Remote Real-Time Trajectory Simplification

Ralph Lange Tobias Farrell Frank Dürr Kurt Rothermel
Institute of Parallel and Distributed Systems
Universität Stuttgart, Germany
Email: <*firstname.lastname*>@ipvs.uni-stuttgart.de

***Abstract*—Moving objects databases (MODs) have been proposed for managing trajectory data, an important kind of information for pervasive applications. To save storage capacity, a MOD generally stores simplified trajectories only. A simplified trajectory approximates the actual trajectory of the mobile object according to a certain accuracy bound.**

In order to minimize the costs of communicating position information between mobile object and MOD, the trajectory simplification should be performed by the mobile object. To assure that the MOD always has a valid simplified trajectory of the remote object, we propose the generic remote trajectory simplification protocol (GRTS) allowing for computing and managing a simplified trajectory in such a system in real-time.

We show how to combine GRTS with existing line simplification algorithms for computing the simplified trajectory and analyze trade-offs between the different algorithms. Our evaluations show that GRTS outperforms the two existing approaches by a factor of two and more in terms of reduction efficiency. Moreover, on average, the reduction efficiency of GRTS is only 12% worse compared to optimal offline simplification.

I. Introduction

Driven by the rapid advances in wireless communication and sensing technologies, context-awareness has become one of the most important characteristics of pervasive computing. Besides time and identity, the location of objects has been identified as primary context, cf. [1] and [2]. Many applications rely on real-time location information of a potentially large number of mobile objects. Application scenarios include asset tracking, traffic monitoring, emergency operations, as well as context-aware service provisioning.

Many of those applications not only require information on an object's current position but also on its locations in the past. In other words, they are interested in the object's trajectory, allowing them to retrieve the object's location at the current or some previous time. Also trajectories are used to determine the set of objects that were located in a certain area at a certain time. Consequently, maintaining trajectory information is a prerequisite for supporting the above mentioned primary contexts time, location, and identity.

Moving objects databases (MODs) have been proposed for managing trajectories of mobile objects, like people, vehicles, or containers. Mobile objects carry devices, like mobile phones, PDAs, or embedded systems, which are equipped with wireless communication capabilities and a positioning sensor, like a GPS receiver. Therefore, mobile objects can locally determine their location and inform MODs about their movements via wireless links.

Generally, a MOD represents an object's trajectory by a polyline in time and space where the vertices are the time-stamped positions acquired by the object's positioning sensor [3]–[10]. However, storing every sensed position as vertex of the trajectory causes high processing cost at the MOD and generally consumes too much storage capacity. For example, an ordinary GPS receiver may generate more than 30 million position data records per year, and applications might require millions of objects to be tracked. Therefore, data reduction by simplifying the trajectory is a crucial issue for MODs. This simplification aims at minimizing the number of the trajectory's vertices such that the simplified trajectory does not deviate by more than a certain accuracy bound from the actual one.

A straight-forward approach for trajectory simplification is to transfer the sensed position data from the tracked objects to the MOD and peform the simplification entirely on the MOD. However, this solution has an obvious drawback as also those data are transferred over the wireless communication network that are dropped later by simplification, which may cause a substantial waste of bandwidth. Therefore, a number of solutions have been proposed that instead do the simplification at the mobile object and only transfer the reduced data to the MOD. We will refer to this approach as Remote Trajectory Simplification (RTS) because simplification is performed remotely from the MOD's perspective.

The RTS approaches proposed in [7], [9], and [10] are based on dead reckoning, a technique originally designed for efficiently tracking the current location of mobile objects. With this technique, each tracked object initially transfers a function predicting its future movement to the MOD. This prediction function is updated at the MOD only if the object's locally sensed position deviates from the predicted one by more than some accuracy bound. Consequently, only those sensing operations that require an adjustment of the prediction cause an update message to be sent. This property has been exploited by the schemes in [7], [9], and [10] to perform trajectory simplification. They use the information included in update messages to build the simplified trajectory and add a new vertex only when an update arrives at the MOD. Consequently, the three schemes use dead reckoning for two different problems, object tracking and trajectory simplification. While this leads to simple solutions, the efficiency of simplification depends on the quality of dead reckoning, which has been designed for tracking rather than trajectory simplification. On the other hand, there exist a variety of efficient line simplification

978-1-4244-3304-9/09/$25.00 ©2009 IEEE

algorithms that could be used for that purpose.

In this paper, we propose an RTS scheme, called Generic Remote Trajectory Simplification (GRTS), which clearly separates tracking from simplification. GRTS also applies dead reckoning for tracking to minimize the messages to be sent over the wireless link. However, the scheme is generic in the sense that it can be combined with any line simplification algorithm suited for trajectories. For line simplification there exist different solutions, which vary in reduction efficiency and computational overhead. For example, an optimal line simplification algorithm provides the best reduction efficiency but causes the highest overhead, while solutions based on heuristics lower the computational overhead at the cost of reduced reduction rates. This flexibility allows applications to trade computational complexity off against reduction efficiency. Note that the latter not only influences the storage consumption of the trajectory data to be maintained at the MOD but also the amount of data to be communicated over the wireless link.

We investigate two variants of GRTS: With GRTS$^{\text{Opt}}$ the optimal line simplification algorithm introduced in [11] is applied on a trajectory segment whenever an update is sent, while GRTS$^{\text{Sec}}$ is based on an efficient simplification heuristic [4], which is processed after each sensing operation. Moreover, we optimize this heuristic to reduce the memory requirements at mobile objects.

Our evaluations show that GRTS$^{\text{Opt}}$ and GRTS$^{\text{Sec}}$ outperform all the existing RTS algorithms at least by a factor of two in terms of reduction efficiency. They also show that the number of vertices of the simplified trajectory obtained by GRTS$^{\text{Sec}}$ is only 12% greater than the number of vertices of the optimal trajectory simplification computed by an optimal offline algorithm.

To summarize, the contributions of this paper are as follows:

1) We propose a generic RTS scheme (GRTS) that can be combined with any suitable line simplification mechanism. To the best of our knowledge this the first RTS scheme that provides this flexibility.
2) We combine GRTS with an optimal [11] and heuristic [4] line simplification algorithm and show that both combinations increase the reduction efficiency by at least a factor of two compared to the existing RTS schemes, without increasing the message overhead.
3) We propose an optimization of the line simplification heuristic in [4] which reduces its space requirements by three-fourths on average.

The remainder of the paper is structured as follows: In Section II we describe our assumptions and the problem of remote trajectory simplification in detail, before we discuss related work in Section III. In Section IV we present GRTS and prove its correctness. Then, we present the two variants GRTS$^{\text{Opt}}$ and GRTS$^{\text{Sec}}$ and propose an optimization of the line simplification heuristic used in GRTS$^{\text{Sec}}$ in Section V. In Section VI we show the efficiency of GRTS$^{\text{Opt}}$ and GRTS$^{\text{Sec}}$ by comparing them to the existing RTS approaches as well as to well-known offline algorithms for line simplification and we discuss experiences with a prototypical implementation of GRTS$^{\text{Sec}}$. Finally, the paper is concluded in Section VII with a summary.

II. Assumptions and Problem Description

We consider a collection of mobile objects with embedded positioning sensors (e.g., GPS receivers) whose trajectories are managed by a remote MOD. The overall number of trajectories stored by the MOD is of no relevance here.

An object's movement over time describes a continuous spatiotemporal function $\vec{a} : \mathbb{R} \mapsto \mathbb{R}^d$ from time to plane ($d = 2$) or space ($d = 3$) called the object's *actual trajectory*. Let t_{C} denote the current time, then $\vec{a}(t)$ is defined up to t_{C} and $\vec{a}(t_{\text{C}})$ is the object's current actual position.

The positioning sensor periodically senses the object's current position with period T_{S}, resulting in a sequence of *sensed positions* $(s_1, s_2, \ldots, s_{\text{R}})$, where s_1 denotes the first and s_{R} the most recent sensed position. Each s_i is a data record consisting of the sensing time t and the sensed position $\vec{p}$.

The sensed positions define the *sensed trajectory* $\vec{s}(t)$, a continuous, piecewise linear function, as follows: Two consecutive positions s_i and s_{i+1} define a *spatiotemporal line section* $\overline{s_i\, s_{i+1}}$ on the domain $[s_i.t, s_{i+1}.t]$ as

$$\overline{s_i\, s_{i+1}} : t \mapsto \frac{(s_{i+1}.t - t)\, s_i.\vec{p} + (t - s_i.t)\, s_{i+1}.\vec{p}}{s_{i+1}.t - s_i.t} \ .$$

Then, $\vec{s}(t)$ is defined by the sequence $(s_1, s_2, \ldots, s_{\text{R}})$ on the domain $[s_1.t, s_{\text{R}}.t]$ as

$$\vec{s} : t \mapsto \overline{s_i\, s_{i+1}}(t) \text{ where } s_i.t \leq t \leq s_{i+1}.t \ .$$

Geometrically, $\vec{s}(t)$ is a time-monotonous polyline in $\mathbb{R}^{1+d}$ given by the sequence of vertices $(s_1, s_2, \ldots, s_{\text{R}})$.

Note that the domain $[s_1.t, s_{\text{R}}.t]$ does not continuously increase over time but periodically by T_{S} with each sensing operation. For current time t_{C}, it holds $t_{\text{C}} - T_{\text{S}} < s_{\text{R}}.t \leq t_{\text{C}}$.

$\vec{s}(t)$ generally deviates from $\vec{a}(t)$ due to inaccuracies of the positioning sensor and the time-discrete sensing. In the following, we assume that this deviation is bound by a certain *maximum sensing deviation* δ, i.e. $\forall\, t' \in [s_1.t, s_{\text{R}}.t]$ it holds $|\vec{s}(t') - \vec{a}(t')| \leq \delta$. See [12] for a discussion how to determine the possible actual movement between two sensed positions by means of physical constraints like the maximum velocity or acceleration. In general, δ may be a statistical value only, which holds with high probability. Deviations beyond δ are considered as sensing errors – not in scope of this paper.

The MOD describes the object's trajectory by a continuous, piecewise linear function $\vec{u} : t \mapsto \mathbb{R}^d$ called *simplified trajectory*. Geometrically, $\vec{u}(t)$ is a time-monotonous polyline in $\mathbb{R}^{1+d}$ given by a sequence of vertices $(u_1, u_2, \ldots, u_m)$ like $\vec{s}(t)$. Each vertex u_j is a data record with attributes t and $\vec{p}$, just as a sensed position.

We refer to any clipping of $\vec{a}(t)$, $\vec{s}(t)$, or $\vec{u}(t)$ given by an arbitrary time interval or a subsequence of vertices as *trajectory segment*.

The algorithmic problem of remote trajectory simplification is to minimize the number of vertices m of the simplified

trajectory $\vec{u}(t)$ under the following two constraints, where t_C denotes the current time:

1) *Simplification constraint:* For a certain *accuracy bound* ϵ known by the mobile object and the MOD, it holds

$$\forall\, t \in [s_1.t, t_C] \,:\, |\vec{u}(t) - \vec{a}(t)| \leq \epsilon \ .$$

2) *Real-time constraint:* At t_C, position $\vec{u}(t)$ is available at the MOD for each $t \in [s_1.t, t_C]$.

The simplification constraint corresponds to the well-known line simplification problem, i.e. given a polyline and a certain accuracy bound, determining another polyline, that approximates the given one according to this bound with a minimal number of vertices. As mentioned above, for line simplification there exist various solutions, which vary in reduction efficiency and computational overhead.

The real-time constraint requires both the current and past position data to be available at the MOD in time. Therefore, a tracking mechanism is needed that efficiently transfers the current and past position data to the MOD. Of course, tracking and simplification need to be synchronized to ensure that the data is reduced on the mobile object so that it arrives in time at the MOD. We will see later that different simplification algorithms might need different types of synchronization.

The goal is to develop an RTS scheme that meets the both constraints stated above. In addition, it should have the following properties:

1) *Flexibility:* Due to a clear separation of tracking and simplification concerns it should be possible to combine the RTS scheme with a variety of line simplification algorithms to be able to trade-off between complexity and efficiency of simplification.
2) *High reduction efficiency:* The reduction efficiency impacts both the storage capacity needed at the MOD as well as the amount of data transferred over the wireless network. Hence, it should be possible to achieve high reduction rates for a reasonable computational overhead.
3) *Low communication overhead:* The amount of data exchanged between mobile objects and the MOD depends on both simplification and tracking. This communication should be minimized due to bandwidth limitations of wireless networks and energy constraints of mobile devices.
4) *Low space requirements:* Since memory on mobile devices is often a scarce resource, the space demand of the scheme is critical.

III. Related Work

In this section, we briefly discuss existing work on line simplification and position tracking as foundation of our work. Then, we address existing approaches for remote trajectory simplification, which support both tracking and simplification.

Line simplification refers to a multitude of algorithmic problems on approximating a given polyline by a simplified one with fewer vertices [11], [13]. Here, we always refer to the min-# problem defined as minimizing the number of vertices of the simplified polyline according to a given accuracy bound. Further, to limit the computational complexity, we only consider strong simplification, where the vertices of the simplified polyline are a subset of the vertices of the original polyline.

The Douglas-Peucker algorithm [14] probably is the best-known heuristic for line simplification and has been also proposed for trajectories [5], [8]. Although it is an offline algorithm, our online approach GRTS achieves better reduction rates.

In [11], Imai and Iri give the first optimal algorithm for line simplification. They reduce the simplification problem to computing a shortest path between two nodes in a directed acyclic graph. We combine our generic approach for remote trajectory simplification with this algorithm, cf. Section V-A. Furthermore, we use it as reference in our evaluations.

Meratnia and de By propose the Opening-Window algorithm for trajectory simplification [4]. Variants of this online algorithm have been also proposed in [13] and [15] with different names. We refer to this algorithm as *section heuristic* and combine our approach GRTS with it, cf. Section V-B.

Threshold-guided Sampling is an online heuristic for trajectory simplification [6]. However, its condition for adding a new vertex to the simplified trajectory generally does not limit the deviation from the actual trajectory.

In [15], a mechanism for preprocessing position data of mobile objects is presented. The component aims at reducing the position data to be stored by a database according to a given accuracy bound. The authors propose five different reduction algorithms, where only one – the above-mentioned section heuristic – yields a connected simplified trajectory.

None of the above works considers the remote trajectory simplification problem.

The most efficient tracking protocols are based on dead reckoning [16]–[18]. Using dead reckoning, the object initially transmits its current position and a prediction on its future movement to the MOD. While the object's actual position and the predicted one do not deviate by more than a certain accuracy bound, no update message is required. Otherwise, if the object impends to reach the accuracy bound, it determines a new prediction using the last sensed positions and transmits it to the MOD. The most simple but nevertheless efficient variant is linear dead reckoning (LDR) [16], [17]. It uses a linear prediction given by a timestamped position and a velocity vector. Dead reckoning does not perform trajectory simplification since it describes the object's movement by a discontinuous function in time.

In [9], Tiešytė and Jensen present an approach for remote trajectory simplification based on LDR. They propose an algorithm for computing a connected trajectory on the basis of the linear predictions which approximates the actual trajectory according to the same accuracy bound used with LDR. However, their findings only apply to pre-known routes like bus lines, i.e movement in $\mathbb{R}^1$.

Now, we address the two existing RTS approaches for arbitrary movement in $\mathbb{R}^2$ or $\mathbb{R}^3$. In [7], Trajcevski et al.

prove that the simplified trajectory given by the origins of the linear predictions of LDR with accuracy bound ϵ approximates the actual trajectory by 2ϵ [7]. Correspondingly, LDR with $\frac{1}{2}\epsilon$ ($\text{LDR}_{\frac{1}{2}}$) allows for remote trajectory simplification with accuracy bound ϵ.

In [10], we propose Connection-preserving Dead Reckoning (CDR) which outperforms $\text{LDR}_{\frac{1}{2}}$. CDR exploits the observation, that the simplified trajectory given by the prediction origins of LDR with bound ϵ approximates the actual trajectory according to ϵ most of time. Therefore, CDR is based on LDR with accuracy bound ϵ using an additional update condition for LDR which guarantees the desired accuracy bound.

In both schemes simplification is solely based on LDR. GRTS proposed in this paper clearly separates tracking from simplification and outperforms CDR at least by a factor of two and $\text{LDR}_{\frac{1}{2}}$ by a factor of three regarding reduction efficiency.

IV. Generic Remote Trajectory Simplification

In this section, we present our Generic Remote Trajectory Simplification protocol (GRTS) and prove its correctness.

A. GRTS Protocol

As motivated above, it is a good idea to separate tracking from simplification issues as far as possible to gain flexibility. However, the simplification process must be synchronized with tracking to make sure that the simplified data arrives in time at the MOD. The GRTS protocol proposed in this section follows a synchronization pattern, which we call *per-update simplification*.

With this pattern, simplification is performed whenever the tracking mechanism decides to send an update message. For that purpose the mobile object stores a partial history of sensed positions which serves as input for the simplification process. Based on this input, the simplification algorithm generates a sequence of vertices of the simplified trajectory, which then is included in the update message. In most cases, the generated sequence only includes one vertex or is even empty. Therefore, GRTS has a better reduction efficiency than the RTS approaches $\text{LDR}_{\frac{1}{2}}$ [7] and CDR [10], which always generate one vertex per update. The advantage of per-update simplification is that it can be combined with both online and offline line simplification algorithms. In Section V-B we will describe a per-sense simplification pattern that optimizes GRTS for online algorithms.

Now we will describe GRTS in more detail. GRTS uses linear dead reckoning (LDR) for position tracking since this is the most efficient, general applicable position tracking protocol [16]–[18]. With LDR, the MOD has a linear prediction function $\vec{l}(t)$ for determining the object's current position. $\vec{l}(t)$ is defined by a sensed position l_O called *prediction origin* and a vector $\vec{l}_\text{V}$ called *velocity vector* as

$$\vec{l}(t) : t \mapsto l_\text{O}.\vec{p} + (t - l_\text{O}.t)\vec{l}_\text{V} \ .$$

For a given accuracy bound ϵ, the LDR protocol guarantees, that $\vec{l}(t)$ known by the MOD approximates the objects' current actual position by ϵ. Formally, at current time t_C, it guarantees $|\vec{l}(t_\text{C}) - \vec{a}(t_\text{C})| \leq \epsilon$. For this purpose, LDR has to send a new prediction to the MOD as soon as $|\vec{l}(t_\text{C}) - \vec{a}(t_\text{C})|$ *impends* to reach ϵ, taking into account the inaccuracy of the positioning sensor, the possible movement within the sensing period T_S, and the time for transmitting an update message (for details see [17] and [12]).

The mobile object stores the current prediction and the *sensing history* $\mathbb{S}$, which includes all sensed positions since the most recent vertex u_m of the simplified trajectory reported to the MOD. More precisely, the sensing history is the sequence of chronologically ordered sensed positions $\mathbb{S} := (s_i : s_i.t \geq u_m.t)$, with $\text{first}(\mathbb{S}) = u_m$ and $\text{last}(\mathbb{S}) = s_\text{R}$, respectively. Once LDR causes a new prediction to be sent to the MOD, the simplification algorithm takes the stored sensing history as input and provides the sequence of vertices $\mathbb{U} := (u_{m+1}, \ldots u_{m+k})$ to be appended to the simplified trajectory managed by the MOD, where $k = |\mathbb{U}|$ mostly equals 0 or 1.

The MOD stores the simplified trajectory $\vec{u}(t)$, which consists of the following of three parts:

- The spatiotemporal polyline given by the vertices $(u_1, \ldots, u_m)$ composes the first part.
- The current prediction, given by the prediction origin l_O and the velocity vector $\vec{l}_\text{V}$, composes the third part.
- The line section $\overline{u_m\, l_\text{O}}$ in-between u_m and l_O composes the second part.

Figure 1 illustrates those three parts. While the second and third part change with each update message, the first one is created in an append-only fashion.

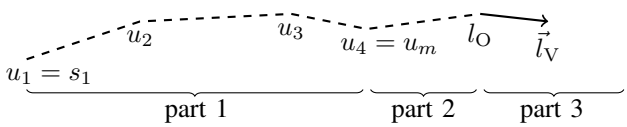

Fig. 1. Three parts of $\vec{u}(t)$, managed by the MOD.

Figure 2 shows the pseudo code of the generic GRTS algorithm executed by the mobile object. For the sake of simplicity, we assume that once the tracking has been started, the mobile object will be tracked forever. Extensions to switch tracking on and off are straightforward.

Initially, the mobile object transmits its most recent sensed position $s_\text{R} = s_1$ as first vertex u_1 to the MOD, together with a prediction with origin $l_\text{O} = s_\text{R}$ (line 4). Then, it executes an infinite loop (lines 7 to 20).

Within each iteration it first senses its current position (line 8) and appends it to the sensing history (line 9). Then, it checks whether it has to send an update message to the MOD according to LDR (line 10).

The update message not only has to contain a new prediction for the third part of $\vec{u}(t)$ but also the vertices to add to the first part, which together also updates the second part. Therefore, the object computes a simplified trajectory segment for the movement between the last vertex u_m known to the MOD and the new prediction origin $l_\text{O} = s_\text{R}$. As the sensed trajectory

```
1:  s_R ← sense position                          ▷ Most recent sensed position.
2:  𝕌 ← (s_R)                                     ▷ New vertices to transmit to MOD.
3:  (l_O, l⃗_V) ← (s_R, 0)                         ▷ Current prediction.
4:  send update message (l_O, l⃗_V, 𝕌) to MOD
5:  𝕊 ← (s_R)                                     ▷ Sensing history.
6:  𝕌 ← ()
7:  while true do
8:      s_R ← sense position
9:      𝕊 ← 𝕊 ‖ (s_R)                             ▷ Append s_R to sensing history.
10:     if LDR causes update then
11:         𝕌′ ← line simplification with bound μ on 𝕊
12:         𝕌 ← 𝕌′ \ (first(𝕌′), last(𝕌′))
13:         (l_O, l⃗_V) ← compute new prediction (s_R, ...)
14:         send update message (l_O, l⃗_V, 𝕌) to MOD
15:         if |𝕌| > 0 then
16:             𝕊 ← (s_i ∈ 𝕊 : s_i.t ≥ last(𝕌).t)
17:             𝕌 ← ()
18:         end if
19:     end if
20: end while
```

Fig. 2. GRTS with per-update simplification executed by the mobile object.

segment given by $\mathbb{S}$ deviates from the actual trajectory $\vec{a}(t)$ not more than the maximum sensing deviation δ, it executes the line simplification algorithm with the *simplification bound* $\mu = \epsilon - \delta$ and stores the resulting vertices of the simplified segment in $\mathbb{U}'$ (line 11). Therefore, the simplified segment given by $\mathbb{U}'$ approximates $\vec{a}(t)$ on the respective domain $[u_m.t, s_\mathrm{R}.t]$ according to the accuracy bound ϵ.

For $\mathbb{U}'$, it holds $\mathrm{first}(\mathbb{U}') = \mathrm{first}(\mathbb{S}) = u_m$, which is already known to the MOD, and $\mathrm{last}(\mathbb{U}') = \mathrm{last}(\mathbb{S}) = s_\mathrm{R}$, which is going to be the new prediction origin. Therefore, these two vertices are removed from $\mathbb{U}'$ resulting in $\mathbb{U}$, which may be empty (line 12).[1] Next, a new prediction is determined and transmitted to the MOD together with $\mathbb{U}$. Finally, if $\mathbb{U}$ is not empty, the sensed positions before the new last vertex u_m stored by the MOD are removed from $\mathbb{S}$ (line 16) since the respective segment now is approximated by the first part of $\vec{u}(t)$.

The algorithm executed by the MOD is rather simple. On receiving a message $(l_\mathrm{O}, \vec{l}_\mathrm{V}, \mathbb{U})$, it appends the k vertices given in $\mathbb{U}$ to $(u_1, \ldots, u_m)$ as $(u_{m+1}, \ldots, u_{m+k})$, sets m to $m+k$, and replaces the current prediction with the new one.

Now, let us see how the MOD can use the stored information to find out about an object's position at time t':

1) $t' \leq u_m.t$ *:* The MOD calculates $\vec{u}(t')$ by linear interpolation between the vertices u_j and u_{j+1} with $u_j.t \leq t' \leq u_{j+1}.t$ as described in Section II.
2) $u_m.t \leq t' \leq l_\mathrm{O}.t$ *:* The MOD calculates $\vec{u}(t')$ by linear interpolation between u_m and l_O.
3) $t' \geq l_\mathrm{O}.t$ *:* The MOD calculates $\vec{u}(t')$ by means of the prediction function $\vec{l}(t)$ given by l_O and $\vec{l}_\mathrm{V}$.

[1] Actually, $\mathbb{U}'$ and $\mathbb{U}$ can be one and the same data structure. $\mathbb{U}'$ is introduced for readability only. It always holds that $\mathbb{U}' = (u_m) \,\|\, \mathbb{U} \,\|\, (s_\mathrm{R})$.

B. Correctness of GRTS

To show the correctness of GRTS, we prove that GRTS satisfies the simplification and the real-time constraint, cf. Section II. Both constraints are to be fulfilled within the time interval $[s_1.t, t_\mathrm{C}]$, where t_C denotes the current time.

Clearly, the simplified trajectory $\vec{u}(t)$ always is defined on $[s_1.t, t_\mathrm{C}]$ since the domain of $\vec{u}(t)$ actually is $[s_1.t, \infty]$.

However, to show that GRTS satisfies the simplification constraint and real-time constraint, we now prove, that for every $t' \in [s_1.t, t_\mathrm{C}]$ the MOD knows $\vec{u}(t')$ with $|\vec{u}(t) - \vec{a}(t)| \leq \epsilon$. For that purpose, we consider the three parts of $\vec{u}(t)$ illustrated in Figure 1:

- *Part 1:* It is $s_1.t \leq t' \leq u_m.t$. Let s_i and s_{i+1} be the sensed positions that enclose t', i.e. $s_i.t \leq t' \leq s_{i+1}.t$. According to line simplification (line 11), it holds that $|\overline{s_i\, s_{i+1}}(t') - \vec{u}(t')| \leq \mu$. Using the maximum sensing deviation δ defined in Section II we conclude the triangle inequality $|\vec{u}(t') - \vec{a}(t')| \leq |\vec{u}(t') - \overline{s_i\, s_{i+1}}(t')| + |\overline{s_i\, s_{i+1}}(t') - \vec{a}(t')| \leq \mu + \delta = \epsilon$.
- *Part 2:* It is $u_m.t < t' < l_\mathrm{O}.t$. At the time of the most recent execution of the line simplification algorithm the current prediction origin l_O was equal to $\mathrm{last}(\mathbb{S})$. Therefore, u_m and l_O were the last two vertices in $\mathbb{U}'$. If $|\mathbb{U}| = 0$, they even were the only two vertices in $\mathbb{U}'$. In any case, the line section $\overline{u_m\, l_\mathrm{O}}$ is a simplification of the segment given by the sensed positions within the time interval $[u_m.t, l_\mathrm{O}.t]$ according to the bound μ. Analogous to part 1, we conclude $|\overline{u_m\, l_\mathrm{O}}(t') - \vec{a}(t')| \leq \epsilon$ using triangle inequality.
- *Part 3:* It is $l_\mathrm{O}.t \leq t' \leq t_\mathrm{C}$. As explained above, the LDR protocol guarantees $|\vec{l}(t_\mathrm{C}) - \vec{a}(t_\mathrm{C})| \leq \epsilon$ for $\vec{l}(t)$ given by l_O and $\vec{l}_\mathrm{V}$ known to the MOD. As the MOD has not received a new prediction up to current time t_C, we conclude that $\forall\, t' \in [l_\mathrm{O}.t, t_\mathrm{C}] : |\vec{l}(t') - \vec{a}(t')| \leq \epsilon$. □

V. Combining GRTS with Line Simplification Algorithms

In this section, we present two combinations of GRTS with different line simplification algorithms. First, we describe how to combine GRTS with the optimal offline line simplification algorithm by Imai and Iri [11]. Then, we present combining GRTS with the section heuristic [4]. Finally, we propose a novel optimization of the section heuristic.

A. GRTS with Optimal Line Simplification

Here, we describe GRTS$^{\mathrm{Opt}}$ which combines GRTS with the optimal simplification algorithm introduced in [11]. Although this algorithm has originally been designed for offline usage, we apply it online following the per-update simplification pattern. That is, whenever LDR decides to send a new update, the algorithm is initiated with input $\mathbb{S}$. It computes the (possibly empty) set of new vertices $\mathbb{U}$, which then is included in the update message. Consequently, LDR divides the sensed trajectory $\vec{s}(t)$ into a set of segments, which are simplified independently from each other.

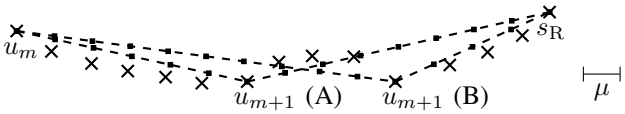

Fig. 3. Two possible simplified trajectories (A) and (B) with minimal number of vertices $\mathbb{U}' = (u_m, u_{m+1}, s_\mathrm{R})$ for the sensed positions illustrated by small crosses. Here, (B) is the better choice.

In detail, the algorithm first considers the sensed positions in history $\mathbb{S}$ as vertices of an unweighted, directed graph and adds an edge for each pair of sensed positions (s_i, s_{i+x}), where the line section $\overline{s_i\, s_{i+x}}$ approximates the sensed positions $(s_i, \ldots, s_{i+x})$ by the simplification bound μ. This particularly applies to every pair (s_i, s_{i+1}). Second, it computes a shortest path between the first vertex $\mathrm{first}(\mathbb{S}) = u_m$ and the last vertex $\mathrm{last}(\mathbb{S}) = s_\mathrm{R}$. The vertices $\mathbb{U}'$ of the shortest path compose a simplified trajectory which approximates $\mathbb{S}$ by μ and thus $\vec{a}(t)$ within the time interval $[u_m.t, s_\mathrm{R}.t]$ by the accuracy bound ϵ.

Due to the segment-wise simplification, GRTS$^{\mathrm{Opt}}$ generally does not achieve the optimal, best possible reduction rate as it would be achieved with the optimal line simplification algorithm being applied offline to the overall sequence of sensed positions. The fundamental reason for that is, that there may exist several possible simplifications with a minimum number of vertices $\mathbb{U}'$ if GRTS$^{\mathrm{Opt}}$ simplifies a segment given by $\mathbb{S}$. Figure 3 gives an example of two possible sequences of vertices $\mathbb{U}' = (u_m, u_{m+1}, s_\mathrm{R})$ implying two possible sequences of vertices $\mathbb{U} = (u_{m+1})$ to be sent to the MOD. Generally, choosing $\mathbb{U}$ with maximum $\mathrm{last}(\mathbb{U}).t$ – here (B) – is a good heuristic as it minimizes the size of $\mathbb{S}$ in the subsequent execution of the line simplification algorithm. Nevertheless, there may be also cases, where choosing another $\mathbb{U}$ would yield a better overall reduction efficiency.

It is important to notice that the segmentation itself is determined by LDR. Consequently, the reduction efficiency is still influenced by LDR, however to a much lower degree than with the existing approaches LDR$_{\frac{1}{2}}$ [7] and CDR [10].

B. GRTS with Section Heuristic

The section heuristic is a simple online line simplification algorithm which has been proposed in [4][2], [13], and [15].

For simplifying a sequence of sensed positions $(s_1, s_2, \ldots)$ by bound μ, the section heuristic works as follows: First, it sets s_1 as vertex u_1 of the simplified trajectory. Then, it iteratively probes the line sections $\overline{s_1\, s_2}, \overline{s_1\, s_3}, \ldots$ until it finds the first section $\overline{s_1\, s_x}$ that would violate μ, i.e. where $\exists\, s_i \in (s_1, \ldots, s_x) : |s_i.\vec{p} - \overline{s_1\, s_x}(s_i.t)| > \mu$. In this case, the section heuristic adds the previous line section $\overline{s_1\, s_{x-1}}$ to the simplified trajectory by storing $u_2 := s_{x-1}$. Next, it repeats the above procedure starting at s_{x-1}, and so on.

Since this online algorithm processes the sensed positions iteratively, it allows for *per-sense simplification*. Figure 4 shows the corresponding pseudocode of the resulting combination GRTS$^{\mathrm{Sec}}$. For each sensed position s_R, GRTS$^{\mathrm{Sec}}$ checks whether the line section $\overline{\mathrm{first}(\mathbb{S})\, s_\mathrm{R}}$ approximates the sensed positions in-between by simplification bound μ or not (line 9). If not, then it appends the last sensed position – the one before s_R – to $\mathbb{U}$ (line 10) and reduces the sensing history accordingly (line 11). Thus, in contrast to the general GRTS algorithm, the sensing history only comprises the sensed positions between the last vertex u_{m+k} known by the mobile object and s_R. When LDR causes a new update message to be sent, GRTS$^{\mathrm{Sec}}$ simply includes the (possibly empty) set of vertices $\mathbb{U}$ into the message and then resets $\mathbb{U}$ to the empty sequence.

The advantage of per-sense simplification is that simplification is performed as early as possible, resulting in a smaller sensing history $\mathbb{S}$ on average. Moreover, the computing time for line simplification is distributed over all iterations of GRTS.

C. Optimization of the Section Heuristic

The average size of the sensing history $|\mathbb{S}|$ can be further reduced by a novel optimization of the section heuristic. The basic idea of this optimization is the following: Each sensed position $s_i \in \mathbb{S}$ poses a constraint on the next line section $\overline{u_m\, u_{m+1}}$ that is going to approximate $\mathbb{S}$. If the constraint given by another sensed position s_{i+x} completely encloses the one given by s_i, then s_i can be removed from $\mathbb{S}$ without affecting the simplification. This reduces the space consumption of the section heuristic by three-fourths on average, cf. Section VI.

The constraint defined by a $s_i \in \mathbb{S}$ requires the distance $|s_i.\vec{p} - \overline{u_m\, u_{m+1}}(s_i.t)|$ to not exceed μ. The GRTS$^{\mathrm{Sec}}$ algorithm checks this constraint for every potential line section $\overline{u_m\, s_\mathrm{R}} = \overline{\mathrm{first}(\mathbb{S})\, s_\mathrm{R}}$ (line 9). Geometrically, for each s_i, the line section has to pass the circle with center $s_i.\vec{p}$ and radius μ at time $s_i.t$ as illustrated in Figure 5. As the line section's

```
 1: s_R ← sense position                    ▷ Most recent sensed position.
 2: 𝕌 ← (s_R)                               ▷ New vertices to transmit to MOD.
 3: (l_O, l⃗_V) ← (s_R, 0)                   ▷ Current prediction.
 4: send update message (l_O, l⃗_V, 𝕌) to MOD
 5: 𝕊 ← (s_R)                               ▷ Sensing history.
 6: 𝕌 ← ()
 7: while true do
 8:     s_R ← sense position
 9:     if ∃ s_i ∈ 𝕊 : |s_i.p⃗ − first(𝕊) s_R(s_i.t)| > ε then
10:         𝕌 ← 𝕌 ‖ (last(𝕊))              ▷ Append last sensed position.
11:         𝕊 ← (last(𝕊))
12:     end if
13:     𝕊 ← 𝕊 ‖ (s_R)                      ▷ Append s_R to sensing history.
14:     if LDR causes update then
15:         (l_O, l⃗_V) ← compute new prediction (s_R, ...)
16:         send update message (l_O, l⃗_V, 𝕌) to MOD
17:         𝕌 ← ()
18:     end if
19: end while
```

Fig. 4. GRTS$^{\mathrm{Sec}}$ algorithm with per-sense simplification.

[2]The authors of [4] refer to the section heuristic as Opening-Window algorithm (OPW) and distinguish two variants with different distance metrics. The one with the better reduction efficiency which corresponds to the section heuristic as explained here is called BOPW-TR.

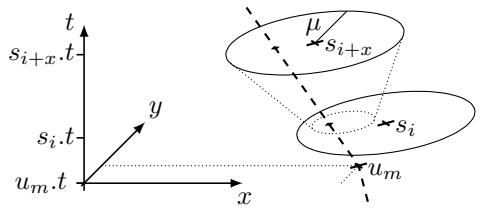

Fig. 5. The line section starting at u_m has to pass the circles at s_i and s_{i+x}. Here, the constraint by s_{i+x} encloses the one by s_i.

first vertex is known, the circles of two sensed positions s_i and s_{i+x} can be normalized regarding time and compared with each other: The circle of s_{i+x} poses the same constraint like the circle with center $\overline{\text{first}(\mathbb{S})\, s_{i+x}}(s_i.t)$ and radius

$$\mu \, \frac{s_i.t - \text{first}(\mathbb{S}).t}{s_{i+x}.t - \text{first}(\mathbb{S}).t}$$

at time $s_i.t$. Now, if this circle is contained by the circle of s_i as pictured in Figure 5, then s_i can be removed from $\mathbb{S}$ accordingly. Thus, for each sensed position s_R, GRTS$^\text{Sec}$ can remove every s_i from $\mathbb{S}$ whose circle contains the normalized circle of s_R at $s_i.t$ – except $s_i = \text{first}(\mathbb{S})$. In Figure 4 this removal should be included between line 12 and 13.

VI. Evaluation

We evaluated GRTS in two ways: For significant results on its performance, we simulated and analyzed GRTS$^\text{Sec}$ and GRTS$^\text{Opt}$ with hundreds of real trajectories and compared it to the existing RTS approaches as well as to offline simplification. For practical experiences, we conducted experiments with a prototypical implementation of GRTS$^\text{Sec}$ and an in-memory MOD allowing for tracking trajectories of multiple mobile objects in Google-Earth in real-time.

In the following, we first discuss the simulation-based analysis followed by the experiences with the prototype.

A. Simulation-based Analysis

For analyzing the performance of GRTS, we implemented a simulation software for GRTS$^\text{Sec}$ and GRTS$^\text{Opt}$, the existing RTS approaches LDR$_\frac{1}{2}$ [7] and CDR [10], as well as the optimal line simplification algorithm (Ref$^\text{Opt}$) by Imai and Iri [11] and the Douglas-Peucker algorithm (Ref$^\text{DP}$) [14] in the C programming language. We selected Ref$^\text{Opt}$ as a reference for comparing our results to the best possible reduction rate, while Ref$^\text{DP}$ is a commonly used offline heuristic.

For simulating these algorithms with realistic data, we downloaded hundreds of GPS trajectories (GPS traces) each containing more than 1000 recorded positions from the OpenStreetMap website [19]. In several processing steps, we filtered those trajectories, that provide distinct position fixes for each second – i.e. that have not undergone any previous data reduction – and that could be clearly classified according to their means of transportation into foot, bicycle, and motor vehicle. For classifying a trajectory, we not only relied upon its velocity characteristics but also on representative tags specified on the OpenStreetMap website.

Then, we simulated the execution of LDR$_\frac{1}{2}$, CDR, GRTS$^\text{Sec}$, and GRTS$^\text{Opt}$ by sequentially feeding the algorithms with the recorded positions given in the GPS trajectories. For each algorithm, we measured the number of vertices of the resulting simplified trajectories, the number of update messages, and the amount of transmitted data, depending on ϵ varying from 50 to $500\,\text{m}$. Further, we measured the space requirements and the computational effort for each algorithm. In accordance with the GPS trajectories, we used a sensing period of $T_\text{S} = 1\,\text{s}$ and a maximum sensing deviation of $\delta = 20\,\text{m}$ in our simulations. The latter value takes into account a GPS inaccuracy of up to $10\,\text{m}$ and a maximum movement deviation of $10\,\text{m}$ from the line section between two sensing operations, given by $\frac{1}{2} T_\text{S} v_\text{max}$ with $v_\text{max} = 20\,\text{m/s}$ as explained in [12].

Also, we applied the offline algorithms Ref$^\text{Opt}$ and Ref$^\text{DP}$ with bound $\mu = \epsilon - \delta$ to the entire trajectories and measured the number of vertices of the resulting simplified trajectories.

All experiments were performed on an AMD Opteron Linux Server with $2.8\,\text{GHz}$ and $4\,\text{GB}$ RAM.

The different velocities of the means of transportation do not yield any significant differences when comparing the simplification approaches with each other, but only when considering the absolute values for reduction efficiency and communication. Therefore, we give the average results of the 3×100 largest trajectories of the three means of transportation and refer to the individual means of transportation where necessary. Each of the 300 trajectories comprises 1400 to 16500 GPS positions, i.e. spans about $20\,\text{min}$ to $5\,\text{h}$.

Next, we give the results on data reduction, followed by the results on communication and computational cost.

1) Reduction Efficiency: The reduction efficiency of trajectory simplification is measured by the *reduction rate* defined as the number of sensed positions divided by the number of vertices of the simplified trajectory $\vec{u}(t)$, i.e. $|(s_1, \ldots, s_n)| \,/\, |(u_1, \ldots, u_m)|$.

Figure 6 shows the reduction rates of the RTS algorithms and the two reference algorithms Ref$^\text{Opt}$ and Ref$^\text{DP}$. As expected, the reduction rates increase with increasing ϵ.

Both combinations of GRTS outperform the existing RTS approaches by a factor of two and more. More precisely, on average, the reduction rate of GRTS$^\text{Opt}$ is 2.9 times greater

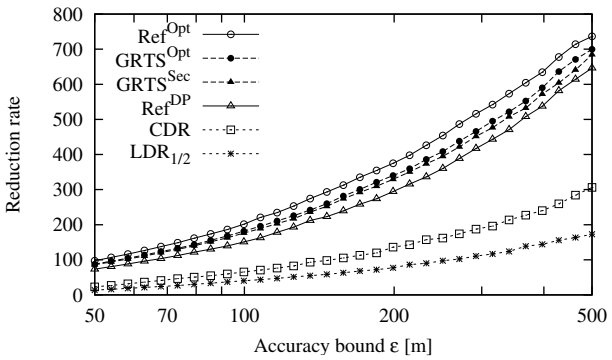

Fig. 6. Reduction rate depending on ϵ.

than the rate of CDR and 5.2 times greater than the rate of $\text{LDR}_{\frac{1}{2}}$. GRTS^{Sec} outperforms CDR by 2.8 and $\text{LDR}_{\frac{1}{2}}$ by 5.1 on average. The reason is, that line simplification with GRTS is largely or completely independent of LDR as explained in Section V, while CDR and $\text{LDR}_{\frac{1}{2}}$ add a new vertex to $\vec{u}(t)$ with each update message. For example, for $\epsilon = 100\,\text{m}$ CDR sends about 84 update messages within one hour and hence generates a simplified trajectory comprising 84 vertices for that span of time. GRTS^{Opt} and GRTS^{Sec} likewise send 84 update messages but only generate about 35 vertices, respectively.

Both combinations of GRTS achieve similar reduction rates. Nevertheless, the reduction rate of GRTS^{Opt} is always slightly greater or equal to the rate of GRTS^{Sec}. More precisely, GRTS^{Opt} outperforms GRTS^{Sec} by 3% on average.

We can see that both GRTS^{Opt} and GRTS^{Sec} always outperform Ref^{DP}. With GRTS^{Sec} the reduction rate is 15% greater than with Ref^{DP}. With GRTS^{Opt} it even is 19% greater than with the Douglas-Peucker heuristic. This is a surprising result given the fact, that Ref^{DP} is performed offline on the entire GPS trajectories. Moreover, on average, the reduction rate of GRTS^{Sec} is only 12% worse than the best possible reduction rate by the optimal algorithm Ref^{Opt}. With GRTS^{Opt} it is 9% worse than the reduction rate of Ref^{Opt} due to the segmentation by LDR.

All these values similarly hold for the individual means of transportation. For example, with motor vehicles GRTS^{Opt} and GRTS^{Sec} outperform CDR by a factor of 3.0 and 2.9 respectively, and by foot they outperform CDR by a factor of 3.1 and 3.0. However, the absolute reduction rates depend on the mean of transportation due to the different ratio between the corresponding velocity and ϵ. For instance, for $\epsilon = 100\,\text{m}$ the reduction rate of GRTS^{Opt} is 62.3 for motor vehicles, 128.5 for the bicycle, and 330.6 for walking by foot.

Figure 7 renders these differences more precisely by showing the reduction rate depending on the velocity. For that purpose we grouped the GPS trajectories by their average velocities and then computed the average reduction rate for each group and simplification algorithm for $\epsilon = 100\,\text{m}$.

Clearly, the reduction rate of each algorithm decreases with increasing velocity. For example, with GRTS^{Opt} and GRTS^{Sec} it is about 360 for an average velocity of $1.2\,\text{m/s}$, but only about

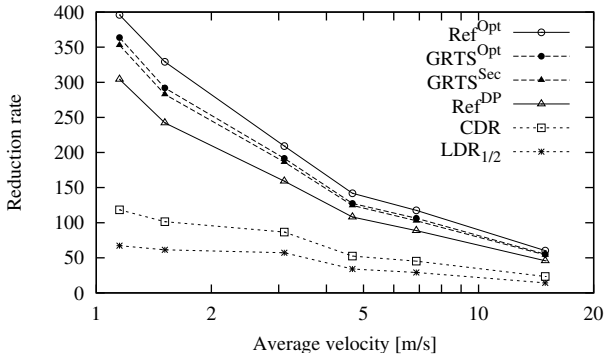

Fig. 7. Reduction rate depending on the average velocity.

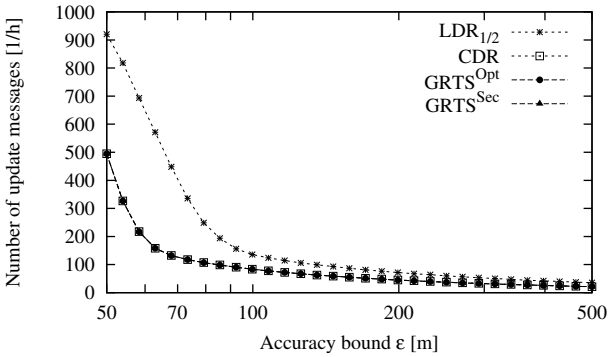

Fig. 8. Number of update messages per hour depending on ϵ.

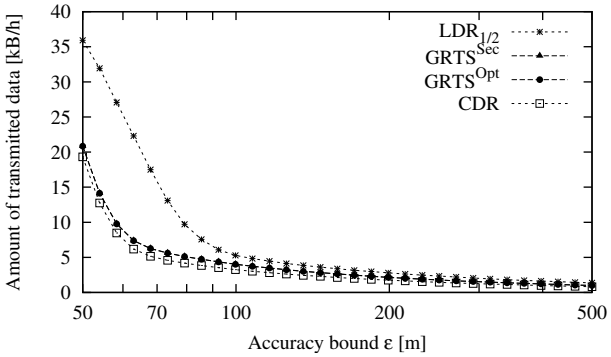

Fig. 9. Amount of data transmitted to MOD per hour depending on ϵ.

55 for $15\,\text{m/s}$. The results further show, that the ratios between the reduction rates of the different algorithms widely are independent of the velocity. In particular, the average reduction rates of GRTS^{Opt} and GRTS^{Sec} of the different groups always are only 8 to 11% and 10 to 14% worse than the reduction rates of Ref^{Opt}, not correlated with the average velocity.

2) Communication Cost: Figure 8 shows the number of update messages generated by GRTS, CDR, and $\text{LDR}_{\frac{1}{2}}$ per hour depending on ϵ. Clearly, both GRTS variants cause the same numbers since they use the same LDR mechanism. The message overhead of CDR is marginally greater compared to GRTS since CDR extends LDR by an additional update condition, cf. Section III. $\text{LDR}_{\frac{1}{2}}$ causes about 60 to 260% more messages than CDR and GRTS, depending on ϵ. With all algorithms the number of messages per hour significantly increases for $\epsilon < 70\,\text{m}$ towards $50\,\text{m}$ since the mobile object frequently impends to reach the accuracy bound causing LDR to send an update. This particularly applies to $\text{LDR}_{\frac{1}{2}}$ as it uses the accuracy bound $\frac{1}{2}\epsilon$.

While with $\text{LDR}_{\frac{1}{2}}$ and CDR each update message only contains a prediction whose origin represents a vertex of $\vec{u}(t)$, GRTS may additionally insert a sequence of vertices. Obviously, this causes GRTS to transmit a higher amount of data than CDR as illustrated in Figure 9. However, the additional amount of transmitted data is small compared to the higher reduction rates of the GRTS variants of more than a factor of two. For example, assuming a header size

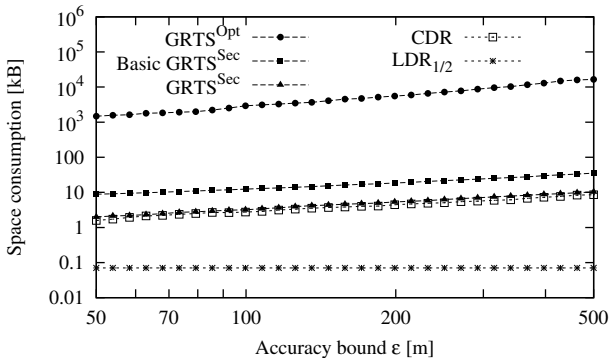

Fig. 10. Space consumption on the mobile object depending on ϵ.

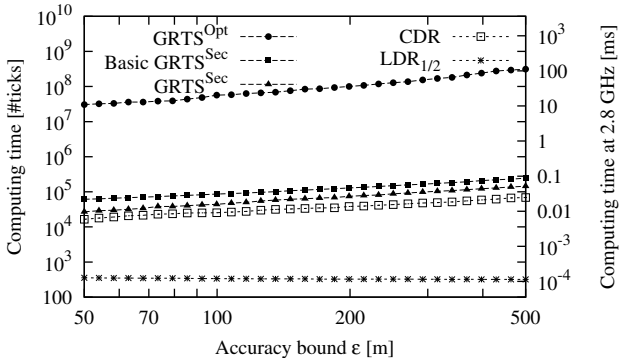

Fig. 11. Maximal computing time per position fix depending on ϵ.

of 28 byte (UDP/IP), it is only about 11%. Clearly, $\text{LDR}_{\frac{1}{2}}$ transmits significantly higher amounts of data due to the larger number of update messages. The amount of data transmitted by GRTS^{Opt} is 0.3% smaller compared to GRTS^{Sec} due to the higher reduction efficiency.

3) Computational Cost: We now analyze the maximum space consumption and computing time per sensed position of $\text{LDR}_{\frac{1}{2}}$, CDR, GRTS^{Opt}, and GRTS^{Sec} on the mobile object. The space consumption is measured in bytes by summing up the space consumption of the different variables and arrays, particularly of the sensing history $\mathbb{S}$. The maximum computing time for processing a new sensed position is measured in processor ticks using the processor time stamp counter. To filter out interrupts of the process under test, we simulated the RTS algorithms without other user processes and repeated each measurement ten times.

Figure 10 shows the maximum space consumption of the RTS algorithms depending on ϵ. The space consumption of $\text{LDR}_{\frac{1}{2}}$ is constant, since it does not store a sensing history. For all other algorithms it increases with ϵ as the sensing history generally contains more sensed positions. With CDR and GRTS^{Sec} the space consumption is significant smaller than 100 kB which seems acceptable for most mobile devices. On the other hand, GRTS^{Opt} may consume more than 10 MB, which may exceed the available memory of some devices.

Basic GRTS^{Sec} refers to GRTS^{Sec} without the optimization of the section heuristic given in Section V-C. Clearly, this optimization saves valuable memory capacity. On average, it reduces the space consumption by 70 to 78%, slightly increasing with ϵ. Moreover, it also reduces the maximum computing time per sensed position as shown in Figure 11.

The computing times of $\text{LDR}_{\frac{1}{2}}$, CDR, and GRTS^{Sec} are all below 1 ms, while GRTS^{Opt} takes up to about 100 ms. Analogous to the space consumption, this may be unacceptable for some mobile devices. Given the fact that the reduction rate of GRTS^{Opt} is only a few percent greater than the reduction rate of GRTS^{Sec}, we argue that GRTS^{Opt} should be preferred to GRTS^{Sec} only if the mobile object has sufficient computational resources and reduction efficiency is of highest priority.

B. Experiences with GRTS-based Tracking System

To gather practical experiences with GRTS, we implemented a prototypical tracking system for multiple mobile objects using GRTS^{Sec}. Figure 13 depicts the system architecture with the major software components *MobileApp*, *ServerApp*, and *KMLClient*.

MobileApp is executed by each mobile object being tracked. It is implemented in the Java programming language and consists of three sub-components: *GPSUnit* parses the NMEA output of the object's GPS receiver. *GRTSAlg* implements GRTS^{Sec} with per-sense simplification as given in Figure 4. *UpdateSender* implements ordered, reliable, encrypted message passing with ServerApp based on UDP over UMTS/GPRS. Moreover, MobileApp offers a GUI plotting the sensed trajectory and the simplified trajectory with the current prediction as shown in Figure 12. The large circle illustrates ϵ while the small circle depicts the maximum sensing deviation δ depending on the current DOP value.

ServerApp is implemented in Java and realizes a simple in-memory MOD storing the vertices $(u_1, \ldots, u_m)$ and the current prediction $(l_{\text{O}}, \vec{l_{\text{V}}})$ of each mobile object. Furthermore, it implements a simple HTTP server which provides the simplified trajectories of the mobile objects in the *Keyhole Markup Language* (KML).

Clients can display the simplified trajectories in real-time by using Google Earth querying ServerApp for the current trajectories every second.

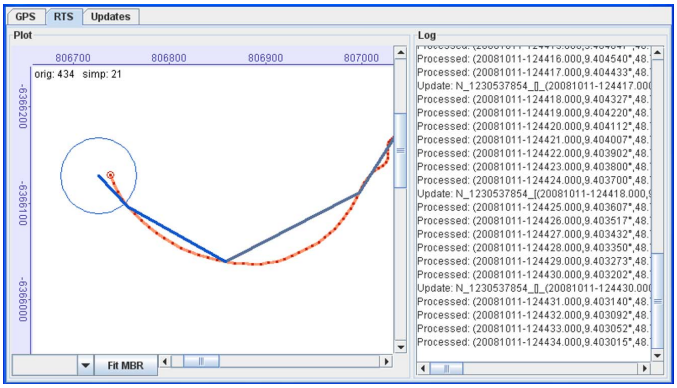

Fig. 12. Screenshot of MobileApp executed on the mobile objects.

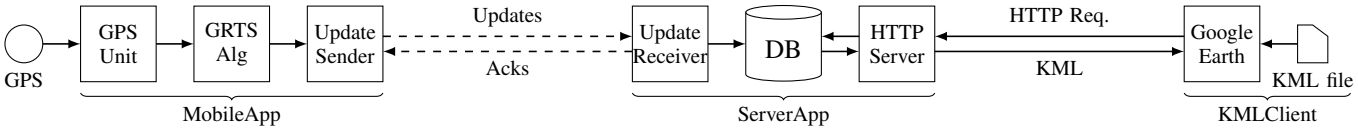

Fig. 13. Architecture of prototypical GRTS-based tracking system.

We conducted several experiments driving a car equipped with an OQO model 01+ subnotebook [20] and a Wintec WBT-300 GPS receiver [21] providing four position fixes per second. This update rate particularly allows for tracking fast objects with small ϵ. During our experiments, we used $\epsilon = 25\,\mathrm{m}$. Besides four network outages lasting several minutes, the prototypical tracking system successfully allowed for tracking the car and its trajectory for more than nine hours from several home computers.

During this experiment, we measured a reduction rate of 70. Per hour, only $60\,\mathrm{kB}$ of data were transmitted to ServerApp including all communication overhead such as retransmissions due to lost UDP packets. These experimental results coincide with the results of our simulations.

In rural areas, not covered by UMTS but only GPRS, we observed latencies between 500 and $1500\,\mathrm{ms}$ for sending an update message. Depending on ϵ such latencies may be too long for remote trajectory simplification. However, they may be significantly reduced by suitable network protocols.

VII. Conclusions

In this paper we studied the problem of remote trajectory simplification for MODs, managing the trajectories of a collection of mobile objects with embedded positioning sensors. With such a system, the trajectory simplification has to be performed on the mobile objects to also minimize wireless communication costs besides the ultimate goal of reducing the trajectory data according to a certain accuracy bound to save storage capacity of the MOD.

Therefore, remote trajectory simplification involves position tracking as well as line simplification. We proposed a novel generic remote trajectory simplification protocol (GRTS) which separates tracking from simplification issues as far as possible. GRTS can be combined with any line simplification algorithm suited for trajectories. We presented two combinations with an optimal simplification algorithm [11] and the section heuristic [4] which allow to trade computational complexity off against reduction efficiency. Both variants outperform all existing RTS approaches at least by a factor of two in terms of reduction efficiency.

Furthermore, we presented an optimization of the section heuristic which reduces its space requirements by three-fourths on average.

Acknowledgments

The work described in this paper was partially supported by the German Research Foundation (DFG) within the Collaborative Research Center (SFB) 627.

References

[1] A. Schmidt, M. Beigl, and H.-W. Gellersen, "There is more to context than location," *Computers & Graphics Journal*, vol. 36, no. 6, pp. 893–902, Dec. 1998.
[2] A. K. Dey and G. D. Abowd, "Towards a better understanding of context and context-awareness," in *Proc. of the CHI 2000 Workshop on the What, Who, Where, When and How of Context-Awareness*, The Hague, Netherlands, Apr. 2000.
[3] R. H. Güting and M. Schneider, *Moving Objects Databases*. San Francisco, CA, USA: Morgan Kaufmann Publishers, 2005.
[4] N. Meratnia and R. A. de By, "Spatiotemporal compression techniques for moving point objects," in *Proc. of the 9th Int'l Conf. on Extending Database Technology*, Heraklion, Crete, Mar. 2004, pp. 765–782.
[5] H. Cao, O. Wolfson, and G. Trajcevski, "Spatio-temporal data reduction with deterministic error bounds," *VLDB Journal*, vol. 15, no. 3, pp. 211–228, Sep. 2006.
[6] M. Potamias, K. Patroumpas, and T. Sellis, "Sampling trajectory streams with spatiotemporal criteria," in *Proc. of the 18th Int'l Conf. on Scientific and Statistical Database Management*, Vienna, Austria, Jul. 2006, pp. 275–284.
[7] G. Trajcevski, H. Cao, P. Scheuermann, O. Wolfson, and D. Vaccaro, "Online data reduction and the quality of history in moving objects databases," in *Proc. of the 5th ACM Int'l Workshop on Data Engineering for Wireless and Mobile Access*, Chicago, IL, USA, Jun. 2006.
[8] J. Gudmundsson, J. Katajainen, D. Merrick, C. Ong, and T. Wolle, "Compressing spatio-temporal trajectories," in *Proc. of the 18th Int'l Symp. on Algorithms and Computation*, Sendai, Japan, Dec. 2007, pp. 763–775.
[9] D. Tiešytė and C. S. Jensen, "Recovery of vehicle trajectories from tracking data for analysis purposes," in *Proc. of the 6th European Congress and Exhibition on Intelligent Transport Systems and Services*, Aalborg, Denmark, Jun. 2007.
[10] R. Lange, F. Dürr, and K. Rothermel, "Online trajectory data reduction using connection-preserving dead reckoning," in *Proc. of the 5th Int'l Conf. on Mobile and Ubiquitous Systems: Computing, Networking and Services*, Dublin, Ireland, Jul. 2008.
[11] M. Iri and H. Imai, *Computational Morphology*. North-Holland Publishing Company, 1988, ch. Polygonal Approximations of a Curve – Formulations and Algorithms, pp. 71–86.
[12] D. Pfoser and C. S. Jensen, "Capturing the uncertainty of moving-object representations," in *Proc. of the 6th Int'l Symp. on Advances in Spatial Databases*, Hong Kong, China, May 1999, pp. 111–131.
[13] P. K. Agarwal, S. Har-Peled, N. H. Mustafa, and Y. Wang, "Near-linear time approximation algorithms for curve simplification," *Algorithmica*, vol. 42, no. 3–4, pp. 203–219, Jul. 2005.
[14] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Canadian Cartographer*, vol. 10, no. 2, pp. 112–122, Dec. 1973.
[15] N. Hönle, M. Großmann, D. Nicklas, and B. Mitschang, "Preprocessing position data of mobile objects," in *Proc. of 9th Int'l Conf. on Mobile Data Management*, Beijing, China, Apr. 2008.
[16] O. Wolfson, A. P. Sistla, S. Chamberlain, and Y. Yesha, "Updating and querying databases that track mobile units," *Distributed and Parallel Databases*, vol. 7, no. 3, pp. 257–287, Jul. 1999.
[17] A. Leonhardi and K. Rothermel, "A comparison of protocols for updating location information," *Cluster Computing: The Journal of Networks, Software Tools and Applications*, vol. 4, no. 4, pp. 355–367, Oct. 2001.
[18] A. Čivilis, C. S. Jensen, and S. Pakalnis, "Techniques for efficient road-network-based tracking of moving objects," *IEEE Trans. on Knowledge and Data Engineering*, vol. 17, no. 5, pp. 698–712, May 2005.
[19] OpenStreetMap Project. [Online]. Available: www.openstreetmap.org
[20] OQO, Inc. [Online]. Available: www.oqo.com
[21] Wintecronics Ltd. [Online]. Available: www.wintec.com.tw