as in the former, the state of nearby nodes is reported more often than farther nodes. For reasons of exposition, in this paper we present a version of the algorithm (hereby referred to as *DSS*) that provides snapshots which are distance sensitive in latency and error, and then updates these snapshots at the highest rate for all nodes. We note that, in order to meet the requirements of the pursuit control application, it would have been sufficient to progressively decrease the update rates at larger distances. The refinement to DSS that adds distance sensitivity in rate can be found in [24].

B. Network model

We consider a sensor network consisting of N nodes that induce a connected network where each node can communicate at W bits per second. Nodes are assumed to know their geographic location. Let V denote the set of nodes in the network. The nodes may have an irregular communication range and may be arbitrarily deployed in a bounded region. Nodes are assumed to be synchronized in time. Let m denote the number of bits allocated to represent the state of each node in the network.

DSS depends on an underlying hierarchical partitioning of the nodes into clusters of increasing sizes and the implementation of a tree data structure for routing on those clusters. We partition the network into a hierarchical one with a number of levels $L_{max} = O(log(N))$ with the clusters at level 0 representing the individual nodes in the network. In order to provide this clustering, we use a clustering service *FLOC* [25]. The algorithm starts by first creating a 1 level clustering [25] which is then iterated with clusterheads at each new level resulting in a hierarchical clustering [26]. The algorithm for clustering is distributed, local and finishes in $O(1)$ time. More specifically, the properties provided by the clustering service are listed below.



(a) (b)

Fig. 2. (a) Hierarchical clustering for a regular grid network. All clusters at a given level are of same size (given by property $C2$). (b) Hierarchical clustering in a network with *holes*, non-uniform density and irregular communication range. All clusters at a given level have a minimum size (given by $C2$), but nodes beyond that minimum distance and up to a maximum distance (specified by $C3$) from the clusterhead can belong to the same cluster in order to be locally self-stabilizing. All distances correspond to communication *hop* distances as radio range may not be same for all nodes. Level 1 clusterhead A and its 7 neighboring level 1 clusterheads are shown in the Figure. All nodes with 1 hop of the clusterhead belong to the cluster and nodes that are 2 hops from the clusterhead may belong to the cluster.

$C1$: A unique node is designated as clusterhead at each level.

$C2$: All nodes within distance $\frac{3^r-1}{2}$ from a level r clusterhead belong to that cluster.

$C3$: The maximum distance of a node from its level r clusterhead is $3^r - 1$.

$C4$: There exists a path from each clusterhead to all nodes in that cluster containing only nodes belonging to that cluster.

$C5$: At all levels $r : 1 \leq r < L_{max}$, there is at least one and at most η_b neighboring level r clusters for each level r clusterhead and there exists a path between any two neighboring clusterheads.

We note that such a clustering can tolerate the addition and deletion of nodes in the network in $O(1)$ time and in a *locally self-stabilizing* manner by which the changes to the clustering are contained within a constant distance from the location of the added or deleted node and do not propagate network-wide causing a global reassignment of clusters and clusterheads. In order to maintain the local self-stabilizing properties in the presence of node additions and deletions, it is shown in [25] that we cannot impose a requirement on cluster sizes at given level to be exactly the same. A factor of 2 in the allowable cluster size is necessary to ensure that any changes in topology are resolved in a local manner. This is reflected in property $C3$ of the clustering service. Thus all nodes within distance $\frac{3^r-1}{2}$ from a level r clusterhead are required to belong to that cluster but the maximum distance of a node from its level r clusterhead can be $3^r - 1$.

Let η_0 denote the maximum number of level 0 nodes within a level 1 cluster. By default, the distances stated in properties $C1 - C5$ denote communication hop distances. In Fig. 2(a), we

show an example of the clustering for the scenario where the network contains uniformly spaced nodes with equal communication range and separated by a one hop communication range (which we hereby refer to as a regular grid network). In this scenario, all clusters belonging to the same level are of the same size, each cluster at a given level has exactly 8 neighbors at that level, and the distances in properties $C1 - C5$ reduce to geometric distances. In Fig. 2(b), we show an example of the clustering within a single level when the network radio range is irregular and may contain regions of higher and lower density. In this case, *FLOC* does not yield clusters of the same size any more. Instead there is a minimum and maximum size for each cluster as indicated by properties $C2$ and $C3$. Moreover, the distances stated in properties $C1 - C5$ are now hop distance (as radio ranges may be non-uniform and nodes are not equi-distant) and not necessarily geometric distance.

Let $j.L$ denote the highest level for which j is clusterhead. Let h_j^r denote the clusterhead for node j at level r. Thus $h_j^r = j(\forall r : 0 \leq r \leq j.L)$. Let $\{N_j^r\}$ denote the set of neighbors for node j at level r. Note that by property $C5$, there are at most η_b neighbors at each level for each node in the network. We implement virtual trees on the clusters at each level. Let $tree(r, j)$ denote a level r tree formed with j as root and spanning all nodes in the level r cluster of j and all level r clusters that are its neighbors. Let $j.in(r, y)$ denote j's parent towards root y on $tree(r, y)$. Let $j.out(r, y)$ denote the set of j's descendants on $tree(r, y)$. Let $M(r, j)$ denote the level r summary computed by a level r clusterhead j. In Fig. 3, a level 1 tree rooted at j is shown as an illustration for a grid network.

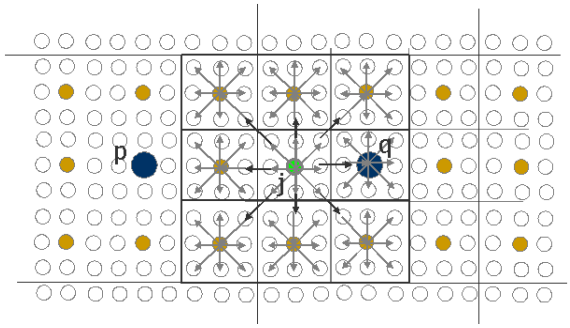


Fig. 3. Illustrating level 1 tree rooted at j. The level 1 tree at j spans all level 0 members in j's level 1 cluster and all its neighboring level 1 clusters.

C. Algorithm for distance sensitive snapshots

We present the algorithm for distance sensitive snapshots by first describing three building blocks for *DSS*: (1) aggregation, (2) scheduling and (3) storage. We then use these building blocks to present the actions executed at each node.

1) Aggregation: In this subsection, we describe how the location of objects being tracked is encoded in the state of individual nodes in the network and how this information is aggregated at higher levels in the hierarchy. Let n_e denote the number of objects in the network, numbered from $1, .., n_e$.

An object e is said to reside at node i if object e is closest to node i compared to all other nodes in the network and node i is then responsible for encoding the location of e. Accordingly, the area over which the network is deployed is divided into Voronoi cells with each node responsible for

encoding the location of objects within its own cell. Let Ψ_i denote the Voronoi cell corresponding to node i. Each node i then divides Ψ_i into g equi-sized regions. If an object e lies within Ψ_i, then node i marks the location of e as one of the g regions and $\lceil log_2(g) \rceil$ bits are used to encode the location. Thus, the state of each node i in the network at a given time consists of $\lceil log_2(g) \rceil$ bits per object in the network and contains the location of an object if the object resides at i. A total of $m = n_e(\lceil log_2(g) \rceil)$ bits are allocated to represent the state of each node at any instant. The number of regions (g) used to represent the location of an object within a cluster corresponds to the granularity of an object's location within a cluster. For instance, if $g = 1$, then only the presence or absence of each object within a cluster will be known by this encoding strategy, but the error in the location of the object provided by a level l clusterhead will be equal to the maximum radius of a level l clusterhead as the object can be anywhere in the cluster.

At higher levels in the hierarchy, a clusterhead j at each level $r : 0 < r \leq L_{max}$ is responsible for aggregating the location of all objects that lie within the area enclosed by the Voronoi cells of all level 0 nodes in its cluster at level r, and we denote this area as $A_{r,j}$. To perform this aggregation, each clusterhead j divides $A_{r,j}$ into the same number of regions g. The aggregation function at a level l clusterhead then maps the location of an object from one of the $\eta_b g$ regions provided by the η_b neighboring level $r-1$ clusterheads into one of the g regions of the level r cluster.

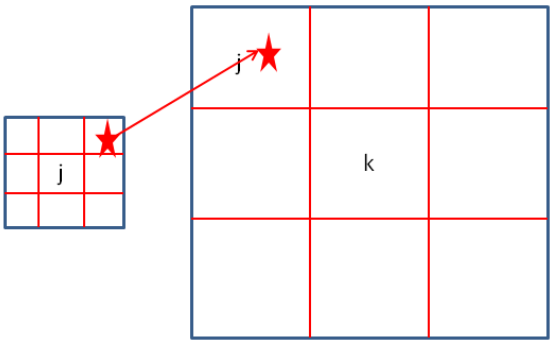


Fig. 4. Aggregation: The evader (shown as star) is located in one of $g = 9$ regions in area $A_{l,j}$ by level l clusterhead j. At the next level in the hierarchy, clusterhead k marks the location of the evader as one of $g = 9$ larger regions inside $A_{l+1,k}$, giving rise to an error in location that is proportional to cluster size.

Let $M_e(r, j)$ denote the location of an object e in a level r summary $M(r, j)$ computed by node j. Let $M_e(r, j) = \perp$ if e does not lie within $A_{r,j}$. Note that $M_e(r, j)$ will correspond to one of g regions within $A_{r,j}$ if $M_e(r, j) \neq \perp$. Now consider area $A_{r+1,k}$ corresponding to a clusterhead k at level $r+1$ such that $A_{r,j} \subset A_{r+1,k}$. Let $A_{r+1,k} \triangleleft M_e(r, j)$, denote one of the g regions within $A_{r+1,k}$ that contains the centroid of the region $M_e(r, j)$. In Fig. 5, we state the aggegation function AF, that computes the next higher level aggregate $M(r, j)$ using the summaries $M(r - 1, y)$ computed by the level $r - 1$ neighbors of node j, and the level $r - 1$ summary $M(r - 1, j)$ that is computed by node j itself.

Computing M(r,j) at node $j : j.L > 0$**, where** $0 < r \leq j.L$
for each y such that $y \in \{N_j^{r-1}\}$
  for each object e such that $M_e(r-1, y) \neq \perp$
    $M_e(r, j) = A_{r,j} \triangleleft M_e(r-1, y)$
  end for
end for
for each object e such that $M_e(r-1, j) \neq \perp$
    $M_e(r, j) = A_{r,j} \triangleleft M_e(r-1, j)$
end for

Fig. 5. Aggregation function AF.

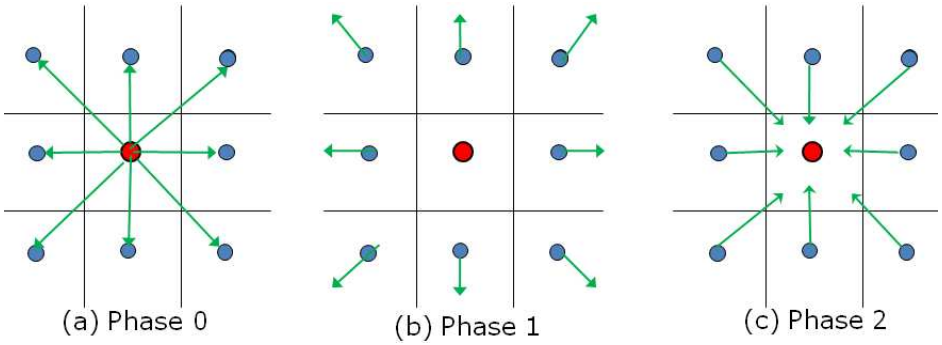


Fig. 6. Transmission in each round is divided into 3 phases. At the end of each round, information is exchanged between neighboring level 1 clusterheads. (a) Phase 0 corresponds to slot 0. Nodes with $j.L > 0$ compute the summaries of clusters for which they are leaders and transmit these to descendants on the respective trees. They also forward information received in previous round on trees that they belong to but are not the root, and forward this information to descendants on the respective trees. (b) In phase 1 all level 0 nodes take turns to transmit information heard in phase 0 to descendants on all trees that they belong to. This results in information propagating outwards from the level 1 cluster that they belong to. (c) In phase 2 all level 0 nodes take turns to transmit information heard in phase 1 to descendants on all trees that they belong to. This results in information coming inwards to their respective level 1 clusterheads.

We note that at higher levels of the hierarchy, information is aggregated into the same number of bits m, and error in the location of an object being tracked increases (proportional to the maximum size of a cluster). We now state the following proposition.

Proposition III.1. *The error in the state of a node i in a level r summary corresponds to the maximum error in encoding the location of any object residing at i in a level r summary and is bounded by $O(3^r)$.*

2) Schedule: We schedule the nodes to transmit in rounds. A *round* is defined as a unit of time in which information is exchanged between a level 1 clusterhead and all of its η_b neighboring level 1 clusterheads. Each round is divided into multiple slots in 3 phases (illustrated in Fig. 6). In the first slot (phase 0), all nodes with $j.L > 0$ transmit. In the remaining slots, all level 0 nodes in each cluster transmit twice, once in each phase. The second transmission by a node within a round (phase 2) takes place after all its neighbors have transmitted at least once. The messages that are transmitted during these slots are stated in Fig. 7. A simple non-interference schedule that satisfies these constraints in a grid network (with 8 neighbors per node) is one where all level 0 nodes take turns. In general, each round will consist of a constant number of slots, η_s that depends on the schedule chosen.

3) Local storage: The snapshot received by a node j at the end of each round consists of $M(x, y)$ received by node j in that round for each x, y such that $j \in tree(x, y)$. Each node stores only the most recent snapshot. Thus each node j's local storage contains the following summaries:

- $M(x, h_j^x) \qquad (\forall x : 0 \leq x \leq L_{max})$
- $M(x, y) \qquad (\forall x, y : (0 \leq x \leq L_{max}) \wedge (y \in N_j^x))$

4) Actions at each node:

In this subsection, we describe the actions executing at each node. These actions are stated in Fig. 7. In slot 0 of each round nodes with $j.L > 0$ compute the aggregate $M(r,j)$ for each level r that they are a clusterhead of based on the level $r-1$ aggregate received in the previous round, using the aggregation function described previously. The computed aggregate at each level r is then transmitted to the descendants on the respective tree rooted at j, i.e., $j.out(r,j)$. In addition, for each $tree(x,y)$ that j belongs to but is not a root of, $M(x,y)$ heard in previous round from $j.in(x,y)$ is transmitted to $j.out(x,y)$. In their respective phase 1 and phase 2 slots, nodes at level 0 simply transmit $M(x,y)$ as heard in the previous phase of that round from $j.in(x,y)$ to $j.out(x,y)$.

Actions for node $j : j.L > 0$
In slot 0 of each round:
  $\forall r : 1 \leq r \leq j.L$
    Compute: $M(r,j)$ using AF
    Send: $M(r,j) \rightarrow j.out(r,j)$
  $\forall x,y : j \in tree(x,y)$
    Send: $M(x,y) \rightarrow j.out(x,y)$
Actions for node $j : j.L = 0$
In each transmission slot for j determined by the scheduling block:
  $\forall x,y : j \in tree(x,y)$
    Send: $M(x,y) \rightarrow j.out(x,y)$
Receive Actions at node j
Upon receiving $M(x,y)$ from node i
  Store $M(x,y)$ if $i = j.in(x,y)$

Fig. 7. Actions at each node in DSS for distance sensitive snapshots.

Thus, aggregates computed at each level are transmitted outwards to descendants along a tree. This is sufficient for a level r node to compute aggregates from level $r-1$ nodes, because a tree at level $r-1$ extends up to all level 0 nodes in neighboring level $r-1$ clusters. And one of the neighboring level $r-1$ node is a level r node. Thus, when a computed aggregate by any node is being dispersed to nodes in its own cluster and the neighboring clusters, it is also being sent *in* to a higher level node to compute an aggregate. In Fig. 3, nodes p and q are level 2 clusterheads. Note that the level 1 tree rooted at j reaches the level 2 clusterhead q that j belongs to. We now analyze the latency and error in the snapshots provided by *DSS*.

D. Analysis

Let s_w denote the duration of each transmission slot in algorithm DSS. Let $j.\gamma_i$ denote the staleness in the state of a node i in the snapshot delivered by algorithm DSS at node j.

Theorem III.1. *In* DSS, $j.\gamma_i = O(d)$ *where* $d = dist(i,j)$.

Proof: Consider a node p at level r. To compute a summary at level r, level $r-1$ summaries are needed. The maximum distance between p and its neighboring level $r-1$ clusters is $2(3^{r-1})$ (From property $C3$). Thus, a level r summary is computed based on a level $r-1$ summary that was generated $2\eta_s s_w 3^{r-1}$ time ago, since latency between each pair of level 1 nodes (length of a *round*) is η_s slots. A level $r-1$ summary is computed based a level $r-2$ summary, and so on until level 0. Upon summation, we see that the staleness of a level 0 (individual node) state information in a level r summary bounded by $\eta_s s_w 3^r$. Now, the maximum distance traveled by a level r summary is 3^{r+1} and the latency is bounded by $\eta_s s_w 3^{r+1}$. Therefore, the maximum

total staleness in the state of any node i in a level r summary is bounded by $4\eta_s s_w 3^r$. Now, the minimum distance d between j and i for which a level r summary is the smallest level that contains information about i is 3^{r-1}. Expressing the maximum staleness in terms of the minimum distance d, we get the following equation:

$$j.\gamma_i = 12\eta_s s_w d = O(d) \tag{13}$$

■

Theorem III.2. *In DSS, the error of state of a node i in a snapshot received at node j is $O(d)$ where $d = dist(i, j)$.*

Proof: Recall from proposition $III.1$ that the maximum error in the state of a node i provided by a level r clusterhead is bounded by $O(3^r)$. The minimum distance between i and j at which j gets a level r summary of i but not a level $r-1$ summary of i is 3^{r-1}. Thus, the error in the state of i in a snapshot received at j is $O(d)$, where $d = dist(i, j)$. ■

Thus *DSS* provides snapshots at each node that satisfy properties of distance sensitive latency and error. These snapshots are updated at each node for all levels at a regular rate with an interval of one *round* length ($\eta_s s_w$).

Theorem III.3. *In DSS, the average communication cost in the network to deliver a snapshot of one sample from each node to all nodes is $O(N * log(N) * m)$.*

Proof: Consider a node j at any level r, where $0 \leq r \leq L_{max}$. From $C5$, we note that at most η_b trees at levels $1..L_{max}$ can pass through each node where $L_{max} = O(log(N))$. There are at most η_0 neighbors for j at level 0 and so at most η_0 level 0 trees can pass through j. Hence, the maximum message length needed per slot in algorithm DSS is $O(mlog(N))$ bits.

To deliver a snapshot with a sample from each node, every node thus communicates $O(m * log(N))$ bits N times. And to deliver a snapshot with y samples from each node, every node communicates $O((N + y) * (m * log(N)))$ bits, since all the y samples are pipelined. Hence, if y is large and $y = \Omega(N)$, the average communication cost at each node to deliver a snapshot of a sample from each node to all nodes is $O(m * log(N))$. The average communication cost over N nodes is $O(N * (m * log(N))$.

Note that if y is small, for instance, if there is only one sample from each node, then the communication cost is $O(N * n * (m + log(n/m))$. Pipelining the delivery of snapshots improves the average communication cost to $O(N * (m * log(n))$. ■

Theorem III.4. *In DSS, the memory requirement per node is $O(log(N) * m)$ bits.*

Proof: Recall that the data structure maintained at each node is the most recent value of $M(x, y)$ received by i for each $tree(x, y)$ that i belongs to. Nodes do not buffer information to be forwarded over multiple rounds. The maximum number of trees through any node is

$O(log(N))$, with m bits of information flowing along each tree. The result follows. ■

E. Adding distance sensitivity in rate

In order to add distance sensitivity in rate, we make the following refinement to algorithm DSS (which we denote as algorithm $DSSR$). Consider a level 0 node with $j.L = r$. A level r summary is computed by this node once every 3^r rounds based on the most recent level $r-1$ summaries it receives. Instead of transmitting this summary $M(r,j)$ in one round, it is now transmitted in slot 0 of each round with $max(1, \frac{m}{3^r})$ bits per round. Thus, a level r summary is sent over $min(3^r, m)$ rounds. The actions for forwarding nodes remain the same except for the change that each node now only receives a fraction of $M(x,y)$ in every round for each $tree(x,y)$ that it belongs to, and it forwards only that fraction in the next round. This refinement (Algorithm DSSR) improves the energy efficiency of the snapshot service while decreasing the periodicity of updates linearly with distance. Below, we state results on latency and rate when using this refinement.

Let $j.\nu_i$ denote the interval between two successive updates received by node j from node i.

Theorem III.5. *In $DSSR$, $j.\nu_i$ is $O(d)$, where $d = dist(i,j)$.*

Proof: Consider levels $0 \leq r \leq log(m)$. Note that a complete level r snapshot is sent every 3^r rounds in a pipelined manner. Thus every 3^r rounds, a level r snapshot is received by a node. The time corresponding to 3^r rounds is $3^r \eta_s s_w$.

Recall that in order to update the local data structure of j, the state of a node i is updated using summary $M(x',y')$ where x' is the lowest level which contains information about k. Now the minimum distance between j and i for which a level r summary is the smallest level that contains information about j is 3^{r-1}.

The maximum interval between when a node j receives the state of node i is given by the following equation:

$$\begin{aligned} j.\nu_i &= \frac{3^r \eta_s s_w}{3^{r-1}} \times 3^{r-1} && (14)\\ &= O(\eta_s * s_w * d) && (15)\\ &= O(d) && (16) \end{aligned}$$

Note that for levels $r > log(m)$, 1 bit is allocated per round. Thus, for all these levels, a summary can be sent out in less than 3^r rounds. The maximum interval between receiving two successive state information for those nodes whose state is obtained using summaries greater than level r is less than that derived in the above equation. The maximum interval between when a j receives the state of i is thus $O(d)$. ■

Theorem III.6. *In $DSSR$, $j.\gamma_i$ is $O(d)$ where $d = dist(i,j)$.*

Proof: Consider a node p at level r where $r \leq log(m)$. To compute a summary at level r, level $r-1$ summaries are needed. The maximum distance between p and its neighboring level $r-1$ clusters is $2(3^{r-1})$ (From property $C3$). Thus, the latency to travel from level $r-1$ node to level r node is given by $2\eta_s s_w 3^{r-1}$ time ago, since latency between each pair of level 1 nodes (length of a *round*) is η_s slots. Note that in this algorithm, a level $r-1$ summary is actually transmitted in $3^(r-1)$ rounds by dividing it into 3^{r-1} parts. Thus, a level r summary is computed based on level $r-1$ summary that was generated $3\eta_s s_w 3^{r-1}$ time ago. A level $r-1$ summary is computed based a level $r-2$ summary and so on until level 0. Upon summation, we see that the staleness of a level 0 state information in a level r summary is $1.5\eta_s s_w 3^{r-1}$.

Note that a complete level $r-1$ snapshot is sent every 3^{r-1} rounds in a pipelined manner. Thus every 3^{r-1} rounds, a level $r-1$ snapshot is received by a node. On the other hand a level r snapshot is computed only every 3^r rounds. Thus a fresher level $r-1$ snapshot is always available to compute a new level r snapshot. Now, the maximum distance traveled by a level r summary is 3^{r+1} and the latency is bounded by $\eta_s s_w 3^{r+1}$. Therefore, the maximum total staleness in the state of any node i in a level r summary is bounded by $4\eta_s s_w 3^r$.

Recall that in order to update the local data structure of j, the state of a node i is updated using summary $M(x', y')$ where x' is the lowest level which contains information about k. Now the minimum distance between j and i for which a level r summary is the smallest level that contains information about j is 3^{r-1}.

The maximum staleness in the state of a node i at node j is then given by the following equation:

$$j.\gamma_i = O(\eta_s s_w d) \quad (17)$$

$$= O(d) \quad (18)$$

Note that at levels $r > log(m)$, 1 bit is allocated per round. Thus, for all these levels, a summary can be sent out in less than 3^r rounds. The maximum staleness for those nodes whose state is obtained using summaries greater than level $log(m)$ is less than that derived in the above equation.

■

F. Distance sensitivity in an irregular network

In a network with non-uniform density such as the one in Fig. 2(b), the tree structure between clusters may not yield a path between nodes that is proportional to the geometric distance between them. However, the clustering properties $C1 - C5$ ensure that the path between two nodes is proportional to the *hop* distance or the shortest path available between the two nodes in the network graph. This ensures that the distance sensitivity properties of the snapshot service are preserved in terms of the *hop* distance as opposed to the geometric distance.

IV. Adapting snapshot service to control application parameters

In this section, we illustrate the co-design aspect of our approach by describing how the overall system performance can be optimized by adapting network parameters to specific application requirements. We first provide an example of how the application parameters (specifically the relative speeds of the pursuer and evader) can be used to decide on allowable network latency, thus affecting the system communication cost. We then provide an example of how the application parameters (specifically the relative speeds of the pursuer and evader) can be used to decide on allowable network error, thereby affecting the system communication cost.

A. Adjusting slot width to match control requirements

Recall from Theorem II.1 that the allowable latency at any instant is given by Eq. 9. In *DSS*, the maximum staleness in the state of an object at any instant is given by Eq. 13. Using these, the allowable slot width for *DSS* can be determined as a function of the pursuer and evader speeds, as stated in the following Lemma.

Lemma IV.1. *In order for* DSS *to satisfy the requirements of eventual pursuit control stated in Theorem* $II.1$*, the duration of each transmission slot* s_w *must be smaller than* $\frac{1}{12\eta_s v_e}(1-\frac{\alpha+k+1}{k\alpha})$.

As evader speed decreases, we note that the requirement on the network service is relaxed and the allowable slot width increases which consequently reduces the cost of communication. Thus we are able to select network parameters that minimally satisfy the conditions for successful pursuit control, enabling us to optimize overall system performance.

B. Adjusting message size at each level

Recall that in *DSS*, information at each level is aggregated into $m = n_e(log_2(p)+1)$ bits. The number of regions (p) used to represent the location of an object within a cluster corresponds to the granularity of an object's location within a cluster. For instance, if $p = 1$, then only the presence or absence of each object within a cluster will be known by this encoding strategy, but the error in the location of the object provided by a level l clusterhead will be equal to the maximum radius of a level l clusterhead as the object can be anywhere in the cluster. The size of m thus depends on the granularity at which an object location is represented in each cluster. Specifically, depending on the relative speed of the pursuer and the evader, we can adjust the number of regions p that each cluster is divided into and meet the requirement on error imposed by Theorem II.1. This relation is summarized by the following Lemma for a grid network.

Lemma IV.2. *In order for the DSS algorithm to satisfy the requirements of eventual pursuit control stated in Theorem II.1, the number of regions* p *that each cluster is divided into in a grid network must be greater than* $\frac{9(\frac{\alpha+1}{\alpha-1})^2}{4}$.

Proof: Since each cluster is divided in to p regions and each clusterhead marks the location of the evader as one of the p regions, if a level l information is available to the pursuer about

evader e, then the maximum error in location is $\frac{3^l}{2\sqrt{p}}$. If the smallest level at which information about an evader e is available to pursuer p is l, then it must be the case that d_{pe} is at least 3^{l-1}. Thus for any given distance d_{pe}, the maximum error in location is bounded by $\frac{3d_{pe}}{2\sqrt{p}}$.

Recall from Theorem II.1 that if d_{pe} is the distance between pursuer and evader at any instant, then the maximum error in location allowed at the pursuer is $\frac{d_{pe}}{k}$ where $k > \frac{\alpha+1}{\alpha-1}$. Thus, we require that $p > \frac{9(\frac{\alpha+1}{\alpha-1})^2}{4}$. ■

As the ratio of pursuer to evader speed decreases, the number of regions p required within a cluster at any level increases and therefore m increases. Thus, depending upon the application parameter of relative speed, we can optimize the system communication cost.

V. Simulation results

In this section, we evaluate the performance of our distance sensitive snapshot services using simulations in JProwler. The goals of our simulation are: (1) to verify the distance sensitivity properties of information transfer in terms of latency and resolution and (2) to analyze the impact of node failures on the latency and resolution of snapshots.

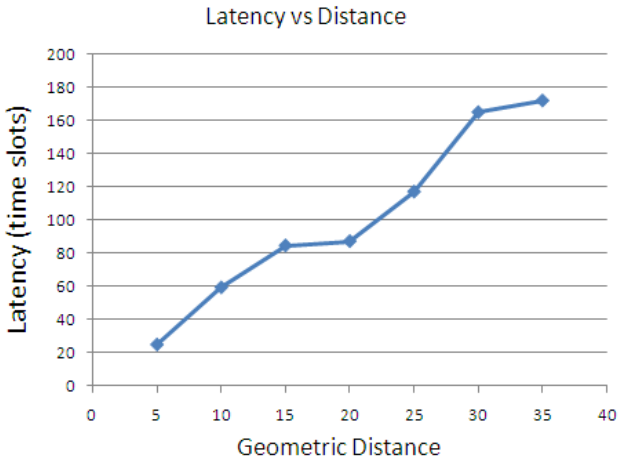
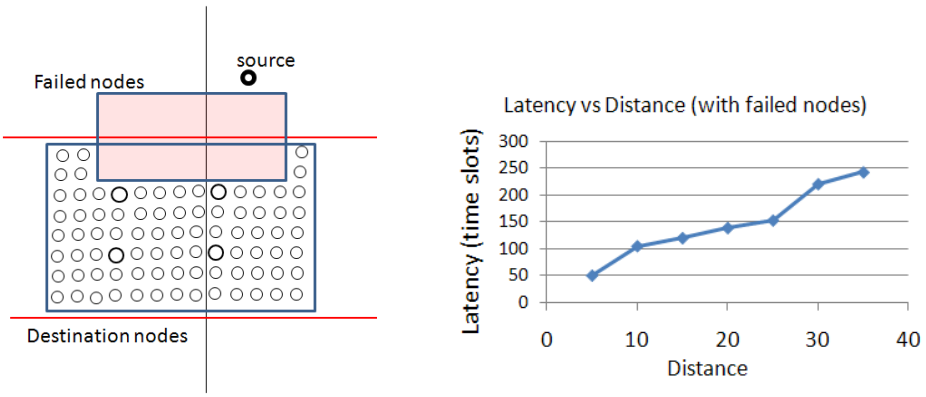


Fig. 8. Latency of information transfer measured in terms of number of transmission slots, plotted against geometric distance between nodes

Specifically, we consider an 81 by 81 wireless sensor network, arranged on a regularly spaced grid. A clustering service is assumed to provide each node with knowledge of its respective tree neighbors at each level in the hierarchy. A collision free schedule is assigned to nodes by which they take turns in transmitting. Each node is allotted a slot to transmit in every round and in other slots it listens to messages from its neighbors. In every round, a node generates 8 bits of information that is exchanged with other nodes in the network using the snapshot algorithm.

In order to measure the latency of information exchange we pick randomly selected source destination pairs across the network for a given distance. We measure the number of rounds for data generated at a given instant at the source to reach the destination node at the corresponding



(a) (b)

Fig. 9. (a)The figure shows a 15 by 9 section of nodes in the network that are failed. The latency of information transfer is measured between the node marked as source and nodes at different distances in the region marked as destination nodes. (b) The graph shows the latency of information transfer measured in terms of number of transmission slots in the presence of a 15 by 9 section of failed nodes shown in Fig. (a), plotted against the geometric distance between nodes

resolution (depending on the distance). We average these measurements and repeat the experiment for multiple distances. The results are shown in Fig. 8. The figure shows a linear increase in number of rounds as distance increases.

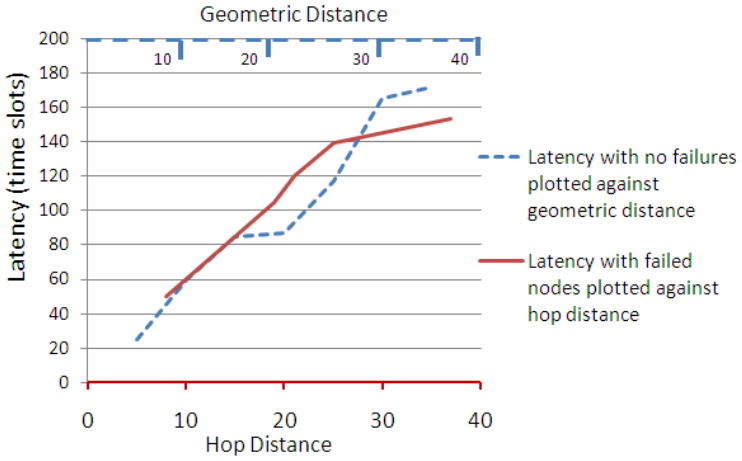


Fig. 10. Latency of information transfer measured in terms of number of transmission slots in the presence of a 15 by 9 section of failed nodes, plotted against the hop distance between nodes. This is compared with the latency in the presence of no failures at different geometric distances and the graphs closely match. This shows the preservation of distance sensitivity in terms of the hop distance in the presence of failed nodes.

In order to understand the impact of failures, we simulate a contiguous set of 135 failed nodes on a grid of 15 by 9 as shown in Fig. 9. We keep the source at one side of this *hole* in the network and consider destinations at different distances on the other side of the failed region, that are likely to be impacted by the failed nodes. The measured latency at different distances is shown

in Fig. 9. In Fig. 10, we plot these measurements against the *hop* distances between the nodes in the presence of failures. Alongside this graph, we plot the latencies measured without failures against the geometric distance obtained from Fig. 8 (show by blue dotted lines). These numbers closely match the latencies measured between nodes at corresponding geometric distance without failures. This illustrates that the snapshot algorithm preserves the distance sensitivity in terms of hop distance.

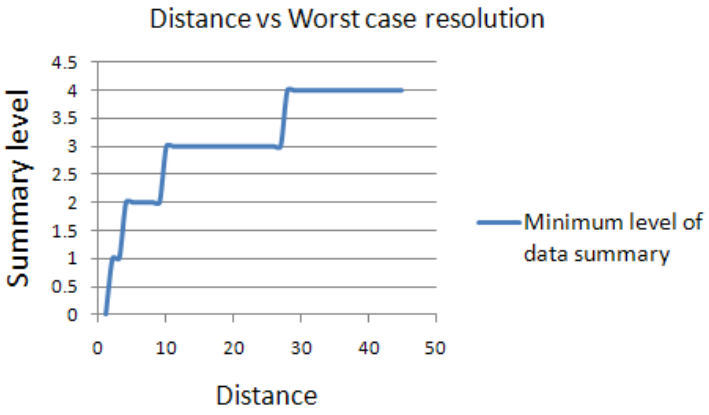


Fig. 11. The minimum resolution at which information is obtained at a node, plotted against the distance between nodes. The resolution is measured in terms of the level at which the aggregated summary is available.

In Fig. 11, we plot the minimum resolution at which information is obtained at a node, against the distance between nodes. We consider nodes at different distances across the network. For each pair of nodes, we determine the highest resolution at which information can be exchanged measured in terms of the aggregated summary level. For different pairs of nodes at the same distance, we consider the lowest resolution of information exchange at that distance and use that to plot the graph in Fig. 11.

In Fig. 12, we show the impact of failures on the resolution of received information. The two nodes shown in the figure, separated by a green arrow, are neighbors at level 1 without the failed region. In the presence of failures, the network is clustered in such a way that the nodes are now level 2 neighbors. In the presence of the failed nodes, the shortest hop distance between the nodes is increased to 9. Hence it is guaranteed by the properties of the clustering algorithm that the two nodes will be neighbors at level 2 or smaller. Since the two nodes are neighbors at level 2, they are able to exchange level 2 summaries and thus the error is now proportional to the hop distance between the nodes.

VI. Conclusions and Future Work

In this paper, we adopted a co-design approach to operate a distributed control application on top of a wireless sensor network. Specifically, we exploited distance sensitivity as a locality concept to enable the pursuit control application to be successfully run on top of a wireless

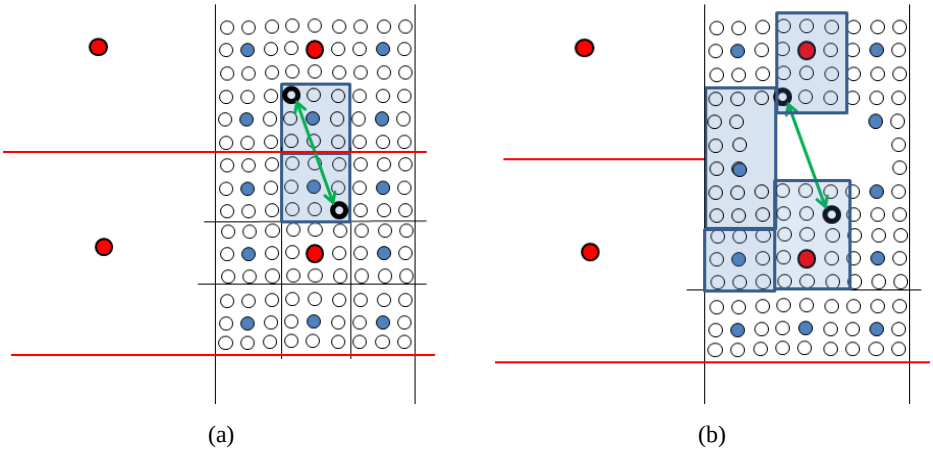


Fig. 12. (a) The two nodes represented by thick circles and separated by green arrows are neighbors at level 1. (b) In the presence of the failed nodes, they are neighbors at level 2 and exchange summaries at level 2.

sensor network. It is infeasible to ensure instantaneous, lossless delivery of information about evaders to mobile agents distributed across a network. Instead we showed that information that degrades linearly with distance in terms of latency, error and rate of delivery is sufficient to guarantee pursuit of evaders. We then designed algorithms that periodically deliver *distance sensitive* snapshots of the system to pursuing agents. We quantified the maximum rate at which information can be generated at each node so that snapshots are periodically delivered across the network and we showed how network parameters can be tuned to match application requirements.

The snapshot service designed in this paper can be additionally optimized for efficiency in the following ways: (1) Firstly compression in the temporal domain can be achieved by transmitting only the state of nodes that have changed from the previous *round*. (2) Secondly, the snapshot service can be specialized to transmit information only to a subset of nodes. With knowledge of future pursuer locations, the algorithm can be tuned at run-time to supply only aggregated information in specific directions and thereby realize savings in communication cost.

In this paper, we have ignored errors in the underlying object detection service with respect to object localization, track association, false alarms and missed detections. Consideration of these errors for reliable pursuit control is feasible and is an interesting avenue for further research.

In future, we also plan to explore the possibility of applying distance sensitive information in the context of control applications such as control of distributed parameter systems [27]–[29] as well as environmental and process control systems [30], [31]. We believe that distance sensitivity in information transfer is naturally applicable to many such distributed systems in which the impact of control or perturbation diminishes gradually with distance.

REFERENCES

[1] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, and V. Naik et. al. A Line in the Sand: A Wireless Sensor Network for Target Detection, Classification, and Tracking. *Computer Networks, Special Issue on Military Communications Systems and Technologies*, 46(5):605–634, July 2004.

[2] T. He, S. Krishnamurthy, and J. Stankovic. VigilNet: An Integrated Sensor Network System for Energy-Efficient Surveillance. *ACM Transactions on Sensor Networks*, 2(1):1–38, 2006.

[3] J. Liu, J. Reich, and F. Zhao. Collaborative in-network processing for target tracking. *Journal on Applied Signal Processing*, 2002.

[4] A. Arora and R. Ramnath. ExScal: Elements of an Extreme Wireless Sensor Network . In *The 11th International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 102–108, 2004.

[5] B. Sinopoli, C. Sharp, L. Schenato, S. Schaffert, and S. Sastry. Distributed Control Applications within Sensor Networks. In *Proceedings of the IEEE*, volume 91, pages 1235–46, Aug 2003.

[6] H. Cao, E. Ertin, and A. Arora. MiniMax Equilibrium of Networked Differential Games. *ACM Transations on Autonomic and Adaptive Systems*, 3(4):1–21, 2008.

[7] H. Cao, E. Ertin, V. Kulathumani, M. Sridharan, and A. Arora. Differential Games in Large Scale Sensor Actuator Networks. In *IPSN*, pages 77–84. ACM, 2006.

[8] M. Demirbas, A. Arora, and M. Gouda. A Pursuer-Evader Game for Sensor Networks. In *Sixth Symposium on Self-Stabilizing Systems(SSS)*, pages 1–16, 2003.

[9] J. Lewin and J. Breakwell. The surveillance-evasion game of degree. *Journal of Optimization Theory and Applications*, 16(3-4):339–353, 1975.

[10] J. Breakwell and A. Merz. Toward a complete solution of homicidal chauffeur problem. In *International Conference on Theory and Applications of Differential Games*, 1969.

[11] T. Muppirala, R. Murrieta-Cid, and S. Hjutchinson. Optimal Motion Strategies Based on Critical Events to Maintain Visibility of a Moving Target. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2005.

[12] P. Kachroo, S. A. Shedied, and H. Vanlandingham. Pursuit evasion: The herding noncooperative dynamic gamethe stochastic model. *IEEE Transactions On Systems, Man, and Cybernetics*, 32(1):37–42, 2002.

[13] K. Fregene, D. C. Kennedy, and D. W. L. Wang. Toward a systems and control-oriented agent framework. *IEEE Transactions On Systems, Man, and Cybernetics*, 35(5):999–1012, 2005.

[14] R. Issacs. *Differential games*. John Wiley and Sons, New York, 1965.

[15] L. Schenato, S. Oh, and S. Sastry. Swarm Coordination for Pursuit Evasion Games using Sensor Networks. In *International Conference on Robotics and Automation*, 2005.

[16] R. Vidal, O. Shakernia, H. J. Kim, D. H. Shim, and S. Sastry. Probabilistic pursuit-evasion games: Theory, implementation, and experimental evaluation. *IEEE Transaction on Robotics and Automation*, 18(5):662–669, Oct 2002.

[17] S. Oh, S. Russell, and S. Sastry. Markov Chain Monte Carlo Data Association for General Multiple-Target Tracking Problems. In *IEEE Conference on Decision and Control*, 2004.

[18] D. P. Spanos, R. Olfati-Saber, and R. M. Murray. Approximate Distributed Kalman Filtering in Sensor Networks with Quantifiable Performance. In *4th International Symposium on Information Processing in Sensor Networks (IPSN)*, page 18, Piscataway, NJ, USA, 2005. IEEE Press.

[19] A. Jadbabaie, J. Lin, and A. S. Morse. Coordination of Groups of Mobile Autonomous Agents using Nearest Neighbor Rules. *IEEE Transactions on Automatic Control*, 48:988–1001, 2003.

[20] M. Chandy and L. Lamport. Distributed Snapshots: Determining Global States of Distributed Systems. *ACM Transacttions on Computer Systems*, 3(5):63–75, 1985.

[21] J. Gao, L. J. Guibas, J. Hershberger, and L. Zhang. Fractionally Cascaded Information in a Sensor Network. In *IPSN*, pages 311–319, 2004.

[22] R. Sarkar, X. Zhu, and J. Gao. Hierarchical Spatial Gossip for MultiResolution Representations in Sensor Networks. In *IPSN*, pages 311–319, 2007.

[23] J. Shin, L. Guibas, and F. Zhao. A distributed algorithm for managing multi-target indentities in wireless ad hoc networks. In *International Workshop on Information processing in Sensor Networks IPSN*, pages 223–238, 2003.

[24] V. Kulathumani. *Network Abstractions for Designing Reliable Applications Using Wireless Sensor Networks*. PhD thesis, The Ohio State University, Columbus, OH, 2008.

[25] M. Demirbas, A. Arora, V. Mittal, and V. Kulathumani. A Fault-Local Self-Stabilizing Clustering Service for Wireless Ad Hoc Networks. *IEEE Transactions on Parallel and Distributed Systems*, 17(9):912–922, 2006.

[26] V. Mittal, M. Demirbas, and A. Arora. LOCI: Local Clustering Service for Large Scale Wireless Sensor Networks. Technical Report OSU-CISRC-2/03-TR07, The Ohio State University, 2003.

[27] K. Frampton. A Comparison of Hierarchies for Decentralized Vibration Control. *Journal of Vibration and Acoustics*, 2001.

[28] V. Kulathumani, Y. M. Kim, and A. Arora. Reliable Control System Design Despite Byzantine Actuators. In *Fifth International Conference on Multibody Systems, Nonlinear Dynamics and Controls (MSNDC)*, 2005.

[29] Q. Zhao, Y. Weng, and J. Jiang. Design of Reliable Control Systems with Multiple Sensors. In *Proceedings of the 11th Canadian Conference on Electrical and Computer Engineering*, volume 1, pages 225–228, Toronto, Canada, May 1998.

[30] C. J. Rozell and D. H. Johnson. Evaluating Local Contributions to Global Performance in Wireless Sensor and Actuator Networks. In *International conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 1–16, 2006.

[31] X. Cao, J. Chen, Y. Xiao, and Y. Sun. Distributed Collaborative Control Using Wireless Sensor and Actuator Networks. In *Second International Conference on Future Generation Communication and Networking*, pages 3–6, 2008.