

# DTTC: Delay-Tolerant Trajectory Compression for Object Tracking Sensor Networks

Yingqi Xu \* Wang-Chien Lee \*

Department of Computer Science and Engineering  
The Pennsylvania State University  
University Park, PA 16801, USA  
{yixu, wlee}@cse.psu.edu

## Abstract

*Taking advantage of the delay tolerance for objects tracking sensor networks, we propose delay-tolerant trajectory compression (DTTC) technique, an efficient and accurate algorithm for in-network data compression. In DTTC, each cluster head compresses the movement trajectory of a moving object by a compression function and reports only the compression parameters, which drastically reduces the total amount of data communications required for tracking operations. DTTC supports a broad class of movement trajectories using two techniques, DC-compression and SW-compression, which are designed to minimize the total number of segments to be compressed. Furthermore, we propose an efficient trajectory segmentation scheme, which helps both compression techniques to compress movement trajectory more accurately at less computation cost. An extensive simulation has been conducted to compare DTTC with competing prediction-based tracking technique, DPR [14]. Simulation results show that DTTC exhibits superior performance in terms of accuracy, communication cost and computation cost and soundly outperforms DPR with all types of movement trajectories.*

## 1 Introduction

Object tracking is a killer application of wireless sensor networks. In this application, sensor nodes collectively monitor and track the movements of a moving object. As other types of sensor networks, object tracking sensor networks are characterized by limited onboard and non-refreshable battery power, and limited computation and storage resources. This has spurred a need for designing object tracking techniques that are tailored specifically toward tracking applications and make optimized tradeoff between the energy consumption, network performance and operation fidelity.

A wide range of object tracking applications can be categorized into real-time tracking and delay-tolerant tracking. In real-time object tracking, the sensor readings have to be collected by the sink node in a real-time manner. For instance, a battle field surveillance application usually requires real-time tracking of enemy soldiers for making prompt reactions. On the other hand, in delay-tolerant object tracking, a sink node can tolerate a delay in collecting sensor readings about object

movements. For example, in wildlife habitat monitoring, scientists may only need to know the movement trajectory of a group of zebras in most of the time, instead of their real-time movements.

The existing tracking techniques are usually designed for real-time object tracking in sensor networks. In order to avoid the continuous updates about locations of moving objects from sensor nodes to the sink node, many existing tracking techniques employ prediction models for estimating the future location of a moving object based on its movement history [1, 5, 8, 13, 14, 15]. When the prediction made by a sink node is correct, sensor nodes do not report their readings to the sink node. Otherwise, sensor nodes correct the sink node by sending their readings. However, in practice, the movement trajectory of moving objects is complex, and individual object may exhibit completely different movement patterns, which poses significant challenges for a prediction-based tracking technique to accurately predict the object movements. Moreover, some prediction-based techniques predict an object's future movements based on the movement history of other moving objects. However, these techniques are based on the assumption that there are a limited number of movement states (e.g., moving speed and direction) that a moving object can have, such that the probability for an object taking each movement state can be calculated and used for predictions.

In addition to the above deficiencies, the prediction-based tracking technique is not the best choice for delay-tolerant object tracking, which, by releasing the requirements on timeliness domain, provides significant opportunities for reducing network communication cost in tracking operations. In this paper, we investigate a novel technique, i.e., *delay-tolerant trajectory compression* (DTTC) for delay-tolerant object tracking sensor networks. DTTC can present a sink node accurate movement trajectory of a moving object and can significantly decrease the communication cost, by compressing the sensor readings into a set of compression parameters. Specifically, a cluster infrastructure is organized inside network, and each sensor node sends its readings to the cluster head for further signal processing and data fusion. Instead of sending object's locations at consecutive time instants to the sink node, the cluster head compress those data by a compression function, such that only the compression parameters that models the movement trajectory of tracked object are sent to the sink node. The core of our DTTC technique is *trajectory segmentation*, which allows a complex movement trajectory to be compressed accurately with

\*Wang-Chien Lee and Yingqi Xu were supported in part by National Science Foundation grant IIS-0328881.

the minimum number of compression parameters. Two compression techniques, *DC-compression* and *SW-compression* are studied for two different object movement patterns.

Our contribution can be summarized as follows:

- We propose DTTC technique for delay-tolerant object tracking in wireless sensor networks. DTTC compresses a complex movement trajectory into a set of compression parameters, which significantly decreases the total amount of data communications required for tracking operations.
- Two trajectory compression techniques, i.e., DC-compression and SW-compression, are proposed to reduce the computation overhead for two different movement patterns that a moving object may follow.
- We carefully craft a trajectory segmentation scheme, which wisely segments a trajectory with one scan of the data collected from a cluster.
- We provide an extensive experimental study of proposed DTTC technique with various movement trajectories (i.e., both linear and non-linear movements), and make comparisons against a prediction-based tracking technique. DTTC demonstrates superior performance in compression accuracy, communication cost and computation cost.

The rest of the paper is organized as follows. Section 2 examines the related work. In section 3, we state the problem to be tackled, and present the design of DTTC. Section 4 reports experimental results and finally Section 5 presents concluding remarks and future directions.

## 2 Related Work

DTTC technique, to our best knowledge is the first trajectory compression technique proposed for delay-tolerant object tracking sensor networks. However, our work is inspired by a number of related research efforts.

Object tracking, concerned with approximating the trajectory of a moving object based on some partial information provided by sensor nodes, has been extensively studied [1, 8, 9, 12, 13, 14, 15, 17, 18]. All these research work target on minimizing the total amount of data communications. We identify three different, while complementary, techniques for communication reduction:

**Filtering** restricts the data generated by the sensor nodes by either reducing the sampling frequency or filtering sensor readings locally. Filtering is simple and can be performed by each sensor node independently, but is likely to jeopardize tracking accuracy, as less information is available for deriving the movement states of tracked objects.

**Compression** transforms a large amount of raw sensor readings into a small amount of information that can be recovered to describe the object movement states. The compression usually takes place at a set of selected nodes that collect the readings from a group of sensor nodes over a period of time. Compression reduces the communication cost by sending compressed information, at the cost of reduced tracking accuracy and longer data collection delay experienced at the sink node.

**Prediction** techniques are based on the assumption that the overall movement states (e.g., direction and speed) of tracked objects usually remain relatively stable for a period of time [1, 5, 8, 9, 18]. A sink node, based on the information extracted from its movement history, predicts the future

movement states of an object, which are sent to the corresponding sensor nodes. If the prediction matches the sensor readings, the prediction is correct. Otherwise, the sensor nodes correct the sink node by sending their own readings. [13, 14, 15] further improve this technique by saving the transmissions of prediction packets from the sink node to sensor nodes. They propose a dual prediction scheme, in which the predictions take place at both sensor nodes and the sink node. To make predictions about object's future movement, sensor nodes pass the movement history of tracked object along its movement trajectory.

Based on the movement history used for prediction, prediction technique can be classified into *prediction with individual history* and *prediction with group history*. The prediction with individual history (e.g., [5, 13, 14, 15]) predicts the movement of an object from its own history. Considering in practice an arbitrary movement trajectory that an object may follow, the simple prediction models in the existing work results in poor prediction performances. The prediction with group history attempts to categorize the objects into groups, and the prediction of a moving object is made based on the history of all objects from the same group. Group history provides richer information about the object movements than the individual history, thus more advanced techniques can be applied to extract the movement pattern for a group of objects (e.g., [1, 3, 4, 8, 11]). These techniques usually require building models from a large amount of history movements before making predictions. Moreover, those approaches assume a limited number of movement states (e.g., moving speed and direction) that a moving object can have, such that the possibility for an object taking each movement state can be calculated and used for future predictions. In addition, prediction with group history does not consider the case that, even within the same group, different objects may demonstrate different movement patterns at different times (e.g., morning, noon and night) and/or with different tasks (e.g., surveillance and disaster response).

## 3 Design of DTTC

We first present the problem that we tackle in this paper in Section 3.1, and present the basic idea of delay-tolerant trajectory compression in Section 3.2. The detailed compression techniques and trajectory segmentation schemes are discussed in Section 3.3 to Section 3.4, respectively.

### 3.1 Problem Setting

We assume a *sensing interval*, a period of time between two consecutive sensing operations conducted by a sensor node has been defined by the object tracking applications. To simplify our presentation, we assume the sensing operations conducted by sensor nodes are synchronized. However, our work can be easily adapted to unsynchronized case. At time  $t$ , a set of sensor nodes are required to participate in tracking a moving object. A cluster infrastructure is formed within the network field and sensor nodes report their readings to the cluster head for data fusion<sup>1</sup>. A cluster head receives a sequence of sensor reading  $\langle t, R, S_{id}, O_{id} \rangle$ , where  $R$  represents the sensor readings from node  $S_{id}$  about moving object  $O_{id}$  stamped at time  $t$ . By processing the sensor readings  $R$  with the same  $t$  and  $O_{id}$ , the cluster head derives the movement state, i.e., geographical location, for object  $O_{id}$

<sup>1</sup>Within each cluster, a smaller-sized tree/cluster can be used for data fusion [10, 14], which further reduces the total amount of data communications.

at time  $t$ , denoted as  $\langle t, x, y, O_{id} \rangle$ , where  $x$  and  $y$  are the geographical coordinates for  $O_{id}$  at  $t$  assuming a two-dimensional space. Thus, the input to our problem is the movement trajectory of  $O_{id}$ , represented by a sequence of geographical locations at time  $t_1, t_2, \dots$ , i.e.,  $\mathbf{L} = \{l_1, l_2, l_3, \dots\}$ , where  $l_i = \langle t_i, x_i, y_i, O_{id} \rangle$ . The cluster head issues a trajectory compression request upon  $\mathbf{L}$ . The output is sets of  $(\beta, \epsilon) := \text{Trajectory\_Compression}(\mathbf{L})$ , in which  $\beta$  is the compression parameters used in compressing one trajectory segment, and  $\epsilon$  is the corresponding compression error that should be less than the application-defined error bound for compression  $\xi$ . After compressing the movement trajectory of  $O_{id}$  within its cluster, the cluster head reports  $\beta$  and other needed information (which will be discussed shortly in Section 3.2) to the sink node.

To minimize the total amount of data communications, the compression, with insurance of  $\epsilon < \xi$  for each trajectory segment, needs to be maximized, i.e., the least amount of compression parameters are needed for reconstructing the movement trajectory of a moving object. In the following, we address this problem by discussing both compression techniques and trajectory segmentation scheme.

### 3.2 Basic Idea of Trajectory Compression

To compress movement trajectory  $\mathbf{L} = \{l_1, l_2, \dots\}$ , a compression function is needed to capture those location points  $l_i$  along the movement trajectory. For instance, a quadratic function, i.e.,  $\hat{y} = \beta_0 + \beta_1 x + \beta_2 x^2$  can depict many kinds of moving pattern with different parameter  $\beta_0, \beta_1$  and  $\beta_2$ . This compression function can also capture linearity as a special case. However, with a degree of 2, a quadratic compression function can only capture a trajectory with one turning point, and is not applicable for more complicated trajectories (e.g., with more turning points). Moreover, a higher degree compression function may not only increase the computation complexity but may also not be sufficient for all kinds of movement trajectory.

Our solution to the above problem is based on the observation that a trajectory can be segmented, and the piece-wise segments can be approximated with a relatively simple compression function (e.g., a quadratic model or even a linear model when the segments are small enough). A segment  $\mathcal{S}$  is characterized by a set of parameters

- $t_s$  and  $t_e$ : the timestamps representing the time when the first and last  $l$  in segment  $\mathcal{S}$  are generated by the sensor nodes, i.e.,  $\mathcal{S}(t_s, t_e) = \{l_i = \langle t_i, x_i, y_i, O \rangle \mid t_s \leq t_i < t_e\}$ , and thus determining the length of a segment. We denote  $x(t_s, t_e) = \{x_i \mid x_i \in \mathcal{S}(t_s, t_e)\}$ , and  $y(t_s, t_e) = \{y_i \mid y_i \in \mathcal{S}(t_s, t_e)\}$ .
- $x_{t_s}$  and  $x_{t_e}$ : the x-coordinates of the first  $l_{t_s}$  and of the last  $l_{t_e}$  in segment  $\mathcal{S}$ . These two values are used to reconstruct the movement trajectory of an object.
- $\beta$  and  $\epsilon$ :  $\beta$  represents the compression parameters of segment  $\mathcal{S}$ , and  $\beta = (\beta_1, \beta_2, \dots, \beta_m)$ .  $\epsilon$  is the error introduced by the compression on a segment  $\mathcal{S}$ . We adopt the *root mean squared error (RMS)*, which is widely used as the error metric in regression techniques. Other error metrics can also be used by DTTC with minor changes.

Subroutine `Seg_Compression()` shown in Algorithm 1 is at the core of DTTC. This function compresses a segment of movement trajectory, i.e.,  $\mathcal{S}(t_s, t_e)$ , and computes the regression parameters  $\beta$ , as well as the compression error  $\epsilon$ . The compression function is represented as

$y(t_s, t_e) = \sum_{j=0}^m \Phi_j(x(t_s, t_e))\beta_j$  where  $\Phi_0, \Phi_1, \dots, \Phi_m$  are functions of  $x(t_s, t_e)$ . For instance, when  $\Phi_0 = 1, \Phi_1 = x(t_s, t_e), \dots, \Phi_m = x(t_s, t_e)^m, y(t_s, t_e) = \beta_0 + \beta_1 x(t_s, t_e) + \dots + \beta_m x(t_s, t_e)^m$ .

By assuming a sampling interval as one time unit, compression function can be represented in vector notation,

$$\begin{bmatrix} \hat{y}_{t_s} \\ \hat{y}_{t_{s+1}} \\ \vdots \\ \hat{y}_{t_e} \end{bmatrix} = \begin{bmatrix} \Phi_0(x_{t_s}) & \Phi_1(x_{t_s}) & \dots & \Phi_m(x_{t_s}) \\ \Phi_0(x_{t_{s+1}}) & \Phi_1(x_{t_{s+1}}) & \dots & \Phi_m(x_{t_{s+1}}) \\ \vdots & \vdots & \ddots & \vdots \\ \Phi_0(x_{t_e}) & \Phi_1(x_{t_e}) & \dots & \Phi_m(x_{t_e}) \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_m \end{bmatrix}$$

---

#### Algorithm 1 Seg\_Compression()

---

**Require:**  $t_s, t_e, \mathbf{x}(t_s, t_e), \mathbf{y}(t_s, t_e)$

- {Compression of Segment  $\mathcal{S}(t_s, t_e)$
  - 1:  $N(t_s, t_e) \leftarrow$  the total number of entries in Segment  $\mathcal{S}(t_s, t_e)$
  - 2:  $\Phi_i \leftarrow$  functions of  $\mathbf{x}(t_s, t_e)$   
{compute the compression parameters}
  - 3:  $\beta = \left( (\mathbf{x}(t_s, t_e))^T \mathbf{x}(t_s, t_e) \right)^{-1} \left( \mathbf{x}(t_s, t_e)^T \mathbf{y}(t_s, t_e) \right)$   
{compute the compression RMS error for segment  $\mathcal{S}(t_s, t_e)$ }
  - 4:  $\epsilon(t_s, t_e) = \sqrt{\frac{1}{N(t_s, t_e)} \sum_{i=0}^{N(t_s, t_e)-1} \left( y_{t_s+i} - \sum_{j=0}^m \beta_j \Phi_j(x_{t_s+i}) \right)^2}$
  - 5: return  $(\beta, \epsilon)$
- 

Algorithm 1 can be executed once a segment is identified. A naive way of constructing segments is to divide the movement trajectory  $\mathbf{L}$  randomly/evenly into a number of segments, and strictly applying algorithm 1 on each segment. However, as our goal is to minimize the total number of compression parameters and to reduce data communications, random segmentation cannot guarantee that DTTC reaches the goal. There are two possible approaches to the above problem: 1) to minimize the total number of segments, which, however, may generate more parameters for each segment, as a longer segment of movement trajectory likely requires a more complicated compression function; 2) to minimize the number of parameters (i.e.,  $\beta$ ) for each segment, which implies using simpler compression functions. In this case, to ensure  $\epsilon < \xi$ , the total number of segments may increase. As the compression function is usually pre-determined by applications, we study two compression techniques that for a given compression function (e.g., a  $m$ -degree of polynomial compression function), aim at minimizing the total number of segments to be compressed.

### 3.3 Compression Techniques

We introduce two compression techniques, namely Divide-and-Conquer compression (denoted as DC-compression) and Stepwise compression (denoted as SW-compression), which both aim at minimizing the total number of segments and reducing data communications, with a given compression function. Meanwhile, these two techniques target different movement patterns that an object may follow. We assume that  $\mathbf{L}$  is stored in an ascending chronological order at a cluster head.

#### 3.3.1 DC-compression

The DC-compression is based on the idea of divide-and-conquer. The compression takes place after the cluster head obtains all data that need to be compressed.

We assume there are  $N$  data entries available at a cluster head when it performs compression, denoted as  $\mathbf{L} = \{ \langle t_1, x_1, y_1, O \rangle, \langle t_2, x_2, y_2, O \rangle, \dots, \langle t_N, x_N, y_N, O \rangle \}$  for object  $O$ . The DC-compression, shown in Algorithm 2, works recursively. At the beginning, it takes the entire trajectory inside the cluster, i.e.,  $\mathbf{L}$ , as one segment. In each iteration, the segment with the largest compression error is selected and divided into two segments at timestamp  $t_{div}$ . A critical research issue in DC-Compression is to select  $t_{div}$  for the segment that has the largest compression error in an iteration, which will be discussed in detail shortly in Section 3.4. The above process repeats, until the maximum number of segments is exhausted or the compression errors of all segments are less than the error bound  $\xi$ . The compression parameters for a segment can be reported to the sink node once its compression error  $\epsilon < \xi$ .

---

#### Algorithm 2 DC\_Compression()

---

**Require:**  $\mathbf{x}$  and  $\mathbf{y}$

- 1:  $\epsilon_{max} \leftarrow$  the maximal  $\xi$  of all segments
  - 2:  $t_{smax}$  and  $t_{emax}$   $\leftarrow$  the  $t_s$  and  $t_e$  of Segment  $\mathcal{S}(t_s, t_e)$  with  $\epsilon_{max}$
  - 3:  $\xi \leftarrow$  the user-defined error bound
  - 4:  $K_{max} \leftarrow$  the user-defined maximal number of segments
  - 5:  $K_{seg} \leftarrow$  the total number of segments  
  {Initialization}
  - 6:  $\epsilon_{max} = 0, K_{seg} = 1$
  - 7:  $t_{smax} = t_1$
  - 8:  $t_{emax} = t_N$
  - 9:  $\epsilon_{max} = \text{Seg\_Compression}(t_{smax}, t_{emax})$   
  {Divide-and-conquer Compression}
  - 10: **while** ( $K_{seg} \leq K_{max}$ ) AND ( $\epsilon_{max} > \xi$ ) **do**
  - 11:    $\epsilon_{max} =$  maximal  $\epsilon$  among all segments
  - 12:   update  $t_{smax}$  and  $t_{emax}$
  - 13:   select  $t_{div}$  {divide the  $\mathcal{S}(t_{smax}, t_{emax})$  into two Segments at  $t_{div}$ }
  - 14:    $\epsilon(t_{smax}, t_{div} - 1) = \text{Seg\_Compression}(t_{smax}, t_{smax} + t_{div} - 1)$
  - 15:    $\epsilon(t_{smax} + t_{div}, t_{emax}) = \text{Seg\_Compression}(t_{smax} + t_{div}, t_{emax})$
  - 16:    $K_{seg} = K_{seg} + 1$
  - 17: **end while**
- 

### 3.3.2 SW-Compression

The DC-compression aims at maximizing the size of each segment and reducing the total number of segments. However, DC-compression incurs high computation cost when the movement trajectory within a cluster in nature has to be divided into multiple segments for compression, since its compression starts from the entire trajectory within a cluster.

In this section, we introduce another compression technique, called Stepwise compression, or SW-compression for brief. The core idea of SW-compression is as same as DC-compression in that an object's movement trajectory  $\mathbf{L} = \{l_1, l_2, \dots, l_N\}$  is segmented, and the algorithm Seg\_Compression is applied to each segment. Taking a different approach, SW-compression first divides the trajectory into segments, and the compression starts from a single segment, instead of the entire trajectory within the cluster as DC-compression does. SW-compression tries to compress as many segments as possible by one compression function. More specifically, SW-compression involves 1) selecting an

initial segment and compressing it; 2) adding the next new segment to the current segment that has been compressed, compressing and obtaining new compression parameters and errors; 3) terminating step 2 when either no new segment is left or the obtained compression error exceeds  $\xi$ . The above process is repeated until the maximum number of segments is exhausted or there is no segment left. Algorithm 3 depicts SW-compression technique.

---

#### Algorithm 3 SW\_Compression()

---

**Require:**  $\mathbf{x}$  and  $\mathbf{y}$

- 1: starts with a segment  $\mathcal{S}(t_1, t_e)$
  - 2:  $(\beta, \epsilon) = \text{Seg\_Compression}(\mathcal{S}(t_1, t_e))$   
  {Stepwise compression}
  - 3: **while** ( $t_e \leq t_N$ ) **do**
  - 4:   **while** ( $\epsilon \leq \xi$ ) AND ( $t_e \leq t_N$ ) **do**
  - 5:      $\beta_{pre} = \beta, \epsilon_{pre} = \epsilon, t_{pre} = t_e, x_{pre} = x_{t_e}$
  - 6:     selects the next segment  $\mathcal{S}(t_e + 1, t_{next})$
  - 7:     form a new segment by  $\mathcal{S}(t_s, t_e) \cup \mathcal{S}(t_e + 1, t_{next})$ ;
  - 8:      $t_e = t_{next}$
  - 9:      $(\beta, \epsilon) = \text{Seg\_Compression}(\mathcal{S}(t_s, t_e))$
  - 10:   **end while**
  - 11:   return segment  $\mathcal{S}(t_s, t_{pre}), \beta_{pre}, \epsilon_{pre}$  and  $x_{pre}$   
  {Prepare for the next SW-compression}
  - 12:   select a new segment  $\mathcal{S}(t_{pre} + 1, t_{next})$
  - 13:    $t_s = t_{pre} + 1$
  - 14:    $t_e = t_{next}$
  - 15:    $(\beta, \epsilon) = \text{Seg\_Compression}(\mathcal{S}(t_s, t_e))$
  - 16: **end while**
- 

Segmentation is critical for SW-compression. Similar to the discussion about DC-compression, a straightforward approach for constructing segments in SW-compression is to divide  $\mathbf{L}$  into a number of segments with the same size. However, the size of a segment is hard to determine. If the size of a segment is too small, the computation cost would be high, as the compression has to be conducted whenever a new segment is added. This computation overhead seems especially unnecessary, when the movement trajectory of an object is smooth, which likely requires less number of segments. On the other hand, when the size of a segment is large and the movement trajectory of an object is complex (e.g., with many turning points), the compression error would increase. In this case, a large segment has to be further divided into small pieces. Thus, for both DC-compression and SW-compression, trajectory segmentation has to be conducted by taking into consideration the movement trajectory of the moving objects, for reducing both computation cost and compression error.

### 3.4 Trajectory Segmentation

A wise trajectory segmentation scheme should satisfy the following two requirements: 1) the formed segments have the right size. More specifically, if a segment  $\mathcal{S}(t_s, t_e)$  can be compressed by a compression function with  $\epsilon \leq \xi$ , and if adding more data into  $\mathcal{S}(t_s, t_e)$  from either direction for compression, i.e.,  $l_{t_{s-1}}, l_{t_{s-2}}, l_{t_{s-3}}, \dots$  and  $l_{t_{e+1}}, l_{t_{e+2}}, l_{t_{e+3}}, \dots$ , results in  $\epsilon > \xi$ ,  $\mathcal{S}(t_s, t_e)$  has the right size; 2) trajectory segmentation should not incur excessive computation overhead. In the following, we study a trajectory segmentation scheme that divides a given trajectory into segments with one scan of all data needed to be compressed, and focuses on forming the

segments that is with the right size, while is resilient to the randomness in object's movement.

Studying a movement trajectory, the turning points along the trajectory, represented as *local maxima* and *local minima* (together refereed as *local extremum*) on both y-axis and x-axis, and dramatic changes in moving direction are the main reason that a compression function cannot compress a complex movement trajectory accurately. Therefore, by examining the location points  $l_1, l_2, \dots, l_N$  along the movement trajectory of a moving object, the cluster head can perform segmentation along  $t$  when local extremum on either x- or y-axis are identified, or a dramatic change in moving direction happens. The methods for identifying local extremum and for identifying the changes in moving direction are similar. Due to the space limitation, we only present how a cluster head segments the movement trajectory based on local extremum, and taking the local extremum on x-axis as example. Y-axis can be processed in the same manner.

The cluster head continuously obtains a sequence of location points along the object's movement trajectory  $l_1, l_2, l_3, \dots$ . At the beginning, the cluster head initializes  $l_{max} = l_{min} = l_1$ , where  $l_{max} = \langle t_{max}, x_{max}, y_{max} \rangle$  records the local maxima at x-axis and  $l_{min} = \langle t_{min}, x_{min}, y_{min} \rangle$  records the local minima at x-axis. For a new arrival of  $l_i$ , if  $x_i > x_{max}$ ,  $l_{max} = l_i$ ; otherwise if  $l_i < x_{min}$ ,  $l_{min} = l_i$ . With a small enough sampling interval, it is not likely that both  $l_{max}$  and  $l_{min}$  are to be set as  $l_i$ . We denote  $\vec{d}(t_i)$  as a general moving direction of the moving object between  $t_i$  and  $t_{i+1}$ .  $\vec{d}(t_i) = 1$ , when  $x_{i+1} > x_i$ , otherwise  $\vec{d}(t_i) = -1$ . In reality, objects move with randomness and their movement trajectory would not be perfectly smooth, such that there exists a large amount of local extremum. In this case, a cluster head, when forming the segments needs to ignore *bumps* that are the local extremum slightly deviated from the overall movement trajectory of a moving object. Therefore, if  $\vec{d}(t_i)\vec{d}(t_{i+1}) < 0$ , the cluster head records it as a bump  $l_{bump} = l_{i+1}$ , and examines the location points  $l_{i+2}, l_{i+3}, \dots$  to decide whether it is a bump, or a new segment should be formed. There are two possible cases:

- $\vec{d}(t_i) = 1$ : Let  $j = 2, 3, \dots$ . If  $\vec{d}(t_i)\vec{d}(t_{i+j}) > 0$ , the cluster head considers location points  $l_{i+2}, l_{i+3}, \dots, l_{i+j-1}$  as small deviations from the object's movement trajectory, ignores them in segmentation, and clears  $l_{bump}$ . Otherwise, if  $\vec{d}(t_i)\vec{d}(t_{i+j}) < 0$ , the cluster head calculates the geographical distance between  $l_{bump}$  and  $l_{i+j+1}$ , i.e.,  $B = ((x_{bump} - x_{i+j+1})^2 + (y_{bump} - y_{i+j+1})^2)^{1/2}$ . If  $B \geq B_{max}$ , the cluster head forms a segment between  $t_{min}$  and  $t_{max}$ , clears the  $l_{bump}$ , and resets  $l_{max} = l_{i+1}$  and  $l_{min} = l_{i+j+1}$  for the next segment. If  $B < B_{max}$ ,  $j$  increases by 1.
- $\vec{d}(t_i) = -1$ : This is similar to the previous case, except that the new segment is formed by setting  $l_{min} = l_{i+1}$  and  $l_{max} = l_{i+j+1}$ .

The above trajectory segmentation only requires one scan of all data to be compressed, to find out the local extremum for constructing segments. Each formed segment tolerates small deviations in object movements.

## 4 Performance Evaluation

In this section, we evaluate the performance of DTTC technique and compare it with a prediction-based tracking

technique, called DPR [14].

### 4.1 Simulation Settings

This section lays out the basic simulation settings, including the movement trajectories that a moving object may follow, and the network settings.

We implement all schemes under comparison in MATLAB. We assume a sensor network is deployed in a  $1000m \times 1000m$  two-dimensional region. Without losing generality, we assume that in DTTC the network region is divided evenly into a set of clusters, and a cluster head is placed in the middle of each cluster.<sup>2</sup> The transmission range of a sensor node is set to  $40m$  [7], and the node density, i.e., the average number of neighbor nodes in the transmission area of a sensor node, is 20. The number of hops for transmitting a packet is approximated as the ratio of distance between a source node and a destination node over the transmission range of a sensor node (i.e.,  $40m$ ).

To simulate various movement trajectories that a moving object may follow, we consider two kinds of trajectory, i.e., *linear movement trajectory* and *nonlinear movement trajectory*. With linear movement trajectory, an object moves with a fixed moving direction for a period of time before changing its direction. On the other hand, an object following nonlinear trajectory keeps changing its moving direction during its movement. To simulate linear movement trajectory, we employ the following two mobility models:

**Random waypoint mobility model (RWP)**: RWP (as shown in Figure 1(a)) runs as follows: at every instant, an object randomly chooses a destination and moves towards it with a velocity. Even though the above RWP model has been criticized for its unstable movements [16], DTTC focuses on extracting the object's movement trajectory, and is less affected by the object's movement properties (e.g., moving speed).

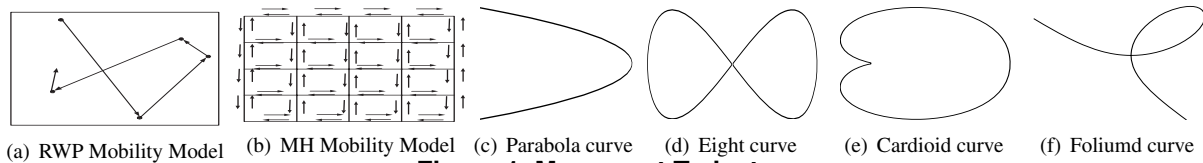
**Manhattan mobility model (MH)**: Different from RWP, MH (as shown in Figure 1(b)) emulates the movement pattern of a moving object on streets defined by a map. We adopt the settings used in [2]: at the intersection of streets, a moving object can turn left, turn right and go straight with a probability of 0.25, 0.25 and 0.5, respectively<sup>3</sup>.

The nonlinear movement trajectories are simulated by some famous mathematical curves. Specifically, we consider *parabola curves* generated by  $y = ax^2 + bx + c$ , *eight curves* generated by formula  $x^4 = a^2(x^2 - y^2)$ , *cardioid curves* generated by formula  $(x^2 + y^2 - 2ax)^2 = 4a^2(x^2 + y^2)$ , and *folium curves* generated by  $x^3 + y^3 = 3axy$ . Figure 1(c) to 1(f) show the movement trajectories defined by the above curves. We simulate the randomness in object movements by adding a *bump* randomly drawn from  $[-\alpha * 50, \alpha * 50]$  to the location points on the defined nonlinear movement trajectories<sup>4</sup>, such that the real movement trajectories are not as smooth as the ones shown by Figure 1. The threshold  $B_{max}$  is set to  $40m$ . Since object's moving speed does not affect the performance of DTTC, we assume the object moves at a

<sup>2</sup>The impact of cluster properties (e.g., cluster shape and the position of the cluster heads) on the performance of DTTC will be studied in our future work.

<sup>3</sup>When the object reaches the boundary of the simulated field, excluding the corners, the object turns left, turns right or goes back with a probability of 0.25, 0.25 and 0.5, respectively. When reaching the corner of the simulated field, the object chooses the next moving direction from two possible directions that allow it to remain in the field with a probability of 0.5.

<sup>4</sup>Since MH does not allow movement randomness, to be comparable, we do not consider the bumps in RWP mobility model as well.



**Figure 1. Movement Trajectory**

constant speed  $10\text{ m/s}$ , which however facilitates the predictions in DPR.

For the linear trajectory, the simulation lasts for 600 seconds, and an object starts its movement from the lower-left corner of the simulated field. The simulation results are obtained by averaging the algorithm performance over 10 runs of randomly generated object movement traces. For nonlinear movements, the results are derived by averaging the performance over 10 runs of the same trajectory with randomly introduced bumps.

Since there is no existing delay-tolerant tracking protocol proposed for object tracking sensor networks, we compare our work (DTTC) against dual-prediction reporting scheme (DPR) proposed in [14] for comparison. According to the experimental settings and results reported by [14], we adopt the *EXP\_AVG* prediction model, which has shown to be the best prediction performance at a less communication overhead. The sizes of a history packet and an update packet in DPR are set to 11 bytes. A compression packet in our proposal is 60 bytes. The error bound  $\xi$  is  $40\text{m}$  for both DTTC and DPR. The usage of an error bound in DTTC has been described in Section 3.3; for DPR, the sensor nodes consider the prediction correct, as long as the distance between the predicted and the actual object's location is less than the error bound. In the following experiments, a polynomial function with degree of 3 is used as DTTC's compression function, which has shown competitive overall performance in terms of compression accuracy and communication cost, compared to polynomial functions with higher degree, which, however are not shown in this paper due to the space limitation.

#### 4.2 Experimental Results

We examine the performance of DTTC by varying the number of clusters formed inside the simulated field. The number of clusters has major impact on the communication cost and computation cost of DTTC. More specifically, when the number of clusters decreases, the communication cost within a cluster increases due to the enlarged region that a cluster covers, while the total number of compression packets from all clusters (indicating the communication cost between cluster heads and the sink node) tends to decrease, as a movement trajectory is less likely segmented by the boundary between clusters. The impact of cluster size on the computation cost is not clear. A small cluster naturally facilitates compression, as the movement trajectory within each cluster is short. On the other hand, too many clusters would cause unnecessary segmentation of the movement trajectory. Therefore, in the following, we study the impact of the number of clusters on the performance of DTTC with different movement trajectories.

We study the performance of DTTC and DPR in terms of *distance error*, *communication cost* and *computation cost* in Section 4.2.1, Section 4.2.2 and Section 4.2.3, respectively.

##### 4.2.1 Distance Error

The distance error in DTTC indicates the compression accuracy, defined as the average distance between the computed location from the compression function and the actual location of a moving object. The distance error in DPR, defined

as the average distance between the predicted and the actual location of a moving object, indicates the accuracy of prediction models and has a main impact on the communication overhead of sending updates packets to the sink node.

Figure 2 compares the distance error for DTTC's DC-compression technique, DTTC's SW-compression technique and DPR algorithm with both linear movement trajectories and nonlinear movement trajectories. The total number of clusters is set to 2, 4, 8 and 16, respectively. As DPR does not rely on cluster infrastructure, the performance of DPR remains constant with the varying number of clusters. Figure 2(a) shows experimental results under RWP model and MH model. Both DTTC and DPR have less distance error under MH model than under RWP model. It is because an object in MH model has a maximum of three possible moving directions at a given time, which helps both DTTC and DPR to make wise decisions when an object changes its moving direction. More specifically, DTTC segments the MH movement trajectory whenever an object makes a turn, which results in that all formed segments are straight lines. We also allow DPR to make similar decisions with MH model, such that the future moving direction is predicted as the previous direction observed. In RWP, since a movement destination is randomly chosen within the simulation field, DPR experiences more difficulties in predicting the object's movement than it is in MH. The reason that DTTC under RWP model has higher distance error than under MH model is because of the case that a moving object enters an adjacent cluster soon after it changes its moving direction in the current cluster. In this case, the current cluster tends to combine these two trajectory segments with different moving directions into one segment, and fits the combined segment into one compression function, which results in a slight increase in the distance error.

For the clarity of presentation, we plot the distance error with eight and parabola curves in Figure 2(b), and that with cardioid and folium curves in Figure 2(c), respectively. DC-compression technique and SW-compression technique show similar distance errors, as the compression in these two techniques is controlled by the same error bound. Moreover, it is encouraging to notice that DTTC demonstrates a good compression accuracy with a large cluster size. It implies that when the size of a network increases, with the same number of clusters, the accuracy of compressing trajectories of moving objects would not dramatically deteriorate. Since similar trends are observed from Figure 2(b) and 2(c), which demonstrates higher distance errors in DTTC compression and DPR predictions than eight and parabola curves, the following experiments only present the performance with cardioid and folium curves for nonlinear movement trajectories.

##### 4.2.2 Communication Cost

The communication cost indicates the energy usage for communication in DTTC and DPR. The cost for transmitting a packet is measured as the number of bytes in the packet  $\times$  the number of hops for transmission. The total communication cost of a studied algorithm is calculated as the sum of cost of transmitting all packets.

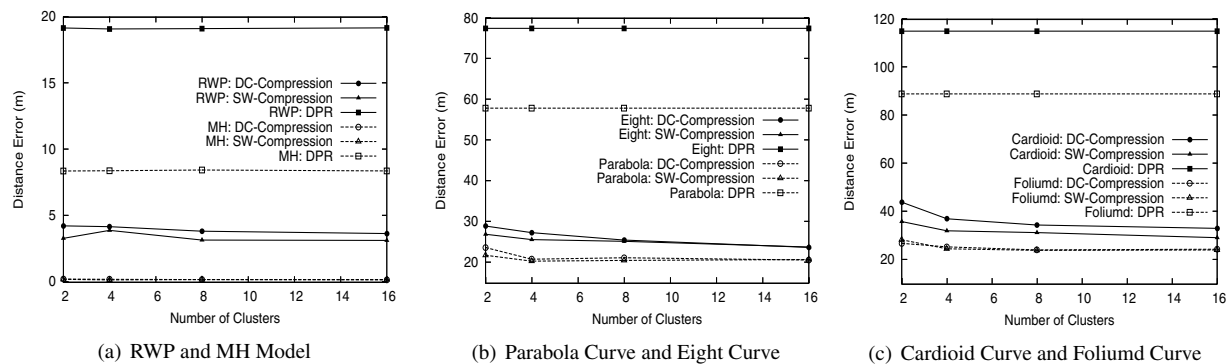


Figure 2. Distance Error

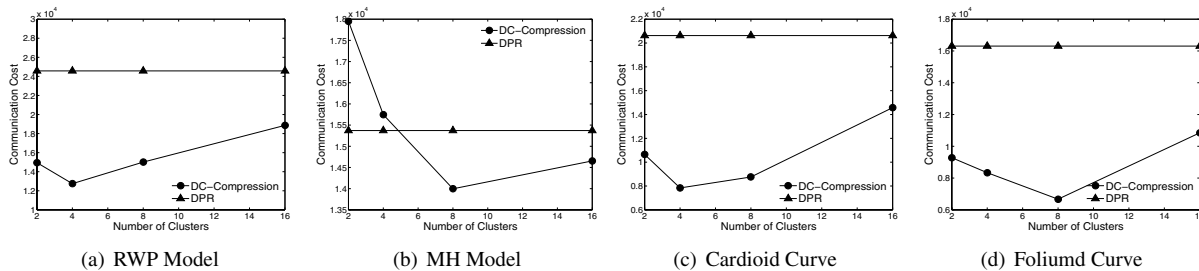


Figure 3. Communication Cost

Figure 3 depicts the communication cost for DPR and DC-compression. Since DC-compression and SW-compression are based on the same cluster infrastructure, and have very close distance error as shown in Figure 2, we do not show the communication cost for SW-compression in Figure 3. Due to the same reason described in Section 4.2.1, DPR under MH model with higher prediction accuracy incurs less communication cost than under RWP model. We observe that DTTC demonstrates a very different performance trend in communication cost under RWP model and MH model. In RWP model, when a small number of clusters are deployed (the number of clusters is less or equal to 4), the communication cost of DTTC decreases due to the shorter transmission distance within a cluster. When more clusters are formed, this saving is quickly overwhelmed by the increasing number of compression packets, which are larger than the data packets transmitted within a cluster. On the other hand, MH benefits from smaller cluster size, as an object in MH changes its moving direction more frequently than in RWP, which leads to shorter trajectory segments in MH. It also explains why DTTC under MH has higher communication cost than under RWP model.

The communication cost for cardioid and folium curves are shown in Figure 3(c) and Figure 3(d). As cardioid and folium curves are averagely shorter than RWP trajectory that an object traverses along, the communication cost of tracking objects with nonlinear trajectories is less than that with RWP model, even though the distance error for RWP is lower than that for cardioid and folium curves. We observe that with nonlinear movement trajectories, DTTC achieves up to 62% savings in communication cost over DPR, more than the improvements in communication cost achieved by DTTC with linear movement trajectories. It is because in nonlinear model, an object keeps changing the moving direction during its movement, which worsens the accuracy of predictions made by DPR about object's future movement (shown in Figure 2(c)) and increases the communication

cost by transmitting more update packets from sensor nodes to the sink node. In addition, when the number of clusters increases, the communication cost for DTTC decreases. However, when the movement trajectory is divided into too many segments, the gain in communication cost is defeated by the increasing cost of sending more compression packets.

#### 4.2.3 Computation Cost

The computation cost for DTTC is defined as the total number of compression operations (i.e., Subroutine Seg\_Compression) needed for compressing the entire movement trajectory of a moving object. The computation cost measures the computation complexity of two compression techniques, i.e., DC-compression and SW-compression.

Figure 4 shows the computation cost of DC-compression and SW-compression techniques with linear and nonlinear trajectories. Generally, SW-compression technique incurs more computation cost than DC-compression with RWP model and with nonlinear model, as DC-compression starts with fitting a compression function with as many segments as possible, which reduces the number of compression operations when the moving direction of an object remains constant for a quite period of time or changes smoothly. MH shows an opposite case, where SW-compression incurs less computation cost, as the object changes moving direction frequently. Both techniques demonstrate increasing computation cost when more clusters are formed inside the network, except for MH model which shows a reversed trend. The reason explained in Section 4.2.2 applies here. With RWP, cardioid and folium movement trajectories, the increasing number of clusters divide the trajectory which naturally can be compressed by one cluster head with one function into multiple segments, which are compressed by different cluster head with multiple compression functions, thus increasing the computation cost. However, MH trajectory in nature are segmented, thus increasing the number of clusters has limited impacts on DTTC's computation cost in MH model.

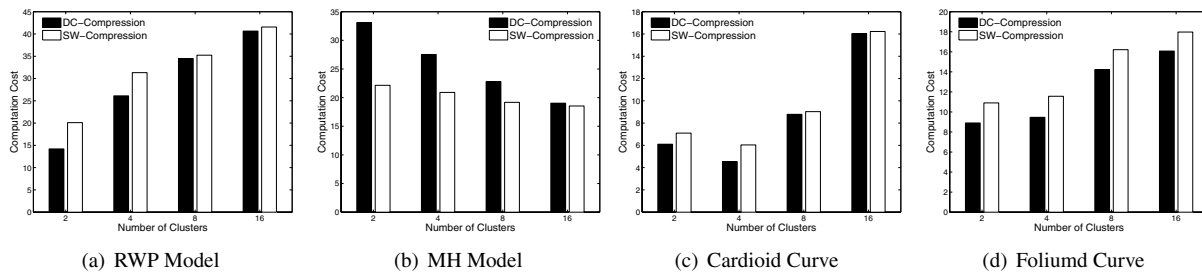


Figure 4. Computation Cost

## 5 Conclusion

By empowering a sink node with the ability of predicting object movements, the existing object tracking techniques aim at minimizing the communication cost. However, considering the complex object movement patterns, it is extremely difficult for existing prediction models to reach a prediction accuracy, at which the overall communication cost is reduced. Our study identifies the existence of delay-tolerant object tracking, and proposes a novel trajectory compression technique, called DTTC for delay-tolerant tracking in sensor networks. To the best of our knowledge, this is the first trajectory compression technique proposed for delay-tolerant object tracking sensor networks. In DTTC, each cluster head only report the compressed movement trajectory of an object, which provides the sink node traceable models about object movements, and also drastically reduces the total amount of data communications. Moreover, we propose two compression techniques, i.e., DC-compression and SW-compression, which favor two different movement patterns. The proposed segmentation scheme helps both compression techniques to compress the trajectory more accurately at less computation cost. An extensive performance evaluation is conducted to study the performance of DTTC and a prediction-based tracking technique, DPR [14]. The experimental results show that DTTC exhibits a superior performance in terms of distance error, communication cost and computation cost, and soundly outperforms DPR with various movement trajectories. Moreover, we draw the following two conclusions for employing DTTC tracking technique in delay-tolerant object tracking sensor networks,

- For moving objects, which move with frequent and dramatic changes in moving direction, employing more clusters helps to decrease the communication and computation cost.
- For moving objects, which move with a smoothly changing moving direction, less number of clusters are preferred to reduce the computation and communication cost. DC-compression is a better choice over SW-compression in terms of computation cost.

DTTC has been shown as a promising tracking technique for delay-tolerant object tracking for wireless sensor networks. As for future work, we are going to investigate the impact of cluster properties, including the cluster shape and the position of the cluster head on the performance of DTTC.

## 6 References

- [1] J. Aslam, Z. Butler, F. Constantin, V. Crespi, G. Cybenko, and D. Rus. Tracking a moving object with a binary sensor networks. In *Proceedings of ACM SenSys*, pages 150–161, 2003.
- [2] F. Bai and A. Helmy N. Sadagopan. The important framework for analyzing the impact of mobility on performance of routing for ad hoc networks. *AdHoc Networks Journal*, 1(4):383–403, 2003.
- [3] M. J. Coates. Distributed particle filtering for sensor networks. In *Proceedings of International Symposium on Information Processing in Sensor Networks*, 2004.
- [4] Q. Fang, F. Zhao, and L. Guibas. Lightweight sensing and communication protocols for target enumeration and aggregation. In *Proceedings of ACM MobiHoc*, pages 165 – 176, 2003.
- [5] S. Goel and T. Imielinski. Prediction-based monitoring in sensor networks: taking lessons from MPEG. *ACM Computer Communication Review*, 31(5), October 2001.
- [6] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *IEEE Proceedings of the Hawaii International Conference on System Sciences (HICSS)*, Maui, Hawaii, January 2000.
- [7] J. Hill, R. Szwedczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. *ACM SIGPLAN Notices*, 35(11):93–104, 2000.
- [8] G. Ing and M. J. Coates. Parallel particle filters for tracking in wireless sensor networks. In *Proceedings of Workshop on Signal Processing Advances in Wireless Communications*, Jun. 2005.
- [9] J. Liu, D. Petrovic, and F. Zhao. Multi-step information-directed sensor querying in distributed sensor networks. In *Proceedings of International Conference on Acoustics, Speech and Signal Processing*, Apr. 2003.
- [10] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a tiny aggregation service for ad-hoc sensor networks. *ACM SIGOPS Operating Systems Review*, 36(SI):131–146, 2002.
- [11] V. Manfredi, S. Mahadevan, J. F. Kurose, and V. Lesser. Switching kalman filters for prediction and tracking in adaptive meteorological sensing network. In *Proceedings of IEEE Conference on Sensor and Ad Hoc Communications and Networks*, 2005.
- [12] S. Patten, S. Poduri, and B. Krishnamachari. Energy-quality tradeoffs for target tracking in wireless sensor networks. In *Proceedings of International Workshop on Information Processing in Sensor Networks*, 2003.
- [13] Y. Xu and W.-C. Lee. On localized prediction for power efficient object tracking in sensor networks. In *In Proc. 1st International Workshop on Mobile Distributed Computing (MDC)*, pages 434–439, Providence, Rhode Island, May 2003.
- [14] Y. Xu, J. Winter, and W.-C. Lee. Dual prediction-based reporting mechanism for object tracking sensor networks. In *Proc. International Conference on Mobile and Ubiquitous Systems: Networking and Services*, pages 154–163, Boston, MA, Aug. 2004.
- [15] Y. Xu, J. Winter, and W.-C. Lee. Prediction-based strategies for energy saving in object tracking sensor networks. In *Proc. International Conference on Mobile Data Management*, pages 346–357, Berkeley, CA, Jan. 2004.
- [16] J. Yoon, M. Liu, and B. Noble. Sound mobility models. In *Proceedings of International Conference on Mobile Computing and Networking*, pages 205–216, San Diego, CA, Sept. 2003.
- [17] W. Zhang and G. Cao. Dctc: Dynamic convoy tree-based collaboration for target tracking in sensor networks. *IEEE Transactions on Wireless Communication*, 3(5):1689–1701, 2004.
- [18] F. Zhao, J. Shin, and J. Reich. Information-driven dynamic sensor collaboration for tracking applications. *IEEE Signal Processing Magazine*, 19(2):61–72, March 2002.