FlashTrack: A Fast, In-network Tracking System for Sensor Networks

Hao Jiang, Jiannan Zhai, Jason O. Hallstrom
School of Computing
Clemson University
{hjiang, jzhai, jasonoh}@clemson.edu

Abstract—We revisit the classic object tracking problem with a novel and effective, yet straightforward distributed solution for resource-lean devices. The difficulty of object tracking lies in the mismatch between the limited computational capacity of typical sensor nodes and the processing requirements of typical tracking algorithms. In this paper, we introduce an in-network system for tracking mobile objects using resource-lean sensors. The system is based on a distributed, dynamically-scoped tracking algorithm which alters the event detection region and reporting rate based on object speed. A leader node records the detected samples across the event region and estimates the object’s location in situ. We study the performance of our tracking implementation on an 80-node testbed. The results show that it achieves high performance, even for very fast objects, and is readily implemented on resource-lean sensors. While the area is well-studied, the unique combination of algorithmic features represents a significant addition to the literature.

I. INTRODUCTION

Mobile object tracking is an interesting (and classic) application area for sensor networks. Typically, the target object is capable of being detected using low-level sensor signals. When a target moves across a network, nearby sensors are affected. The target’s trajectory is reflected in the changing data trend. For example, an acoustic sensor may use perceived volume to indicate the proximity of an object.

Tracking objects in a sensor network using proximity sensors is non-trivial. First, the sensing model is imperfect. Sensor readings are noisy and susceptible to environmental influence. As a result, most distance estimation models are imprecise. Second, since computation, storage, and bandwidth resources are limited on typical sensor nodes, it is difficult for these devices to store, process, and communicate the required data using cooperative signal processing techniques, especially if the result needs to be computed quickly. Third, as the target moves from region to region, a synchronized time reference becomes important for estimating speed and location, especially for fast moving objects. However, time synchronization increases energy and resource consumption. Finally, large-scale networks require highly scalable tracking solutions. Centralized solutions are typically a poor fit.

We present an efficient in-network tracking system for fast moving objects that represents a significant new contribution to a well-studied area. The framework constructs a series of dynamically-scoped event regions that follow the path of the target. A leader is elected in each region, and the size of the region is set based on the target’s estimated speed. Reports from nearby nodes are sent to the leader at a rate proportional to the target’s estimated speed. The target’s trajectory is computed by connecting these estimates using a weighting function. The framework is hierarchy-free, scalable, does not require time synchronization, and is readily implemented on resource-lean devices.

Contributions. (1) We present a novel distributed tracking algorithm which uses a dynamically-scoped event region tuned to the target object’s speed. (2) We present an adaptive snapshotting technique to estimate the target’s location according to the speed of the object, without the use of time synchronization. (3) We evaluate the approach on a large-scale testbed capable of providing ground truth for fast moving objects with high accuracy. The testbed itself is an interesting contribution to the area.

II. RELATED WORK

Object tracking is a classic problem in sensor networks. Guibas [6] originally proposed the problem in this context, but before that, Herlihy and Tirthapura [9] introduced a self-stabilizing algorithm for maintaining a distributed queue. This is a non-sequential flow problem that shares many similarities with distributed tracking problems. Since then, a number of authors have investigated tracking algorithms in sensor networks using a range of tracking models and sensor capabilities.

Aslam et al. [2] present a centralized object tracking algorithm based on particle filtering techniques. The authors assume binary sensors capable of indicating whether an object is moving into or out of sensor range. Two convex hulls of “moving out” sensors and “moving in” sensors are generated. The location of the target is estimated to be inbetween the two hulls, with a position weighted by the time duration the object was in the sensing range of each sensor.

Kim [11] presents a tracking algorithm using a single bit sensor which indicates whether an object is in sensing range. The author assumes constant object speed. The location of the object is estimated using a weighting function that relies on the time the object spent in the sensing range of each sensor. Shrivastava et al. [15] also rely on a binary proximity sensor. They present a linear “stabbing” algorithm to estimate the speed and trajectory of the target. However, the approach is not scalable. Wang et al. [17] improve scalability with a distributed algorithm. Possible variations in sensor detection errors are also considered in their approach.
Singh et al. [16] consider tracking targets of varying speed using single bit proximity sensors and design a multi-target tracking algorithm using particle filters. However, they only investigate the cases in a 1-D space.

Pfarrre and Kumar [14] use a combination of sensors to track an object passing through a directional laser. Speed and location are estimated by the time the object spends cutting the laser line. However, this work requires special deployment of the network and only handles straight trajectories.

Liu et al. [12] present a distributed tracking system to track military vehicles. In their approach, a leader node is selected to estimate the current location of the object and computes the next estimated location using a sequential Bayesian (Kalman) filter. If the next location is closer to another sensor, leadership is passed to that sensor. In their implementation, acoustic amplitude sensors are used to estimate the proximity of targets, and directional acoustic sensors are deployed to provide directional information. Brooks et al. [3] present a similar framework. The authors use extended Kalman filters to compute next-step nodes. The nodes which detect the object are set to active and alert nearby nodes using prediction results. To save communication bandwidth, the authors limit the scope of relevant nodes through spatial subdivision. However, the prediction techniques require resource-rich sensor nodes.

Arora et al. [1] present a military surveillance system which comprises detection, classification, and tracking features; magnetic sensors and radar units are used. The target location is decided by the centroid of the convex hull of sensors which detect the target. To limit noise, a heuristic bounding box is applied to tighten the size of the hull.

Demirbas et al. [5] present a self-stabilizing tracking algorithm that assumes hierarchical partitioning in space. Tracking information is propagated from lower levels to higher levels. Recent information can override misinformation at higher levels. However, the operation of hierarchy-based algorithms is energy-consumptive at cluster boundaries.

He et al. [8] present an object tracking framework that adopts a power management protocol to switch sensor nodes to sleep when they are idle. The authors also describe an efficient wake-up scheme and group aggregation technique to increase detection accuracy with limited delay. While end-to-end detection time is guaranteed by the system, the delay may be significant for fast-moving objects.

To our knowledge, the only work focused on light source tracking in sensor networks was presented by Gupta and Das [7]. Photo sensors are used as proximity sensors for detecting the light source target. They use triangulation to estimate the target position and fit the points with a straight line to estimate the trajectory. The experiments are conducted on a small scale network, and it is not clear if the algorithm is scalable.

While this survey is necessarily incomplete, it is representative. In contrast to prior work, we focus on a lightweight and scalable tracking framework for resource-lean sensor nodes, without using complicated cooperative signal processing techniques. A novel, dynamically-scoped event region and leader selection scheme is used to track moving objects of varying speed. The framework is decentralized, energy-efficient, highly accurate, and readily implemented on resource-lean hardware.

**Novelty.** Our approach is differentiated from prior work in three fundamental ways. First, we focus on a lightweight (but still highly accurate) tracking mechanism. Concretely, we avoid cooperative signal processing techniques, which are computationally-intensive compared to our approach, which is based on discrete neighborhood snapshots. Our tracking system requires approximately 30K of ROM and only 1K of RAM, making it suitable for most resource-lean sensing platforms. Second, our approach dynamically adapts to the speed of the target object, making it suitable for both slow and fast-moving objects, while conserving energy. Further, the snapshot rate is also adaptive based on target speed. Finally, our approach abandons network time synchronization in favor of a lighter-weight, pair-wise synchronization mechanism based on SFD interrupts (via the radio chip). In the following sections, we present this novel tracking approach, which achieves lean resource consumption, yet high accuracy, for both slow and fast-moving objects.

**III. FRAMEWORK DESIGN**

A. Dynamically-scoped Tracking Algorithm

We present a dynamically-scoped tracking algorithm that forms a sequence of event regions, each dominated by a leader node which detects the highest sensor reading within a given range. The leader node records the sensor readings from surrounding nodes that detect the target object. Leadership is passed along the tracking path as the target moves.

Before describing the algorithm, we make our assumptions explicit: Each node has a unique ID and a proximity sensor; this sensor is sampled periodically. We assume that the distance to the target is inversely proportional to the value of the sample. A node detects an object when the sampled value is higher than a specified threshold value. An in-situ classifier [10] could be used to classify the type of mobile object, if multiple sensors are available for detection.

Nodes are activated by a timer with a constant period and may only transmit on each timer event to avoid message congestion. The sampling rate is at least as fast as the transmission timer rate. Recent sample values are stored in a buffer.

To simplify the presentation, we first assume time synchronization. (We will later abandon this assumption.) As a result, sampling events are synchronized. We assume that nodes have location information about themselves. Based on the samples and location information of neighboring nodes (included in messages), it is possible to estimate the target’s position given that the samples are inversely proportional to the distance from the target. To enable location estimation, we assume that at any time $t$, at least two nodes are able to detect the object. The trajectory is estimated by connecting the sequence of discrete points from estimated locations.

When an object is in the network, several nodes may detect the object at the same time. We refer to the node with the
largest sample value as the dominating node\(^1\). Nodes within the dominating range of the dominating node are termed dominated nodes, and the set containing the dominating node and the dominated nodes is termed an event region. Note that it is not necessary to include all the 1-hop neighbors in the event region. A smaller event region involves fewer nodes, which decreases the possibility of message congestion. A smaller event region also avoids interference with other nodes which are not related to the locally detected event—for example, another object detected in another section of the network. The trade-off of using a small event region is that if the moving object is fast enough, in one round of domination, it may “escape” from the event region before the dominating node passes the domination token to the next dominating node. Hence, to track objects at different speeds, we establish the event region according to the speed of the moving object in the recent past. When an object is moving at a high speed (previously), a larger event region will be formed. All the nodes in the event region will be available for the next round of domination. If an object is moving at a slower speed, a smaller event region will be formed.

To formally define the algorithm, we present the action system program shown in Algorithm 1, which runs on each node. Each node has a globally unique ID, \(i\). The constant \(THR\) denotes the threshold of event detection; i.e., the value at which target detection is signaled. Each node maintains its location information, denoted by \(loc_i\), has access to global time, denoted by \(\tau\), and can access its sensor reading via \(x_i(t)\), sampled at time \(t\). The dominating node identifier and sample value (the maximum value in the event region) are stored in \(DOM_i\) and \(M_i\), respectively. Variable \(t_{sample}\) denotes the global time when the dominating value was sampled. Variable \(v_i\) denotes the estimated speed of the target object, and \(\rho_i\) denotes the radius of the current event region. \(SS_i\) denotes the snapshots collected within the event region, comprising a series of snapshots over time. Each snapshot records the state of all reporting nodes captured by the dominating node. The aggregate snapshot records the sensor readings from all nodes in the region, used to estimate the object’s position.

We introduce five functions. \(broadcast(i, x, t, loc, \rho)\) denotes a message broadcast from a dominating node, which contains the node ID, its most recent sample value, the sample time, the location of the node, and the event region radius. In contrast, \(report(DOM_i, i, x, t, loc)\) denotes a unicast message transmitted by a dominated node, which contains the dominating node ID, local node ID, sample value, sample time, and node location. \(addSnapshot(i, x, t, loc)\) adds snapshot information to \(SS_i\). \(computeSpeed(SS_i)\) computes the estimated speed of the object based on the current snapshot set, computed through a moving window. \(dis(loc, loc)\) returns the Euclidean distance between two points.

Initially, each node gets its own location \((X_i, Y_i)\) and sets \(DOM_i\) to bottom, meaning not dominated. The dominating value \(M_i\) is set to \(THR\); when a node detects a value higher

---

\(^1\)Identifiers are used to break ties to ensure a total ordering.

---

**Algorithm 1: Dynamically-Scoped Tracking Algorithm**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(loc_i)</td>
<td>({2D) location of node (i})</td>
</tr>
<tr>
<td>(\tau)</td>
<td>({synchronized) global time)</td>
</tr>
<tr>
<td>(x_i(t))</td>
<td>({sample of (i) at (t)})</td>
</tr>
<tr>
<td>(DOM_i)</td>
<td>({dominating node of (i)})</td>
</tr>
<tr>
<td>(M_i)</td>
<td>({max sample in the view of (i)})</td>
</tr>
<tr>
<td>(t_{sample})</td>
<td>({time when dominating node last sampled})</td>
</tr>
<tr>
<td>(v_i)</td>
<td>({object speed estimated by (i)})</td>
</tr>
<tr>
<td>(\rho_i)</td>
<td>({radius of event region})</td>
</tr>
<tr>
<td>(SS_i)</td>
<td>({snapshots within event region of (i)})</td>
</tr>
</tbody>
</table>

**CONSTANT**

\(THR : \mathbb{N}\) \(\{event threshold, \(THR > 0\)\}\)

**VAR**

\(\rho_{\text{MAX}}\) \(\{\text{radius of event region}\}\)

**INITIALLY**

\(loc_i := (X_i, Y_i); DOM_i := \bot; M_i := THR;\)

\(v_i := v_{\text{MAX}}; \rho_i := \rho_{\text{MAX}};\)

\(SS_i := \emptyset; t_{\text{sample}} := 0;\)

**ASSIGN**

if timer event fires at \(\tau\) then

\(\text{if } x_i(\tau) > M_i \lor (DOM_i = i \land x_i(\tau) > THR)\) then

\(v_i := \text{computeSpeed}(SS_i);\)

\(\rho_i \propto v_i;\)

end if

\(SS_i := \text{addSnapshot}(i, x_i(\tau), loc_i, \tau);\)

\(DOM_i := i; M_i := x_i(\tau);\)

\(broadcast(i, M_i, \tau, loc_i, \rho_i);\)

else if \(DOM_i = i \land x_i(\tau) \leq THR\) then

\(DOM_i := \bot; M_i := THR;\)

\(broadcast(\bot, M_i, \tau, loc_i, \rho_i);\)

else if \(DOM_i \neq i \land DOM_i \neq \bot \land x_i(\tau) > THR\) then

\(\text{report}(DOM_i, i, x_i(t_{\text{sample}}), t_{\text{sample}}, loc_i);\)

end if

end if

if \(\text{recv.broadcas}\(t (DOM_j, M_j, t_j, loc_j, \rho_j)\)\) event then

\(\text{if } (DOM_i = DOM_j \lor M_i < M_j \land j = \bot) \land\)

\(\text{dis}(loc_i, loc_j) < \rho_j\) then

\(DOM_i := j; M_i := M_j; t_{\text{sample}} := t_j; \rho_i := \rho_j;\)

end if

end if

if \(\text{recv.report}(j, i, x_j, t_j, loc_j)\) event then

\(SS_j := \text{addSnapshot}(j, x_j, t_j, loc_j);\)

end if

end if
than $THR$, it declares event detection. $v_i$ is set to the maximum speed, and $\rho_i$ is set to the maximum range (to aid in capturing fast objects, if they appear.) Snapshot set $SS_i$ is set to empty, and $t_{sample}$ is set to 0.

The first rule executes when the periodic timer is fired, at global time $\tau$. The sensor node compares its recent reading $x_i(\tau)$ with its dominating value $M_i$. If the sample $x_i(\tau)$ is greater than $M_i$, $i$ becomes a dominating node. If $i$ is already a dominating node and its sample $x_i(\tau)$ remains greater than $THR$, it continues dominating. When a node is dominating and its current snapshot set contains at least 2 samples greater than $THR$, it continues dominating. When a node is no longer dominating.

If node $i$ was previously dominating, but $x_i(\tau)$ is smaller than $THR$, the object is out of $i$’s sensing range, and as a result, node $i$ gives up domination, setting $DOM_i$ and $M_i$ to their initial values, and broadcasts a message indicating the node is no longer dominating.

If node $i$ was previously dominated, and $x_i(\tau)$ is at least equal to $M_i$, but greater than $THR$, it reports to its dominating node, $DOM_j$, with its sample value at time $t_{sample}$, the timestamp last received from the dominating node (rule 2). The next node adds a new snapshot record at time $\tau$, and sets $DOM_i$ and $M_i$ to $i$ and $x_i(\tau)$, respectively. Finally, a broadcast is sent declaring domination in the event region.

If node $i$ was previously dominating, but $x_i(\tau)$ is smaller than $THR$, the object is out of $i$’s sensing range, and as a result, node $i$ gives up domination, setting $DOM_i$ and $M_i$ to their initial values, and broadcasts a message indicating the node is no longer dominating.

The second rule executes when a node receives a message from a dominating node $j$. There are three possible scenarios if the node is within the event region range of $j$. First, if the dominating node was $j$ in the last round, $i$ still recognizes $j$ as its dominating node. Second, if $j$ is not equal to $DOM_i$, but the new dominating value from $j$ is greater than the current value of $M_i$, $j$ becomes $i$’s new dominating node. Third, if $j$ has already abandoned domination (DOM$_j$ = ⊥), $i$ becomes not dominated. In each case, the node sets $DOM_i$, $M_i$ and $\rho_i$ to $DOM_j$, $x_j$, and $\rho_j$, respectively, and records $j$’s timestamp.

The third rule executes when a node receives a report message from a neighbor in the event region. The receiving node may not be the dominating node now, but it must have been dominating before. When it receives the report, it adds the information to its set of snapshots.

At this point, we haven’t addressed how to generate a snapshot of an event region in an unsynchronized network, nor how target locations are estimated. To improve the algorithm, we describe an adaptive snapshotting technique using asynchronous timestamping and location estimation.

B. Adaptive Snapshots Using Timestamping

Snapshots record the sample states of neighboring nodes at times synchronized with the dominating node. As presented, the snapshotting algorithm assumes time synchronization for recording snapshots at each time $\tau$. However, network synchronization is computationally expensive and introduces additional communication overhead. However, a limited form of network synchronization can be helpful in recording snapshots within an event region. Many time synchronization algorithms use timestamping triggered by a start-of-frame delimiter (SFD) interrupt generated by the radio chip. The interrupt is signaled at the same time on both the transmitter and the receiver (within tens of microseconds) and can be used to establish millisecond-level (or better) synchronization.

Using the SFD interrupt, we can synchronize node samples according to the timestamps received from dominating nodes. Figure 1 illustrates the process. The two horizontal arrows denote the execution timelines of two nodes, A and B; A is the dominating node. The triangles denote (unsynchronized) timer events. Suppose that node $A$ samples its sensor at time $t_{Sen}(A)$ and transmits the data to node $B$. It often takes more than ten milliseconds for $B$ to receive the packet, depending on network traffic and link quality. There is no way for node $B$ to tell when $A$ sensed the data without timestamping. However, at $t_{SFD}(A)$, the SFD interrupt fires; at approximately the same time, the SFD interrupt also fires on $B$ at $t_{SFD}(B)$. Therefore, if we transmit the timestamp of $t_{Sen}(A)$ along with $t_{SFD}(A)$, $B$ can compute the sensing time in its view by $t_{Est}(B) = t_{SFD}(B) - (t_{SFD}(A) - t_{Sen}(A))$. Since the sampling period is small, we assume the sample values change linearly, and the sample $x(t_{Est})$ at $t_{Est}$ can be estimated by Equation 1.

$$x(t_{Est}) = x(t_{Pre}) + (x(t_{Pre}) - x(t_{Pre})) * (t_{Est} - t_{Pre}) / P$$ (1)

where $x(t_{Pre})$ and $x(t_{Pos})$ are samples just before and after $t_{Est}$, respectively, and $P$ is the sampling period. (A fixed size queue is used to buffer the samples.) Hence, it is possible to estimate the sample values across the event region at a time specified by the dominating node, without introducing additional communication overhead.

To keep track of the duration of each domination round, the domination duration is included within the dominating node’s broadcasts. The next dominating node receives both the duration and the sensing time. The time difference between dominating nodes is added to the received duration, since they are not synchronized, resulting in an accurate accumulated duration of object detection along the path.

Another challenge in snapshotting is the rate of snapshot generation. If we transmit the dominating message and record a snapshot on every timer event, as shown in Algorithm 1, the battery will be consumed quickly, especially for slow moving

\footnote{Transmission of $t_{SFD}$ is supported by hardware on most modern radio chips.}
objects, which may stay in the network longer. To reduce the number of transmissions without losing information, we adjust the snapshotting rate according to the target’s movement. Beyond using a constant snapshotting rate to provide periodic snapshots, we also record a snapshot in the following scenarios. First, when a new event region is established, a snapshot is taken. Second, when a previously inactive node detects the object in the event region, a snapshot is recorded. A “join” message is also sent to the dominating node by the newly detecting node. Third, if the rate of change in sample values on the dominating node is higher than a given threshold as compared to the past round, it indicates the object may move fast in the region; thus, a snapshot is required.

IV. SYSTEM IMPLEMENTATION

We implement the algorithm in the context of tracking point light sources. To track high-speed objects, we use a high sampling rate (i.e., 100Hz). We adopt a typical state machine design. Figure 2 presents a simplified automata that captures the high-level behavior of the system.

Fig. 2: System Model

A node state changes only if a periodic timer event is signaled, denoted by a solid edge, or a broadcast message is received, denoted by a dashed edge. Each node begins in the START state, where it computes the ambient light level, which may vary from one node to another. Later, when detecting a light source, each node compares its reading with its ambient value and transmits the difference. This supports tracking in environments with complex ambient conditions.

Next, the node transits to IDLE; it transits back to START periodically to update its ambient light calculation. We define the detection threshold THR as a constant plus the ambient light detected by the node. The node will remain in the IDLE state while its sensor reading is less than THR.

We introduce a temporary state, PREDOM, entered when a node detects an object and tries to dominate the event region by broadcasting its value. When a target appears in the network, several nodes may detect the object, but typically not at the same instant. As a result, the node which detects the object first may not be the one with the largest sample value. PREDOM is introduced to stabilize the event region; only one dominating node will be selected.

If a pre-dominating node doesn’t receive a message with a higher value before the next timer event, it transits to the DOM state and dominates the event region. At this point, the node begins to broadcast its domination and records the snapshots received within the event region according to the object’s speed. The node remains in the DOM state until one of two cases occurs. First, if the sensed value is lower than THR, it transits to the ALERT state. Second, if it receives a higher value from a pre-dominating node in the event region, it transits to the ACTIVE state.

The ACTIVE state is reached when a node is dominated and has a sensor reading higher than THR. ACTIVE nodes will report their values to the dominating node after receiving dominating broadcasts. When a PREDOM or DOM node receives a value higher than its value, it transits to the ACTIVE state. Analogously, when an ACTIVE node has a sensor reading higher than the dominating value, it transits to PREDOM, and may later become a DOM node. When an ACTIVE node obtains a value below THR, it changes to the ALERT state.

The ALERT state indicates that a node is in an event region, but contains a value less than THR. It may report to the dominating node if it was ACTIVE at the time it received the last dominating broadcast. If it has a value higher than THR, but less than the dominating value, it becomes ACTIVE; otherwise, if its value is higher than the dominating value, it transits to PREDOM. If it hasn’t received a dominating message after a given period, it becomes IDLE.

The transition sequence can be complicated. Consider the network containing 4 nodes shown in Figure 3. The nodes are labeled from A to D; we assume they are within a single event region. The horizontal bars denote the state of each node over time. Each triangle denotes an unsynchronized timer event.

Initially, all nodes are in the IDLE state. At time t_0, node A detects an object when its timer fires, moves to the PREDOM state, and broadcasts its message. It takes some time for transmission and reception to complete. Before B receives the message from A, it detects the object at time t_1, with a value higher than A’s value. B also transits to the PREDOM state and broadcasts. When C and D receive the PREDOM message from A, they set themselves to ALERT. When C detects the object with a value higher than A at time t_2, it also moves to the PREDOM state and broadcasts. However, it is preempted when it receives a higher value from B at t_3. Only one node will succeed in the bid for domination; the others will become ACTIVE. To improve stability, a node must have a value higher than the previous proposed value, plus a constant threshold (to avoid domination oscillation).

Assume B becomes the dominating node at time t_4 when its timer fires again and starts to broadcast its dominating value. All the ACTIVE nodes that receive the dominating message must report their values to the dominating node when their timers fire. Assume node D obtains a value higher than THR, but less than B at t_5; it becomes a new ACTIVE node and joins the event region via a join message. When dominating node B receives the message at t_6, it broadcasts again at t_7 and records a new snapshot when reports are collected. At time t_8, when D has a value higher than B, it moves to the PREDOM state and broadcasts. Since node D has received a dominating message from B, it must still report its value at time t_7; the PREDOM broadcast also contains a report value for B. Similarly, node A must report to node B at time t_9, even though it is in the ALERT state, since it was ACTIVE at time
when B requested a report. The captured snapshots at node B may be sent back to the basestation through a collection tree when B enters the IDLE state. D will become the new dominating node at time $t_{10}$.

The upper-bound on tracking speed is constrained by the timer rate and the packet transmission rate. If an object is moving fast enough, it may leave an event region before domination has been established (when the next timer fires); thus, the dominating node is unable to transit from PREDOM to DOM. In such a case, the object may “escape” the event region without being tracked.

### A. Tracking Structure

Figure 4 illustrates the snapshot data structure used at each dominating node, a three-layer tracking list. The first layer contains information about the target. Each identified target is assigned a unique track ID. The track ID comprises two parts, the ID of the first dominating node in the tracking process, and a local counter. The track ID is passed along as the object moves. It allows multiple objects moving in the network at the same time, as long as the active event regions never overlap (at the same time). Next, the tracking structure includes track step, a track counter. Since a node may be selected as a dominating node multiple times during a tracking process, the tracking step is used to disambiguate the domination sequence. Each tracking record maintains a pointer to a snapshot structure (level 2), and a pointer to the next tracking record.

A snapshot records the sensing values within an active event region at the time of request by the dominating node. Each snapshot record records the time when the snapshot was requested, beginning with the first dominating node. Finally, it includes a linked chain of node reports. Each report contains the reporting node ID, node location, and report value. The chain is used to estimate the target’s location; the timestamps are used to estimate speed and trajectory.

### B. Location Estimation

The attenuation of a typical point source is non-linear and difficult to analyze. Measured intensity varies with distance, angle, and reflection. To analyze these issues, we mounted a point light source on a tripod at a distance of 1.5 meters from a wall. Fourteen sensors were deployed side-by-side on the wall. The light source was pointed at the middle of the line. Figure 5 shows the measured light intensity versus distance (from center). The readings are modulo the associated ambient light levels. The attenuation is almost linear with distance – but the slope decreases at the center and edge areas. Based on these observations, we adopt a linear-weighted location estimation approach. We compute the weighted average on both the X and the Y axes according to Equation 2.

$$\mathbf{\bar{r}} = \frac{\sum_{i=1}^{n} x_i [X_i, Y_i]}{\sum_{i=1}^{n} x_i}$$ \hspace{1cm} (2)

where $n$ is the number of reporting sensors, $x_i$ is a reading from sensor $i$, and vector $[X_i, Y_i]$ is the 2D coordinate of $i$.

### V. EVALUATION AND ANALYSIS

We evaluated the tracking implementation on the Clemson University NESTbed system [4]. The testbed consists of 80 Telos nodes with on-board photo sensors (Hamamatsu S10871). The nodes are deployed on a wall-mounted, semi-regular grid of 10 columns and 8 rows. For evaluation purposes, we used a common flashlight as the light source. We didn’t require a fixed distance or angle, and did not control the ambient light in the room.
Consider the mobile light trajectory generated by FlashTrack in Figure 6. The target was moving from the bottom-right corner of the testbed to the top-left corner. The estimated trajectory was generated by connecting the raw estimated locations streamed from the network. In the plot, the “+” signs denote the position of the sensor nodes deployed in the grid. The blue dots denote the dominating nodes, and the green markers denote the active nodes along the target’s path. It is clear that the dominating nodes are selected close to the target’s path, and the active nodes are neighboring sensors which detect the target. On average, a dominating node dominates 3 to 4 active nodes when at the edge of the network, and 5 to 8 active nodes otherwise. The actual event region is larger; other nodes are in the alert state.

To record ground truth, an infrared recording system was developed: A Wiimote [13], which includes an infrared camera, was installed at a fixed point in front of the testbed as a recording device. The Wiimote provides high resolution, high frequency infrared point tracking. We installed a 65mW infrared laser on the flashlight, which provides a reflected tracking point for the Wiimote. The tracking point location is streamed by Bluetooth to a client application and recalibrated to match the testbed’s coordinates.

We evaluate the FlashTrack implementation with objects of varying speeds: (i) slow, (0, 1.5](m/s); (ii) moderate, [1.5, 4](m/s); (iii) fast, [4, 10](m/s); (iv) out of range, [10, ∞](m/s). The cell size is approximately 25 centimeters; as a result, slow objects travel less than 6 cells per second, moderate speed objects travel less than 16 cells per second, and the upper bound is 40 cells per second.

We first investigated a straight line trajectory at a moderate speed. Figure 7a shows the trajectory computed by FlashTrack and the ground truth provided by the Wiimote. The object traversed the network in approximately 1.3 seconds. The average snapshot rate was 13.14 snapshots per second. The figure shows that the trace estimated by FlashTrack matches the ground truth trajectory well. The maximum location estimation error is less than 4 cm. However, because of sensor noise and limited deployment density, the raw data generated a zigzag trace. This can be improved by curve fitting during post-processing, which is not the concern of this paper.

We also investigated the speed estimates provided by FlashTrack for the same trajectory, compared with ground truth. We computed the speed of the object by averaging the speed estimates through a filter of 4 samples. Figure 7b presents the estimated object speed over time. Like the trajectory estimates, the speed trends between the two sets match well. However, it is not precise at each step, particularly, at the beginning of the trajectory, where the estimated speed is slower than ground truth. The explanation is that the object was first detected at the edge of the network (bottom-right in Figure 7a), where there are no active nodes beneath the first dominating node; the estimation result is biased due to asymmetric node coverage. The average speed computed by FlashTrack differs from ground truth by less than 1% in this case, on average.
In summary, our tracking framework captures the trajectory of mobile objects in-network and achieves high fidelity. The algorithm dynamically adapts to target speed and is capable of capturing fast objects.

VI. CONCLUSION

In this paper, we described a lightweight, high-speed, in-network tracking framework for sensor networks. The framework is based on a distributed tracking algorithm which tracks mobile objects using a dynamically-scoped event region. An adaptive snapshotting algorithm is used to capture the state of the nodes in each region. We implemented the framework in the context of point light source tracking and evaluated the implementation on a large-scale testbed. The results show that the implementation efficiently and accurately tracks the target trajectory, even for very fast objects.

ACKNOWLEDGMENTS

This work is supported by the National Science Foundation (CNS-0745846, CNS-1126344).

REFERENCES