MetroTrack: Predictive Tracking of Mobile Events using Mobile Phones

Gahng-Seop Ahn[1], Mirco Musolesi[2], Hong Lu[3], Reza Olfati-Saber[3], and Andrew T. Campbell[3]

[1] The City University of New York, USA, gahn@ccny.cuny.edu
[2] University of St. Andrews, United Kingdom
[3] Dartmouth College, Hanover, NH, USA

Abstract. We propose to use mobile phones carried by people in their everyday lives as mobile sensors to track mobile events. We argue that sensor-enabled mobile phones are best suited to deliver sensing services (e.g., tracking in urban areas) than more traditional solutions, such as static sensor networks, which are limited in scale, performance, and cost. There are a number of challenges in developing a mobile event tracking system using mobile phones. First, mobile sensors need to be tasked before sensing can begin, and only those mobile sensors near the target event should be tasked for the system to scale effectively. Second, there is no guarantee of a sufficient density of mobile sensors around any given event of interest because the mobility of people is uncontrolled. This results in time-varying sensor coverage and disruptive tracking of events, i.e., targets will be lost and must be efficiently recovered. To address these challenges, we propose *MetroTrack*, a mobile-event tracking system based on off-the-shelf mobile phones. MetroTrack is capable of tracking mobile targets through collaboration among local sensing devices that track and predict the future location of a target using a distributed Kalman-Consensus filtering algorithm. We present a proof-of-concept implementation of MetroTrack using Nokia N80 and N95 phones. Large scale simulation results indicate that MetroTrack prolongs the tracking duration in the presence of varying mobile sensor density.

1 Introduction

Urban sensing and tracking [1, 5] is an emerging area of interest that presents a new set of challenges for traditional applications such as tracking noise, pollutants, objects (e.g., based on radio signatures using RFID tags), people, cars, or as recently discussed in the literature and popular press, weapons of mass destruction [16]. Traditional tracking solutions [4, 7] are based on the deployment of static sensor networks. Building sensor networks for urban environments requires careful planning and deployment of possibly a very large number of sensors capable of offering sufficient coverage density for event detection and tracking. Unless the network provides complete coverage, it must be determined in advance where the network should be deployed. However, it is challenging to determine where the network should be deployed because events are unpredictable in time and space. We believe the use of static networks across urban areas has significant cost, scaling, coverage, and performance issues that will limit their deployment.

An alternative design of such a sensor system, which we propose in this paper, is to use people's mobile phones as mobile sensors to track mobile events. Increasingly, mobile phones are becoming more computation capable and embed sensors and communication support. Therefore, making a sensor network based on mobile phones is becoming more of a reality. For example, many high-end mobile phones, such as Nokia N95 phones, include a number of different radio technologies (e.g., multiple cellular radios, WiFi, and Bluetooth), and sensors (e.g., accelerometer, microphone, camera, and GPS) that are programmable. We imagine that micro-electro-mechanical systems (MEMS) technology will allow for the integration of more specialized sensors (e.g., pollution/air quality sensor, bio sensor, and chemical sensor) in the future. In our design, we assume that we can exploit the mobile phones belonging to people going about their daily lives or defined groups (e.g., federal employees, transit workers, police). Ultimately, the more people who opt in to being a part of the sensor network, the better the density and sensing coverage will be and the more effective urban sensing system will become in delivering services.

There are several important challenges in building a mobile event tracking system using mobile sensors. First, mobile sensors must be tasked before sensing [4]. Another issue that complicates the design of the system is that the mobility of mobile phones (therefore, the mobile sensors) is uncontrolled. This work diverges from mobile sensing systems that use the controlled mobility of a device (e.g., a robot) as part of the overall sensor system design. In such cases, the system can be optimized to drive the mobility of the sensors in response to detected events [11]. Due to the uncontrolled mobility of the mobile sensors, there is no guarantee that there will always be high enough density of mobile sensors around any given event of interest. The density changes over time so that sometimes there is a sufficient number of devices around the event to be tracked, and at other times, there is limited device density. One can think of this as dynamic sensor network coverage. The event tracking process has to be designed assuming that the process of tracking will be disrupted periodically in response to dynamic density and coverage conditions. Thus, a fundamental problem is how to recover a target when the system loses track of the target due to changing coverage.

In this paper, we propose *MetroTrack*, a system capable of tracking mobile events using off-the-shelf mobile phones. MetroTrack is predicated on the fact that a target will be lost during the tracking process, and thus it takes compensatory action to recover the target, allowing the tracking process to continue. In this sense, MetroTrack is designed to be responsive to the changing density of mobile phones and the changing sensor network coverage. The MetroTrack system is capable of tasking mobile sensors around a target event of interest and recovering lost targets by tasking other mobile sensors in close proximity of the lost target based on a prediction of its future location.

MetroTrack is based on two algorithms, namely *information-driven tasking* and *prediction-based recovery*. The tasking is information-driven because each sensor node independently determines whether to forward the tracking task to its neighbors or not, according to its local sensor state information. If the sensor readings meet the criteria of the event being tracked, then the sensor node forwards the task to its neighbors, informing them it detected the event.

The recovery is based on a prediction algorithm that estimates the lost target and its margin of error. MetroTrack uses a geocast approach similar to the algorithms in [12, 8] to forward the task to the sensors in the projected area of the target. In our prior work, Olfati-Saber [15] presented the Distributed Kalman-Consensus filter (DKF) that defined the theoretical foundation of distributed tracking of mobile events. In this paper, we extend this work and importantly implement it in an experimental mobile sensing network. We adapt the DKF for the prediction of the projected area of the target.

MetroTrack does not have to rely on a central entity (i.e., a tracking leader) because MetroTrack tracks events based on local state and interactions between mobile phones in the vicinity of a target. Therefore, MetroTrack is simple, flexible, robust, and easy to deploy. However, we do not rule out the potential help from infrastructure. Also, mobile phones occasionally interact with the back-end servers using cellular or infrastructure-based Wi-Fi connectivity for initial tasking purposes or to inform the back-end of the targets progress. In this paper, we focus on the interaction between mobiles and reserve the issues of the interaction with the back-end servers as future work.

Also, we do not discuss what would provide the incentive for more people to opt in (even if we believe mechanisms devised for peer-to-peer systems can be exploited [13, 17]), nor do we discuss the important privacy, trust, and security issues that predicate the wide-scale adoption of these ideas. Rather, we leave those issues for future work and focus on the proof of concept and evaluation of a system that is capable of tracking mobile events using mobile phones. To the best of our knowledge, this is the first sensor-based tracking system of mobile events using mobile phones.

The paper is organized as follows. Section 2 describes the information-driven tasking and the prediction-based recovery of MetroTrack. In Section 3, we present the mathematical formulation of the prediction algorithm that is the basis for the prediction-based recovery. In Section 4, we discuss the implementation and the performance evaluation of MetroTrack. A proof-of-concept prototype of MetroTrack is implemented using Nokia N80 and N95 phones to show that MetroTrack can effectively track a mobile noise source in an outdoor urban environment. Following this, in Section 5, we address the large-scale design space of MetroTrack which cannot be analyzed from a small-scale testbed deployment. Section 6 presents some concluding remarks.

2 MetroTrack Design

2.1 Information-driven Tasking

The *tracking initiation* can be done in two ways, i.e., user initiation or sentry sensor [4] initiation. A user can request to track an event described by certain attributes when the target event is encountered. Another way is to rely on sentry nodes to detect the event to be tracked. The sentry nodes can be selected from mobile nodes that have enough power to periodically turn on their sensors and start sampling. When one of the sentry nodes detects an event that matches the pre-defined event description, the node initiates the tracking procedure. The device associated with the requesting user or first sentry node that has detected the event becomes an initiator.

The *tasking* is a distributed process. Each neighboring sensor node that receives the task message performs sensing. The sensor node does not forward the task message to its neighbors unless it detects the event. The task message is forwarded by the sensors that are tasked and have detected the event. Hence, the nodes in close proximity to the event are tasked and the size of the tasked region is one hop wider than the event sensing range. As a result, the sensors just outside the event sensing range are already tasked and ready to detect the event wherever it moves. Each sensor node locally determines whether it has detected the event by comparing the sensor reading and the description of the event in the task message. As discussed earlier, the description of the event includes the modality of the sensors that can detect the event and the methodology by which the event can be detected (such as a threshold value). If the modality of the sensor node matches one of the modalities specified in the task message (i.e., the device is able to sense the event), then the sensor node starts the sampling process.

The responsibilities of the sensor that detects the event are as follows. The sensor should keep sensing the event using a high sampling rate and report the data to the back-end servers. In addition, the sensor should periodically forward the task message to its neighboring sensor nodes. The sensors that are tasked with one of the task messages containing the same event identifier form a tracking group. We note that this algorithm is not based on the election of a leader. Maintaining a leader for a group requires overhead. In addition, the failure of the leader affects the overall operation of the tracking system. MetroTrack can maintain the group and task the sensors to track the target without the need of a leader.

2.2 Prediction-based Recovery

This section describes the prediction-based recovery. First, the *recovery initiation* is as follows. The task of tracking the event is distributed among multiple mobile sensors. If a sensor is not detecting the event, this is not considered sufficient to infer that the target is completely lost since other sensors may still be sensing the event. In MetroTrack, a mobile sensor listens to other mobile sensors to minimize the false positives of such decisions. A sensor that has detected the event previously but currently is not detecting the event listens to the task messages forwarded from its neighboring nodes. If none of the neighboring nodes is forwarding the task message, the device infers that the target is lost. Assuming that the speed of the target is comparable to that of a tracking node and the sampling rate of sensors is high enough to detect the event, the overhearing will prevent false positives. However, there might still be false positives if the density of sensors is not sufficient. If a sensor makes a wrong decision, each node will forward an unnecessary number of task requests. However, the penalty is bounded by limiting the duration of the recovery process. In addition, MetroTrack performs suppression to explicitly stop the sensors from forwarding unnecessary messages. When one of the sensors declares that the target is lost, as described above, then the sensor initiates the recovery process by broadcasting a recovery message.

The *recovery process* is based on the estimation of the location of the lost target and the error margin associated to the prediction. The recovery message contains the information about the lost target. MetroTrack adopts a geocast scheme similar to the algorithms in [12, 8] to forward the recovery message to the sensors in the projected area in which the target will likely move. The sensors that receive the recovery message attempt to detect the target. If one of the sensors receiving the recovery message detects the target, then the recovery process is complete. The sensor that recovered the target broadcasts a task message, which resumes the information-driven tasking part of the protocol. All the hosts in the recovery area are in the recovery state. We considered a projected circular area. The center of the projected area is the predicted target location and the radius is the error margin of the prediction. The calculation of this area is based on the Kalman filter forecasting techniques, as described in Section 3. MetroTrack calculates the radius R of the recovery area as:

$$R = R_p + R_s + R_c \tag{1}$$

where R_p corresponds to the error margin associated to the prediction (see Equation 8 in Section 3). Our goal is to task all the sensors that are likely to be in contact with the target inside the projected region so we add the sensing range (R_s) to this radius. Finally, we also add the communication range of the devices (R_c) in order to be able to have the nodes that are at a one-hop distance from those at the border of the area with radius R_p+R_s in recovery state. These nodes are likely to enter the area and are particularly useful in spreading the recovery messages in the case of sparse network topologies. We note that the disk-shaped model is an approximated conceptual model that, in a real deployment, is influenced by the GPS errors for localization and by non uniform radio propagation and interferences. A node that has received the recovery message stays in recovery state until the node moves outside the recovery area or the recovery process timer expires. A timeout is specified to limit the duration of the recovery process. If the target is not recovered after the expiration of the timer, MetroTrack stops tracking the target. The nodes in recovery state periodically broadcast the recovery message to their instant (one-hop) neighbors so that new nodes that move into the recovery area can receive the recovery message.

It may happen that some sensors can be still in the recovery state while other sensors have already recovered the target and started to track it. It may also happen that the target event disappears (e.g., a sound source that is suddenly silent). MetroTrack addresses this problem using two mechanisms. First, it limits the duration of the recovery process and the spatial dissemination of the recovery messages. Second, MetroTrack performs a *suppression process* to reduce unnecessary overhead. Every node that recovers the target or receives a task message broadcasts a suppression message that is disseminated among the devices in the recovery area. Every node that receives the suppression message inside the recovery area re-broadcasts the message, or, if the node is in recovery state, it stops the recovery process and stops broadcasting the recovery message.

3 Prediction Algorithm

3.1 Prediction Model

In this section, we provide an overview of the prediction model in order to fully understand the collaborative prediction protocol used for the recovery process. We define a generic model for predicting the movement of a target in geographical space based on the Constant Velocity model [3], which is widely used in mobile tracking. Despite of the term 'constant velocity', the Constant Velocity model represents a moving target with dynamically changing velocity with certain variance. We consider a moving target with position $q \in \Re^2$ and a velocity $p \in \Re^2$. The one-step predictor is defined as follows:

$$\hat{x}(k+1) = A\bar{x}(k) + Bw(k) \tag{2}$$

where $x(k) = [q_1(k), p_1(k), q_2(k), p_2(k)]$ denotes the state of the target at time k. $\bar{x}(k)$ indicates the *prior state estimate* at step k given the knowledge of the movement under observation, whereas $\hat{x}$ indicates the *state estimate* of the same process at time $k+1$. q_1 and p_1 are the position and the speed on the x-axis and q_2 and p_2 are the position and speed on the y-axis, respectively. $w(k)$ is a zero-mean Gaussian noise denoted by $N(0,1)$. The prior estimate is the information stored in the phones and periodically exchanged among the phones that are in reach. The matrix A and B are defined as follows:

$$A = \begin{pmatrix} 1\,\epsilon\,0\,0 \\ 0\,1\,0\,0 \\ 0\,0\,1\,\epsilon \\ 0\,0\,0\,1 \end{pmatrix}, \quad B = I_2 \otimes G, \quad \textit{with} \quad G = \begin{pmatrix} \epsilon^2\sigma_0/2 \\ \epsilon\sigma_0 \end{pmatrix}$$

where ϵ is the interval of steps and $\otimes$ denotes the Kronecker product of matrices. The prediction for the instant $k+2$ is defined as follows:

$$\hat{x}(k+2) = A^2\bar{x}(k) + ABw(k) + Bw(k+1) \tag{3}$$

The generic prediction for the instant $k+m$ is defined as:

$$\hat{x}(k+m) = A^m \bar{x}(k) + \sum_{j=0}^{m-1} A^j B w(k+m-1-j) \tag{4}$$

The meaning of the symbols $\hat{x}$ and $\bar{x}$ is the same of the $k+1$ case. This equation can be rewritten as:

$$\hat{x}(k+m) = A^m \bar{x}(k) + v(k) \tag{5}$$

where $v(k)$ is the noise associated to the $k+m$ prediction defined as:

$$v(k) = \sum_{j=0}^{m-1} A^j B w(k+m-1-j) \tag{6}$$

The variance of $v(k)$ is

$$R_v = \begin{pmatrix} \sigma^2_{v_{q_1}} & 0 & 0 & 0 \\ 0 & \sigma^2_{v_{p_1}} & 0 & 0 \\ 0 & 0 & \sigma^2_{v_{q_2}} & 0 \\ 0 & 0 & 0 & \sigma^2_{v_{p_2}} \end{pmatrix} = [\sum_{j=0}^{m-1} A^j B B^T (A^j)^T] R_w \tag{7}$$

where $R_w = I_4$. Therefore, the center of the recovery region is $(\hat{q}_{(k+m)_1}, \hat{q}_{(k+m)_2})$. We consider a radius for the recovery area equal to:

$$r = max[2\sigma_{v_{q_1}}, 2\sigma_{v_{q_2}}] \tag{8}$$

The value of r is chosen in order to obtain a 95% confidence interval for the projected recovery area. In other words, we can assume that the target will be located in the recovery area with approximately 95% probability.

3.2 Distributed Kalman-Consensus Filter

In our prior work, Olfati-Saber [15] presented the Distributed Kalman Consensus Filter that defined the theoretical foundation of distributed tracking of mobile events. Algorithm 1 is the outcome of [15]. We feed our prediction model presented in the previous section to Algorithm 1 to predict the location of the target after it is lost and to calculate the projected area for the recovery process.

Each node i runs the distributed estimation algorithm shown in Algorithm 1. We indicate with z_i the observation performed by each node. N_i indicates the neighbors of node i. The message that is periodically broadcasted contains the following tuple: msg_i =[u_i, U_i, $\hat{x}_i$]. The local aggregation and calculation is described in step 3, whereas the estimation of the consensus among the neighbors is performed in step 4. The equations of the update of the filter are presented in step 5. The sensing model that we use is the following:

$$z_i(k) = H_i(k)x(k) + v_i(k) \tag{9}$$

Algorithm 1 Distributed Kalman Consensus Filter

```
1: Initialization: $P_i = P_0, \bar{x}_i = x(0)$
2: while new data exists do
3:     Locally aggregate data and covariance matrices:
           $J_i = N_i \cup \{i\}$
           $u_j = H_j^T R_j^{-1} z_j,\ \forall j \in J_i,\ y_i = \sum_{j \in J_i} u_j$
           $U_j = H_j^T R_j^{-1} H_j,\ \forall j \in J_i,\ S_i = \sum_{j \in J_i} U_j$
4:     Compute the Kalman-Consensus estimate:
           $M_i = (P_i^{-1} + S_i)^{-1}$
           $\hat{x}_i = \bar{x}_i + M_i(y_i - S_i \bar{x}_i) + \epsilon M_i \sum_{j \in N_i} (\bar{x}_j - \bar{x}_i)$
5:     Update the state of the Kalman-Consensus filter:
           $P_i \leftarrow A M_i A^T + B Q B^T$
           $\bar{x}_i \leftarrow A \hat{x}_i$
6: end while
```

where $H_i(k)$ is the observation matrix and $v_i(k)$ is the zero-mean Gaussian noise of the measurements of the ith node with covariance R_i. In our implementation, we assume that the value of the observation matrices $H_i(k)$ is the same for the all nodes over time and it is equal to:

$$H = \begin{pmatrix} 1\,0\,0\,0 \\ 0\,0\,1\,0 \end{pmatrix}$$

We also assume that the value of R_i is equal to a constant for all the matrices:

$$R_i = \sigma_R^2 I_2 \tag{10}$$

The value of Q is the same for all the devices since it is only dependent on the value of the process under observation that is the same for all the devices (i.e., the position of the moving target):

$$Q = \sigma_0^2 I_4 \tag{11}$$

Finally, P_0 is defined as

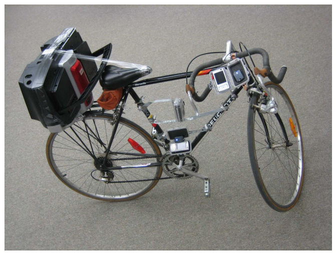
$$P_0 = \sigma_R^2 I_4 \tag{12}$$

4 Implementation and Experiment

4.1 Implementation

We built a proof-of-concept, mobile phone-based testbed to evaluate the MetroTrack system. The testbed consists of Nokia N80 and N95 smart phones (shown

(a)



(b)

Fig. 1: (a)From left to right: N95, GPS dongle, N80. (b)The boombox bike.

in Figure 1(a)) running Symbian OS S60. Both of them are equipped with a microphone and a camera that are accessible via software. With respect to network connectivity, they are both equipped with Bluetooth and WiFi interfaces. The N95 phones also feature an integrated GPS and an accelerometer. Since the N80 phones are not equipped with a GPS, we used an external dongle (shown in Figure 1(a)) based on the SiRFstar III chipset connected to the phone via Bluetooth. The devices use GPS information for sound source localization and the recovery process. In our testbed, we used WiFi for local ad hoc communications between mobile phones and used UDP broadcasting. The MetroTrack system is written in PyS60 [14], Nokia's porting of Python 2.2 for Symbian OS S60. Currently, PyS60 is more flexible than the Nokia implementation of J2ME for the N80 and N95 phones with respect to the programming interface for accessing the sensors embedded on the phones. With respect to the Symbian C++ development environment, it provides high-level abstractions that are extremely useful and convenient for the rapid prototyping of applications.

We implement an experimental sound source tracking application interfaced with MetroTrack. The system architecture is illustrated in Figure 2. We record sound samples using the microphone every 2 s. To estimate the distance from the target, we compute the Root Mean Square (RMS) of the average sound signal amplitude. If the calculated RMS value is distinctively greater than the ambient noise level, the sensor determines that the target event is detected and feeds the RMS value to the distance estimation component. An alternative method is *bearing estimation* [6], but it is not applicable to mobile phones due to the requirement of two microphones on one device with known orientation.

We implement two prediction mechanisms, a local Kalman filter (LKF) [10] and a consensus-based distributed Kalman filter (DKF) [15] in order to evaluate the trade-offs between the two. The LKF is simply a special case of the DKF without sharing information among neighboring devices. We implement the DKF, as described in Section 3. With respect to the mathematical model presented in Section 3, for the LKF, we assume that $J_i = \emptyset \cup \{i\}$.

The distance between the sensor and the sound source can be estimated from the RMS value assuming that we know the original volume of the target sound and the pattern of the sound attenuation over distance. The prototype

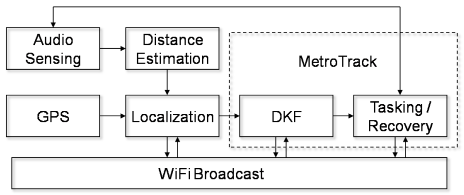


Fig. 2: System Architecture.

is based on a trilateration, which is a widely used localization scheme in GPS. After estimating its location and distance from the target, each sensor shares this information with its one-hop neighbors for trilateration, which require distances from two reference points for 2-D localization. The target location estimated by the sound source localization is fed into the Distributed Kalman filter component as the observation of the node.

4.2 Experiment

We mount a boombox, which plays constant pink noise (i.e., a signal with a frequency spectrum such that the power spectral density is proportional to the reciprocal of the frequency), on the back of a bike (aka boombox bike). We move it at a slow pace along paths around a university campus at approximately walking speed. We set the speaker of the boombox to face down toward the ground (as shown in Figure 1(b)) so that the sound would be reflected and spread omni-directionally in 2-D dimensions.

We set up a tracking testbed composed of two N95 phones and nine N80 phones connected to nine Bluetooth GPS dongles. The sound is sampled by the microphone on each phone for 0.5 s. The sampling is performed every 2 s. The time interval between each sampling is 1.5 s. Because the mobile phones are not always performing the tracking process (i.e., it can be defined as opportunistic), we argue that the maximum achievable sampling rate and minimum transmission interval of the messages should be used. Energy cost is not an issue if the device is not frequently involved in the tracking process. The values of the intervals are those sufficient for both the N80 and N95 phones for the RMS calculation, the distance estimation, the sound source localization, and the Distributed Kalman filter update calculation. We note that in existing tracking systems, the time intervals are much smaller than those used in MetroTrack (i.e., approximately 0.1-0.2 s) [7]. We set the WiFi transmission power to 100 mW. The communication range is between 25 and 30 m.

We perform the sound source tracking experiment evaluating the accuracy of the sound source localization as well as the effectiveness of the MetroTrack tasking and recovery. The GPS trace of the target is shown in Figure 3(a). Each person carries a phone and a Bluetooth dongle. Given the limitation of the number of phones and people, we emulate the density of an urban setting by allowing people to move around within 40 m from the target (i.e., the boombox

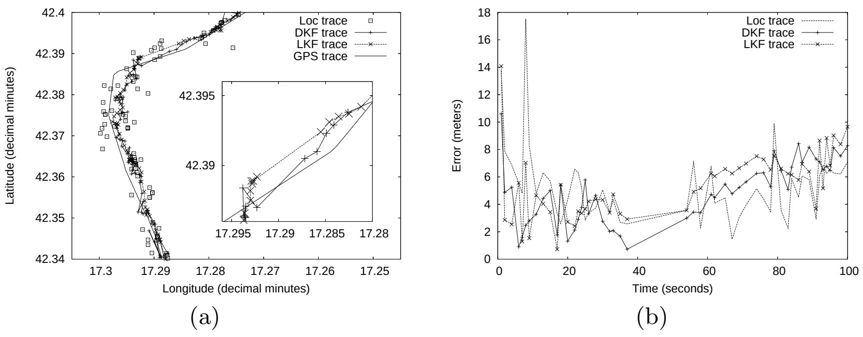


Fig. 3: (a) Trace of target's location. (b) Time trace of the localization error.

bike). Given the restriction of being within 40 m from the target, each person was allowed to move randomly in and out of the sensing range (approximately 20 m). This mobility setup is sufficient for testing the effectiveness of the tasking process. We emulate the case of losing the target by turning the sound off for 16 s and then turning the sound on again to observe whether the recovery process is working effectively.

The trace of the target measured using the sound source localization scheme is shown in Figure 3(a). (See the curve *Loc trace* in the plot.) As observed in Figure 3(a), the measured location is noisy. The sound source localization error is not only caused by the error of the RMS measurement but also by the error of the GPS positioning estimation of the mobile sensors. Each mobile sensor uses its own GPS receiver, and the accuracy of these receivers varies, even if they are of the same model. Also, some mobile sensors do not have valid GPS readings at all on a cloudy day. We have learned that calibrating the GPS reading among different sensors and checking the integrity of the GPS position of the mobile sensors is a real challenge that needs to be addressed in the future. The inset in Figure 3(a) shows a zoomed section of the gap in the traces related to the recovery phase. For clarity, the localization traces are not shown in the inset.

We also test the LKF and DKF estimations by setting σ_R to 7 m because we learned by trying the experiment several times that the standard deviation of the sound source localization error (σ_R) is approximately 7 m. The trace of the LKF and DKF estimations of the target location is also shown in Figure 3(a). In order to show the correctness of the prediction mechanism, we plot the time trace of the error of the location estimation in Figure 3(b). The target in this figure starts at instant $t = 0$ from the top of the area to the bottom. In Figure 3(b) we show the time interval of the first 100 seconds, including the interval during which the target was lost (i.e., between time $t = 37$ s and $t = 54$ s). We observe that the estimation error of DKF is smaller than the error of the LKF.

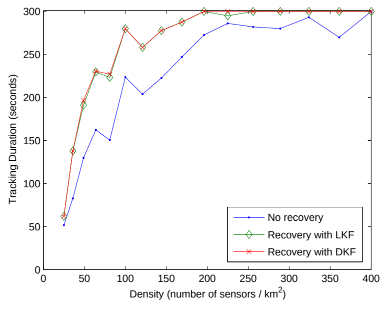
5 Simulation Study

We evaluate the performance of MetroTrack for a number of different deployment scenarios using a time-driven simulator based on MATLAB. The simulation results complement the experimental evaluation by studying issues not easily evaluated in a small-scale testbed, such as scaling and a sensitivity analysis of the system. In this section, we show the tracking duration performance for various sensor densities and sensing ranges.

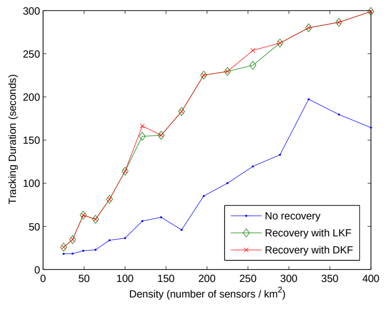
We run each simulation scenario 20 times with each simulation duration of $300s$. The target event is active from the beginning to the end of every simulation run. The simulation area is a $1000m \times 1000m$ square. We assume an omni-directional radio model with a transmission range of $100m$. The sensing ranges of $100m$ and $50m$ are tested. If the target is within the sensing range of a tasked sensor, the sensor is able to estimate the location of the target. The distribution of the localization error is modeled using a zero-mean Gaussian distribution with standard deviation $\sigma_R = 20m$. Targets characterized by mobility patterns with larger standard deviations are more difficult to track. Every tasked sensor estimates the location of the target once in every sampling interval of $1s$. The timeout value for the recovery process is $20s$.

For the mobility of mobile sensors and the target, we consider the Constant Velocity model [3], which is the underlying model that MetroTrack uses for the Kalman filters, as discussed in Section 3. Initially, mobile sensors are randomly placed according to a uniform distribution on the plane. The standard deviation of the movement dynamics of the target and sensor nodes σ_0 is $0.2m/s$. When a target or sensor node reaches the boundary of the simulation area, it changes its direction toward one of the other sides of the simulation area. We have also simulated two other widely used mobility models, the Random Way-point model [9] and the Manhattan model [2]. The results are basically similar to the simulation study using the Constant Velocity model.

One of the main objectives of MetroTrack is to track the target for as long as possible without losing it. Therefore, the duration of tracking is one of the main performance metrics. Figure 4 shows the tracking duration with varying densities and sensing ranges of mobile sensors. We measure the duration of tracking when MetroTrack performs the information-driven tasking but it does not perform the prediction-based recovery (no recovery). We then measure the duration of tracking when MetroTrack performs the prediction-based recovery as well. We compare the tracking duration when MetroTrack uses the Distributed Kalman filter (Recovery with DKF) and when it uses the Local Kalman filter (Recovery with LKF). The x-axis is the density of sensors, and the y-axis is the duration of tracking. We run the simulation for 300 s. The tracking starts from the beginning of the simulation. The target is lost before the simulation ends. As observed in Figure 4, the prediction-based recovery prolongs the duration of the tracking. Moreover, the recovery enables the tracking to last until the end of simulation with the densities of greater than 200 sensors or more per km^2 if the sensing range is $100m$. If the sensing range is 50 m as in Figure 4(b), the recovery enables the tracking to last until the end with a density of 400 sensors. The



(a) Sensing range of 100 m.



(b) Sensing range of 50 m.

Fig. 4: Tracking duration vs. density of mobile sensors.

extended duration by the recovery process is longer for the 50m sensing range than for the 100m sensing range. It is interesting that the recovery processes using both filters do not show much difference in tracking duration, whereas the DKF showed better accuracy in prediction. The forwarding zone is the sum of the radius of the recovery region, the sensing range, and the communication range, as we explained earlier. The size of the forwarding zone is big enough to absorb the impact of the inaccuracy of the prediction of the LKF.

6 Summary

In this paper, we proposed MetroTrack, the first distributed tracking system that tracks mobile events using off-the-shelf mobile phones. We presented the design and implementation of the system and discussed the mathematical foundations upon which our distributed prediction models are based. We evaluated the system through the deployment of a prototype implementation of the system using Nokia N80 and N95 mobile phones and analyzed the performance of the system for a number of different scenarios through simulation. While the proof-of-concept prototype implementation of MetroTrack focused on tracking a mobile audio source, we believe that the algorithms and techniques discussed in this paper are more broadly applicable to an emerging class of problems related to the efficient tracking of mobile events using off-the-shelf mobile devices such as mobile phones, PDAs, and mobile embedded sensors.

7 Acknowledgement

This work is supported in part by Intel Corp., Nokia, NSF NCS-0631289, and the Institute for Security Technology Studies (ISTS) at Dartmouth College. ISTS support is provided by the U.S. Department of Homeland Security under award 2006-CS-001-000001, and by award 60NANB6D6130 from the U.S. Department of Commerce. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of any funding body.

References

1. T. Abdelzaher, Y. Anokwa, P. Boda, J. Burke, D. Estrin, L. Guibas, A. Kansal, S. Madden, and J. Reich. Mobiscopes for human spaces. *IEEE Pervasive Computing*, 6(2):20–29, 2007.
2. F. Bai, N. Sadagopan, and A. Helmy. IMPORTANT: A framework to systematically analyze the Impact of Mobility on Performance of RouTing protocols for Adhoc NeTworks. In *INFOCOM 2003*, San Francisco, CA, USA, Apr. 2003.
3. T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. *Wireless Communications and Mobile Computing. Special issue on Mobile Ad Hoc Networking*, 2(5):483–502, 2002.
4. A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao. Habitat Monitoring: Application Driver for Wireless Communications Technology. In *Workshop on Data Communications in Latin America and the Caribbean*, Apr. 2001.
5. D. Cuff, M. Hansen, and J. Kang. Urban sensing: Out of the woods. *Communications of the ACM*, 51(3):24–33, 2008.
6. L. Girod, M. Lukac, V. Trifa, and D. Estrin. The design and implementation of a self-calibrating distributed acoustic sensing platform. In *4th international conference on embedded networked sensor systems (SenSys'06)*, pages 71–84, 2006.
7. T. He, S. Krishnamurthy, J. A. Stankovic, T. Abdelzaher, L. Luo, R. Stoleru, T. Yan, L. Gu, G. Zhou, J. Hui, and B. Krogh. VigilNet: An Integrated Sensor Network System for Energy-Efficent Surveillance. *ACM Transactions on Sensor Networks*, 2004.
8. Q. Huang, C. Lu, and G.-C. Roman. Spatiotemporal Multicast in Sensor Network. In *First ACM Conference on Embedded Networked Sensor Systems (SenSys'03)*, 2003.
9. D. Johnson and D. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. *Mobile Computing*, pages 153–181, 1996.
10. R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME Journal of Basic Engineering*, March 1960.
11. A. Kansal, A. A. Somasundara, D. D. Jea, M. B. Srivastava, and D. Estrin. Intelligent fluid infrastructure for embedded networks. In *MobiSys'04*, pages 111–124, New York, NY, USA, 2004. ACM.
12. Y.-B. Ko and N. Vaidya. Geocasting in Mobile Ad Hoc Networks: Location-based Multicast Algorithms. In *Workshop on Mobile Computer Systems and Applications (WMCSA'99)*, Feb. 1999.
13. K. Lai, M. Feldman, Stoica, and J. Chuang. Incentives for cooperation in peer-to-peer networks. In *Workshop on Economics of Peer-to-Peer Systems*, 2003.
14. Nokia. Python for s60. http://wiki.opensource.nokia.com/projects/PyS60.
15. R. Olfati-Saber. Distributed Kalman Filtering for Sensor Networks. In *46th IEEE Conference on Decision and Control*, Dec. 2007.
16. Purdue University. Cell phone sensors detect radiation to thwart nuclear terrorism. http://news.uns.purdue.edu/x/2008a/080122FischbachNuclear.html.
17. J. Shneidman and D. C. Parkes. Rationality and self-interest in peer to peer networks. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.