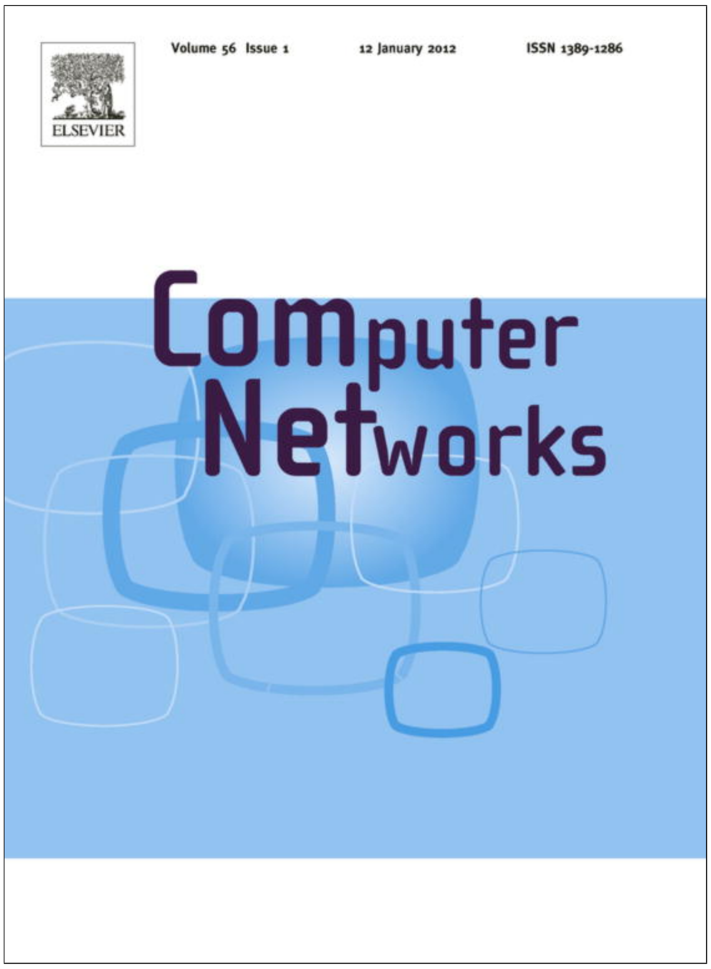

Volume 56 Issue 1
12 January 2012
ISSN 1389-1286
ELSEVIER
Computer Networks

Computer Networks 56 (2012) 345–364

Contents lists available at SciVerse ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Path approximation for multi-hop wireless routing under application-based accuracy constraints ☆

Mustafa O. Kilavuz, Murat Yuksel *

Computer Science and Engineering Department, University of Nevada – Reno, Reno, NV 89557, USA

ARTICLE INFO

Article history:
Received 20 June 2010
Received in revised form 3 August 2011
Accepted 19 September 2011
Available online 1 October 2011

Keywords:
Application-based routing
Top-down design
Trajectory-Based Routing

ABSTRACT

Provisioning of rich routing building blocks to mobile ad hoc networking applications has been of high interest. Several MANET applications need flexibility in describing paths their traffic will follow. To accommodate this need, previous work has proposed several viable routing schemes such as Dynamic Source Routing (DSR) and Trajectory-Based Routing (TBR). However, tradeoffs involved in the interaction of these routing schemes and the application-specific requirements have not been explored. Especially, techniques to help the application to do the right routing choices are much needed. In this paper, we consider techniques that minimize routing protocol state costs under application-based constraints. We study the constraint of "accuracy" of the application's desired route, as this constraint provides a range of choices to the applications. As a crucial part of this concept, we investigate the tradeoff between the size of packet headers (needed to store end-to-end paths) and the network state (needed to store routing tables). We, then, apply the concept to the case of TBR with application-based accuracy constraints in obeying a given trajectory. We begin with simple discrete models to clarify the tradeoff between the packet header size and the network state. We show that the problem of accurate approximation of a trajectory (a.k.a. an application-specific end-to-end path) with the objective of minimizing the cost incurred due to header size and network state is difficult to solve optimally. We design an exhaustive search method as well as a genetic algorithm to find the optimum solution. We also develop heuristics solving this problem with smaller computational complexity and illustrate their performance. Finally, we explore ways of customizing our trajectory approximation framework for power-scarce or memory-scarce networking scenarios.

© 2011 Elsevier B.V. All rights reserved.

1. Introduction

As the reach of networked devices increases, the network infrastructure needs to support *application-specific* designs due to the increased variety of reached applications. This need is more pronounced in sensor networks, especially in terms of the routing functions provided by the network. To accommodate various application-specific routing needs, the *expressiveness of the routing interface* must be at sufficient granularity. The typical wireless routing interface has been a shortest-path interface with simplistic primitives: `Send(src,dst,data)` and `Receive(src,dst,data)`. Recently, there have been a lot of efforts in improving this interface with an "options" argument in the primitives, i.e., `Send(src,dst,data,options)` and `Receive(src,dst,data,options)`. Recent work [2,3] tackled this problem in the general routing context without customizing it for multi-hop wireless routing.

In terms of application-specific routing functions, previous work showed that very flexible routing functions can be implemented [4,5] by using network-specific properties

☆ Earlier version of this work appeared in [1] as a conference paper. This work is supported in part by the US National Science Foundation award 0627039.

* Corresponding author.

E-mail addresses: mkilavuz@cse.unr.edu (M.O. Kilavuz), yuksem@cse.unr.edu (M. Yuksel).

1389-1286/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2011.09.013

such as geographic routing [6] capability. Such routing functions enabled application-specific traffic engineering [7], e.g., load-balancing among multiple paths instead of a single shortest-path. On the other side, over a network where several routing options are provided, applications face the problem of selecting the right options and appropriately identifying their constraints, such that applications' goals are met. Some recent work pinpointed the complexity of this issue within limited contexts, e.g., minimal/maximal exposure path selection [8].

Though provisioning of rich routing building blocks to multi-hop wireless networking applications is of high interest, application-specific requirements and constraints emphasize the challenge of designing routing schemes. It is a major challenge to include application-based "constraints" (which can be of various type such as path quality, path accuracy, and path cost/price) as an additional argument to the routing primitives, which we investigate in this paper. In particular, we investigate tradeoffs involved in the interaction between wireless routing and the application-specific constraints. Especially, low-complexity techniques to help the application to do the right routing choices are much needed, as the time to make such routing decisions is very minimal for mobile nodes [9]. We consider the concept of minimizing of routing state under application-specific path constraints; and how this concept can guide the investigation of various inter-layer design issues from a top-down perspective where applications are offered more flexibility in expressing their desires or needs. We, particularly, focus on finding the right tradeoffs in function placement among the application and the routing layers, such that the overall cost of two conflicting goals is minimized: (i) accommodating application's end-to-end requirements with maximum quality and (ii) performing routing with minimum state and resource costs.

We base our study to the key optimization problem of minimizing routing state costs under application-specific constraints. Specifically, we formulate the problem of minimum cost (i.e., the state to be stored) Trajectory-Based Routing (TBR) [4,5] under application-specific constraint of "path accuracy". That is, we consider an application which desires its packets to follow a trajectory with a bounded error in obeying the trajectory. We illustrate how such a constraint can be quantified. In TBR, as shown in Fig. 1, the application at the source node embeds a desired "ideal trajectory" into packets' headers. This ideal trajectory is to be followed by the packets. The intermediate nodes are assumed to be able to decode the ideal trajectory from the packet header and decide which neighbor to forward the packet next such that the packet obeys the ideal trajectory as much as possible. Though TBR can implement highly flexible routing options, the decision to select the next neighbor optimally can be quite time consuming which breaks the basic premise of simple packet forwarding. Thus, approximating the ideal trajectory is needed so that the nodes can work on the "approximate trajectory" rather than the ideal trajectory which can be quite complicated depending on the application needs. The approximate trajectories can be a concatenation of several pieces of "easy to handle" trajectories such as a line, a polynomial curve, or a Bézier curve. Then, this approximate trajectory is used to make forwarding decisions for the "actual trajectory/route", as illustrated in Fig. 2.

From a larger point of view, our work aims to approximate a path under application-specific constraints. The goal is to perform this path approximation such that the routing overhead is minimized while the application-specific constraints are strictly followed. Particularly, we formulate the *trajectory approximation* problem as a combinatorial optimization problem, and show that it is NP-hard. In our formulation, we allow a number of representations to be used to approximate a portion of the trajectory, such as straight lines, polynomial curves, or Bézier curves. We carry the trajectory approximation problem to a discrete space and outline an exhaustive search algorithm (with pruning) that guarantees finding the global optimum. Since this trajectory approximation is supposed to be frequently performed (i.e. whenever there is data to be sent with TBR) at the source node where the application resides, the computational complexity of the solution is of high importance. Thus, to reduce the computational time requirements of our solution, we map the problem to a genetic algorithm and design two heuristics as well.

In brief, contributions of this paper include:

- *The concept of minimizing routing state under application-based constraints.* We applied this concept on the particular problem of minimizing TBR state under application-based accuracy constraints.
- *Formulation of the trajectory approximation problem minimizing the routing state.* We formulated the problem of generating an approximate trajectory within an application-defined error bound such that TBR state (i.e., the aggregate of packet header length and network/router state) is minimized.
- *Proof that the trajectory approximation problem is NP-hard.*
- *Solutions to solve the trajectory approximation problem.* We designed an exhaustive search algorithm, a genetic

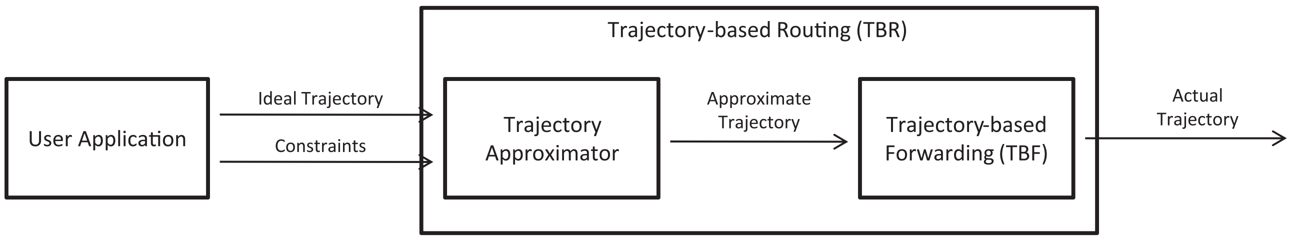


Fig. 1. Trajectory-Based Routing (TBR) framework.

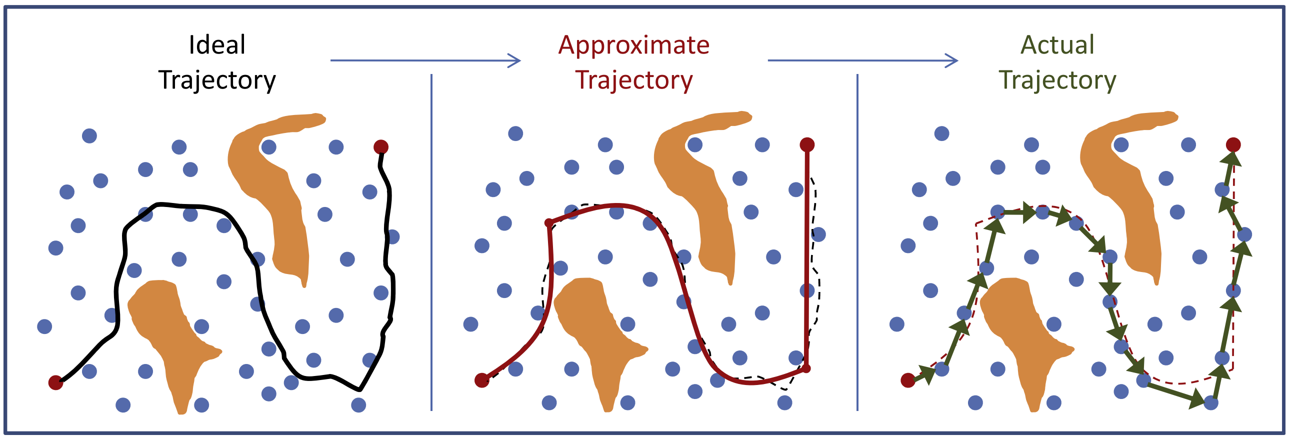


Fig. 2. Ideal, approximate, and actual trajectories.

algorithm, and two heuristics to solve the trajectory approximation problem.

- *Customized the trajectory approximation problem for power-scarce networks.* We increased the perceived cost of packet header length and illustrated how the approximate trajectories should be changed in a power-scarce network.

The rest of the paper is organized as follows: First in Section 3, we describe the TBR in detail. We outline the problem of approximating an ideal trajectory under application-specific accuracy constraint in Section 4. Then, Section 5 details four methods to solve the approximation problem. We present results of our simulation experiments in Section 6 and summarize our work in Section 7.

2. Related work

There has been a lot of recent interest mobile ad hoc networks (MANETs) [10–13,6,14], and sensor networks. Main focus of MANET research has been on scalability and complexity issues raised due to underlying dynamism, specifically on routing. Routing tackles the problem of establishing an *indirection* from a *persistent name (or ID)* to a *locator*. In today's routing, this indirection translates to IP address prefix being the ID and next hop router being the locator. Routing protocols such as AODV [11], DSDV [15], and others, facilitate this indirection by building and maintaining routing tables that map *destination IDs* to *next-hop IDs* based on node or link state information. Dealing with such dynamic indirections usually involves *delaying* the creation of the ID-to-location mapping (a.k.a "*late-binding*"), and reducing the *number and dynamism* of bindings (e.g., using hierarchy [12,16] or consistent hashing [17,14] or filtering [18]). MANET routing has grown into two subclasses: *proactive* routing [15] (using early binding) and *reactive or on-demand* routing [10,11] (involving late-binding).

2.1. Application-based routing

Application-based routing and network design, the most relevant to our work, has been of a key focus area in sensor networks research. Though providing routing expressiveness and flexibility to user applications has been of a great interest [19–21], application-based networking and routing has been mainly studied in the context of sensor networks. Active networking [22] and cognitive networking [23] are efforts towards the same goal of customizing network behavior for various user applications. In the general context of routing, defining application-specific custom routes through user-defined routes [24], through a declarative language [2,3,25], or through concatenations of several user-chosen contracts [26] have attracted interest. However, these efforts either require significant router computation or autonomous system level route descriptions, both are impractical in wireless routing.

The very first application-based wireless routing work was an extreme scheme of Dynamic Source Routing (DSR) [10], which gives a full flexibility to the user application to define the routes. Since DSR could not scale to sufficient number of nodes, schemes like Trajectory-Based Routing (TBR) [4] and landmark selection [27] were proposed to reduce the packet header costs and yet still give reasonable flexibility for users to define their desired routes as trajectories. Recent wireless routing studies focused on customizing routing metric for applications [28] and centralized optimization of routing for application-based goals [29,30]. Our work introduces a new knob of "application-specific constraints" into the routing interface, while considering routing state scalability issues.

2.2. State scalability and complexity

Scaling challenges in terms of network size are addressed to a limited extent by building hierarchies (e.g., ZRP [12], Adaptive Clustering [31,32]). Caching or replica usage is common in all dynamic networks to leverage locality of reference. Protocols like HSLS (Hazy Sighted Link State [18]), Fisheye [16] and DREAM [13] filter updates sent to remote nodes. GLS [14], GHT [33], and MAP [34] use geographical/geometric forwarding and consistent hashing techniques to scale. However, hierarchies and link-states become harder to maintain with increase in mobility, traffic load and network size [18].

MANETs tend to take advantage of *geography or location* information to help scale routing and reduce the complexity of state maintenance. Geographic routing in MANETs is attractive because forwarding is achieved by using cartesian properties (like direction and distance). When a next hop is unavailable, several fallback mechanisms are available [6,35,36]. ID-to-GeoLocation is still a problem. DREAM [13] achieves this by proactive flooding of mappings and LAR [37] uses reactive flooding. LANMAR [38] employs a hierarchy to avoid global flooding, but is susceptible to nodes at the top of the hierarchy being mobile. Terminodes routing [39,40] uses a fixed set of geographic points (anchors) to guide remote forwarding. Hubaux et al. suggest placing the ID-to-location mappings at geo-locations derived from a hash of the ID. As networks become larger and denser, maintaining each entry in the routing table becomes increasingly more costly. Additionally, as node *mobility* increases, increased node location/link information dissemination further adds to overhead. This phenomenon led to several studies of probabilistic schemes [41,42].

2.3. Constrained routing

Finding paths within a set of constraints has been studied extensively in the areas of QoS routing and traffic engineering [43]. Most of these investigations focused on achieving delay-constrained least-cost (DCLC) routing in wireline networking, since real-time multimedia applications care much about the maximum end-to-end delay. The main target of these schemes has been to find a path so that the end-to-end path delay is capped with pre-defined value and approximation error from the DCLC path is bounded with a pre-defined ratio. Achieving polynomial time heuristics to such NP-hard multi-constraint routing problems received a lot of attention, but the focus mainly stayed in wireline networking [44,45] with some recent expansion to the stationary multi-hop wireless networking [46,47].

Our work relates to this literature since we are trying to approximate an ideal trajectory (i.e. end-to-end path) within an accuracy constraint explicitly defined by an application. We show how our problem set can be reduced to the constrained shortest path problem. A crucial difference is that, however, our goal is not to minimize the approximation error but rather the aggregate routing state costs. Since we consider a MANET environment, time complexity of the heuristics is of essence and thus we do not focus on achieving a bound on the approximation error. The heuristics designed by Chen et al. [45] are similar to ours in that they also try to reduce the granularity of the discretization step by either randomizing the discretization points or selecting them according to the end-to-end path delay via combinatorial optimization. The difference in our problem, however, is that the underlying graph is unknown at the beginning, and the heuristic has to devise calculations to explore the graph since the edge weights correspond to the application-specific costs (i.e., path accuracy) between two points on the path. Our heuristics also leverage the end-to-end path characteristics such as the trajectory length, but apply a very different process to find the approximate path. Particularly, we devise recursive inspection of the path to find the minimum number of discretization points (i.e. split points) to achieve an explicit accuracy constraint.

3. Trajectory-Based Routing and accuracy constraint

3.1. Ideal, actual, and approximate trajectories

Trajectory-Based Routing (TBR) [4,5] suggests that the source node encodes the trajectory into packet headers, and then intermediate nodes forward the packets according to the trajectory decoded from their headers so as to make them traverse the source-defined trajectory as much as possible. The "ideal trajectory" is received from the user application which is the demanded path of the traffic flow. In Fig. 2, the ideal trajectory is shown as a hand-drawn curve. This ideal trajectory can be formed based upon the user application's goals, but the network routing still has to determine the best way of attaining the goal of forwarding the packets along this ideal trajectory.

Since the trajectory will be encoded into the packet headers, the ideal trajectory needs to be represented by formulations which can be understood by all nodes in the network. Implementing very complex trajectory decoding hardware at every node is not practical, and thus these trajectory representations must be of simple type such as a straight line, a polynomial curve, or a Bézier curve. This means that the ideal trajectory may not be representable exactly, depending on how complex the ideal trajectory is. Thus, instead of representing the exact ideal trajectory, we focus on generating an "approximate trajectory" which is easier to represent, however it has slight differences than the ideal trajectory. To give the user application a knob on how the routing function generates this trajectory, we consider the limits of the difference between the ideal trajectory and the approximate trajectory is given by the user application as an "accuracy constraint".

Although the approximate trajectory is optimized in the best way under the constraints, it is still sometimes too complex to encode the whole trajectory into the packet headers. Hence, the approximate trajectory is divided into small pieces that each piece can be encoded into packet headers separately. To manage this, some special intermediate nodes (SINs) [5] are selected around the points where the approximate trajectory is divided. The SINs store the information of the trajectory piece starting from that point only. The packets departing from the source receives only the information about the first piece. When they reach the end of this piece, they acquire the next piece's routing information from the corresponding SIN. They keep changing the routing information in their headers whenever they visit another SIN until they reach the destination. For example, in Fig. 3, the approximate trajectory is divided into two pieces, the SIN is located at the point which we call the *split point*. The routing information stored in the packet headers and the routing tables in the SINs are "costs" of maintaining this TBR session. This cost can be translated into other dimensions such as power consumption of the TBR session. We call the cost caused by the routing information (source, destination, next hop, trajectory representation etc.) stored in the packet headers as *packet header cost* and by the routing information stored in the SINs as *network state cost*.

Various trajectory-based forwarding (TBF) techniques were proposed [4,5] to manage the packet traffic over the

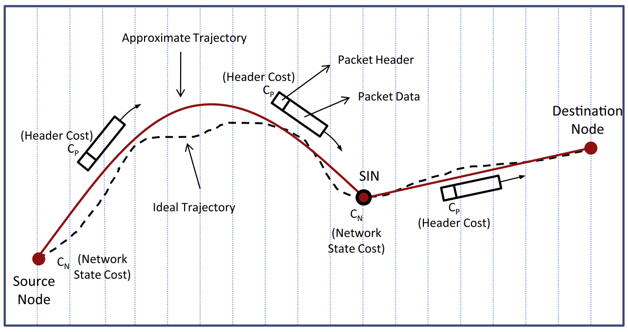


Fig. 3. Packet forwarding in TBR.

approximate trajectory. Depending on the density and placement of the nodes, the packets may not follow the approximate trajectory exactly. Thus, the "actual trajectory" the packets follow may be different than the approximate trajectory as shown by arrows in Fig. 2. In this paper, we do not focus on the actual trajectory the packets take, but rather focus on generating the best possible approximation of the ideal trajectory such that the actual trajectory will have the highest likelihood of being close to the ideal trajectory with a minimal routing state cost.

3.2. Application scenarios

Being able to route on a curve or ideal trajectory, Trajectory-Based Routing (TBR), is needed for several multi-hop wireless network protocols and applications [4,5]. TBR is a geographic routing protocol, and thus, requires each node to have localization capability. In return to this cost, TBR provides great advantages to wireless applications. We discuss some of them below.

3.2.1. Location-specific sensing

In a wireless sensor network deployed over a large geographical area, it is desirable to make location-specific sensing or data collection. Being able to measure temperature around a lake or within layers of a mine are examples of such sensing applications needing the capability of routing on a curve. The capability of defining accuracy of the approximate trajectory further enables such sensing applications to make measurements within a pipe around the ideal trajectory, as illustrated in Fig. 5.

3.2.2. Avoiding obstacles or hostile areas

Several multi-hop wireless applications can utilize the capability of routing on a curve with a particular accuracy in order to avoid an area of interest of disinterest. A battlefield application trying to send sensitive information to a destination might desire to avoid a hostile area through which the information should not traverse. Similarly, a peer-to-peer smartphone application might go around a void area composed of an obstacle such as a big park by routing on a curve.

3.2.3. Source and multi-path routing

Being able to routing on a curve brings two key capabilities which are otherwise not possible in a mobile ad hoc multi-hop wireless network.

First, by defining a routing curve it is possible to achieve source routing by defining a "fixed" path from a source to a destination. Though DSR achieves source routing in mobile ad hoc networks, it still needs to update the state once intermediate nodes in the network change. By fixing a curve, which is defined by coordinates, it is not necessary to update the curve definition even if an intermediate node moves away from the curve. So, TBR is more scalable in achieving source routing in a multi-hop wireless network. However, the cost is the fact that nodes must have localization service or hardware, which has become ubiquitous recently.

Second, by defining multiple routing curves between a source-destination pair, it is possible to achieve multi-path routing in a multi-hop wireless network. Such multi-path routing capabilities are available in wireline networking via layer 2 technologies like MPLS. TBR is a very promising approach towards achieving multi-path routing in a highly dynamic multi-hop wireless network. An application can define these multiple paths such that they have minimal overlap and potentially achieve higher aggregate end-to-end throughput.

3.2.4. End-to-end traffic engineering

Capabilities such as scalable source routing and multi-path routing allow end-to-end traffic engineering and load balancing strategies which are otherwise not practical in a dynamic multi-hop wireless network. Intuitively, if all source-destination pairs use shortest-path routing with a single end-to-end path, then a congested hot spot emerges in the middle of the multi-hop wireless network. Existing solutions to this load balancing problem typically involves heavy centralized computations which can only work when the multi-hop wireless network is stationary or very close to stationary. By dynamically adjusting traffic rates on each end-to-end trajectory as well as the shape of the trajectories, it is possible to more evenly distribute the traffic load on the network nodes in a completely end-to-end basis.

4. Trajectory approximation problem

4.1. System model

When a trajectory is approximated by a series of representations, each piece of the trajectory is associated with some cost in terms of packet header length and network state. We construct the problem of minimizing this aggregate state cost of the whole trajectory while satisfying the application-defined accuracy constraint. We define the constraint as the error by which the approximate trajectory can deviate from the ideal trajectory. Each piece of the approximate trajectory has an error value that it contributes to the total error of the whole approximate trajectory. The accuracy constraint plays the role of restricting the total error of the whole approximate trajectory. We make the following modeling assumptions for the problem:

- There are k choices for representing a given piece of the trajectory. These are denoted as $r_1, r_2, \ldots, r_k$, where r_i is the representation selected for the ith piece of the trajectory. The representations are well-defined lines such as a straight line, a polynomial curve, or a Bézier curve.

- The space in which the trajectory exists is discretized, and there are a maximum of m split points (excluding source and destination), or $m + 1$ pieces into which the trajectory can be split into.
- Since each piece of the trajectory is represented using a single type of representation, we construct a matrix Q of dimensions (m, k), where Q_{ij} is a binary variable denoting which representation is used for a particular piece of the trajectory. This implies that at most a single entry in a row is equal to 1.
- If the algorithm selects a particular representation r_i for a piece of the trajectory, then it makes use of a subroutine to compute the error associated with the representation. This error is calculated such that r_i is fit to its corresponding piece of the ideal trajectory with the least error possible by using algebraic equation solving methods [48].
- There is a packet header cost C_P and a network state cost C_N for each piece of the trajectory, depending on the representation used for the piece of the trajectory being considered.
- The split points are chosen such that each piece of the trajectory has the same or similar length, which means that the packet header cost C_P and the network state cost C_N are independent of the length of the corresponding piece of the trajectory. This is a simplifying assumption and we will later illustrate in Section 6.2.2 how it can be relaxed in a realistic scenario.

We can now formulate the following binary program:

$$\min \quad \sum_{i=1}^{m}\sum_{j=1}^{k} C_P(j)Q_{ij} + C_N(j)Q_{ij} \tag{1}$$

$$\text{subject to}: \quad \sum_{i=1}^{m}\sum_{j=1}^{k} e(Q_{ij}) \leqslant E \tag{2}$$

$$\sum_{j=1}^{k} Q_{ij} \leqslant 1 \quad \forall \quad i = 1 \cdots m \tag{3}$$

$$Q_{ij} \in \{0,1\}. \tag{4}$$

In the formulation (1)–(4), the constraints (2) denote the error associated with the representation of each piece of the trajectory. In constraints (2), the sum of the errors must not exceed an application-defined error E which we assume is an input to the problem. The constraint (4) just states that Q_{ij} is a binary variable. Note that, of the m split points available, we do not necessarily use all of them to approximate the trajectory, which is captured in constraints (3). If a split point i is not to be used as part of the solution, then $\sum_{j=1}^{k} Q_{ij}$ will be 0, which still satisfies the constraint (3). Even though there are m split points available for approximating the trajectory, the best solution to the problem might not need to use all of the m split points. Such cases can happen when the ideal trajectory is simple (e.g. very similar to a straight line) or the accuracy constraint is loose (i.e. large error/deviation from the ideal trajectory is allowed by the application). In such a case, there might be too many split points than needed, and the best approximation to the ideal trajectory might only need a subset of the available split points.

4.1.1. NP-hardness

The formulation above can be modeled using a graph on $m + 2$ vertices (including source and destination nodes), with edges between all nodes (complete graph on $m + 2$ vertices) implying that any of these edges can be chosen in an approximation of the trajectory, subject to the error constraint. Next, we allow multiple edges between two vertices, each edge corresponding to one type of approximation for the corresponding portion of the trajectory. The edges are associated with an error measure as well as a cost due to C_p and C_N. While the formulation in (1)–(4) is *node* based, the edge formulation is implicit. For example a solution which selects some $l \leqslant k$ equal to 1 for some node i, and all other entries in the matrix as 0, implies that the approximation are the edges (s, i) and (i, t) where s and t are source and destination nodes respectively.

By considering the errors associated with edges as "weights" and the state costs as "costs", we note that the problem of finding the approximate trajectory giving minimum state cost (while satisfying the accuracy constraint) is identical to the *shortest weight-constrained path*, which is a well known NP-Complete problem [49]. This is represented diagrammatically in the Fig. 4. In Fig. 4, each edge is associated with a cost as well as a representation error. This is represented on the edges $(S, 1)$ as (C_{S1}, E_{S1}) and $(1, 2)$ as (C_{12}, E_{12}), though all edges are associated with such numbers.

4.2. Cost calculations

The costs C_P and C_N in the objective (1) represent the cost incurred for selecting an approximation for a given portion of the trajectory. We devised a realistic method to calculate these costs in terms of actual bytes. For each representation we calculate how many bytes we have to store additionally in the packet header (i.e., C_P) and in the intermediate nodes (i.e., C_N). In the rest of the paper, we consider three different representations. The more complex the representations are, the more costly they will be. Below are the three representations we use:

- *Line*, $y = ax + b$: Two end points (x_1, y_1) and (x_2, y_2) are enough to express a line. We assume that each parameter is a double. Considering each double takes 8 bytes, the space needed to express a line is $4 \times 8 = 32$ bytes.
- 2nd *degree polynomial curve*, $y = ax^2 + bx + c$: For a 2nd degree curve, we have to store x_1, x_2, a, b, and c. We do not need y_1 and y_2 since they can be calculated by putting x_1 and x_2 into the equation. Thus, the space needed for a 2nd degree polynomial curve is $5 \times 8 = 40$ bytes.
- 3rd *degree polynomial curve*, $y = ax^3 + bx^2 + cx + d$: Similarly, for a 3rd degree curve, we have to store x_1, x_2, a, b, c and d. Again, we do not need to store y_1 and y_2. So, the space needed is $6 \times 8 = 48$ bytes.

For each segment we calculate the total cost as the sum of C_P and C_N, both of which are dependent to the cost of the

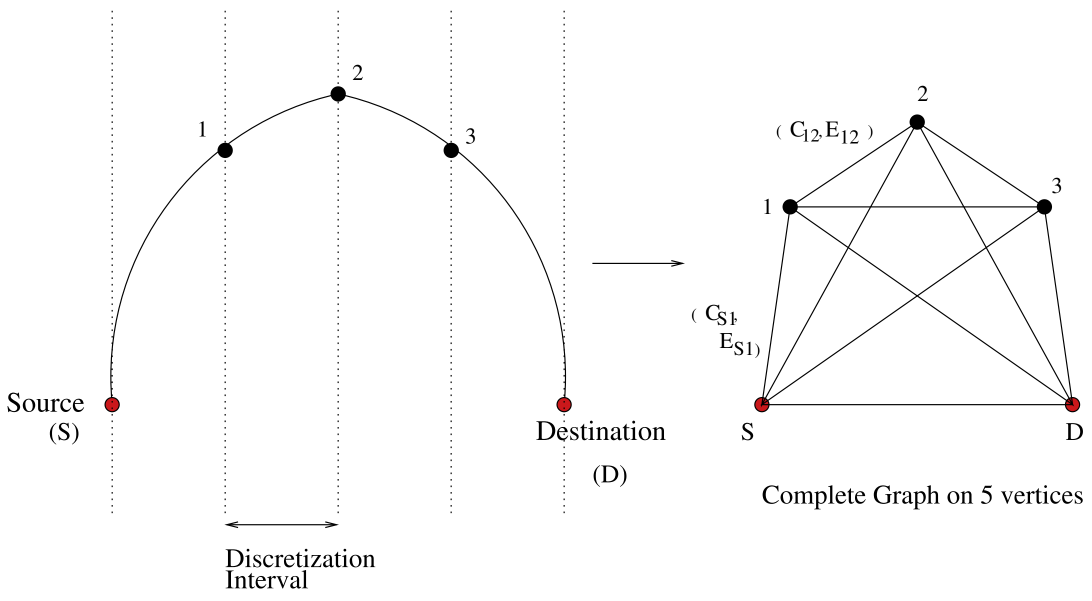


Fig. 4. A sample graph formulation of the discretized trajectory approximation problem.

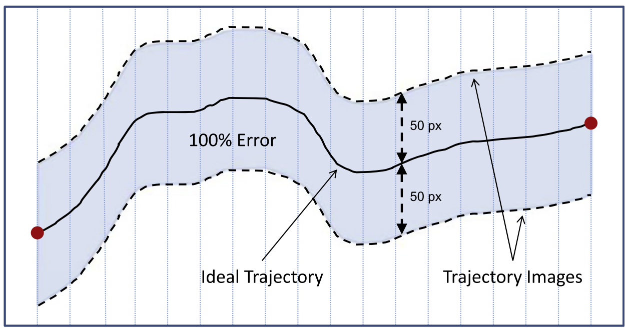


Fig. 5. The application-based accuracy constraint is defined in terms of error from the ideal trajectory.

representation used. Let the representation cost of r_j be R_j, i.e., 32 or 40 or 48 bytes. Then, we can rewrite the aggregate cost as $c_P R_j + c_N R_j$, where c_P and c_N are the weight constants of the packet header and the network state costs respectively. Depending on the application and the context in which TBR session is taking place, these constants *may* depend on the lengths of the trajectories, the average traffic flow over the network, number of connections etc. Unless otherwise said, we assume that $c_P = c_N = 1$. We will also show how different weights effect the structure of the approximate trajectory.

4.3. Error measures for trajectory approximation

In order to let the user application express its accuracy constraint as well as to determine the quality of our trajectory approximation, we define a way of quantifying the trajectory approximation error. We assess the approximate trajectory, by means of an aggregate error, which is the sum of the errors in representation of each piece. We define the error in terms of the *deviation area*, i.e., the total area between the ideal trajectory and the approximate trajectory. To make it more generic we defined the error in percentage instead of any unit of area. Of course, for that, we have to define what "100% error" is, which we think that should be something intuitive. For that, we draw projections of the ideal trajectory above and below it by leaving a constant distance in between. We, then, define "100% error" as the area between the two projections. As shown in Fig. 5, we defined the distance between the ideal trajectory and a projection as 50 pixels in our experiments. So, 100% error is $(50 + 50) \times width$ unit area. This distance can be different according to the size of the space and might be defined as a percentage of the height of it. For instance, 1/4 or 1/8 of the height of the space.

5. Methods to solve the problem

As the trajectory approximation is an NP-hard problem, fast solutions are needed. In practice, this problem will be solved pretty frequently by the nodes where applications reside. In general, before data transmission starts in an end-to-end session using TBR, this trajectory approximation problem must be solved. Note that such implementation of the trajectory approximation will not require an ongoing communication to individual nodes that may be residing along the end-to-end trajectory. First, the approximate trajectory will be calculated at the source node. Then, the information for each segment of the approximate trajectory can be conveyed to the related SINs along the trajectory via a "probe" packet before the data transmission starts. The probe packet will contain all the information about the whole approximate trajectory. Once each SIN knows the specifics of the trajectory representation in the next segment, it will be able to forward individual data packets based on the positions of its neighbors and forwarding needs (e.g., faster forwarding or more accurate forwarding) determined by the application. A detailed discussion and evaluation of such implementation issues are available in [5].

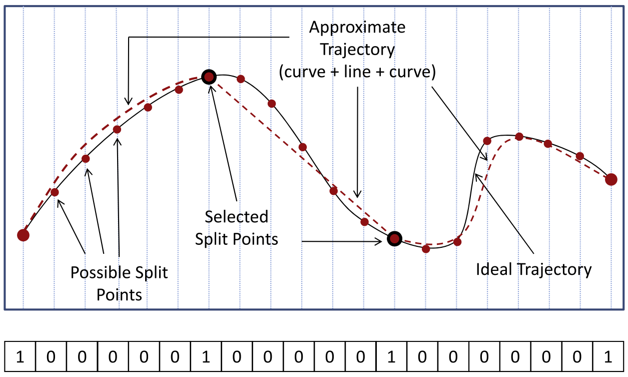


Fig. 6. Split points, trajectories, and selection bits.

As shown in Fig. 1, in our trajectory approximation techniques, we do not consider the positions of the actual nodes in the network while solving the optimization problem. However, it is possible to make our approach applicable to a real network as long as the resolution of the split points is made compatible with the density of the nodes in the actual network. For example, if the resolution of our discretization is not finer than the average distance between two neighbors (i.e., a measure of node density), then the SINs our techniques choose will most likely have at least one actual node residing close to them. To guarantee availability of at least one actual node close to the SINs, maximum distance between two neighbors can be used as the threshold for our discretization resolution.

We now outline four techniques to perform the trajectory approximation: An exhaustive search technique guaranteeing the best solution, a genetic algorithm providing very good solutions in shorter running times, and two heuristics providing solutions in very short running times.

5.1. Exhaustive search with pruning

This algorithm tries all possible combinations of split points, representations, and chooses the best possible solution yielding the minimum routing state cost within the accuracy constraint. With sufficient resolution in the discrete space, this algorithm is guaranteed to find the optimum solution. First we select the possible split points on the trajectory. The distance between any consecutive split points must be the same on the x-axis and should be small enough to make it possible to find the best solution. However, the number of these points should be small enough to have a reasonably short running time. The exponential growth of the running time prevents us to have too many of these points.

The algorithm selects a subset of all possible split points, where the selection is denoted by a binary value (1: selected, 0: ignored) for each possible split point. These points will be used as the end points of the segments of the estimate trajectory. That is, 1 means that the corresponding point is selected as a split point, while 0 means that the corresponding point is ignored. So, a representation that is fit between two consecutive split points forms a segment of the approximate trajectory. For every segment, a representation with its own error and cost values will be chosen. The approximate trajectory formed by these representations which is below the tolerated error bound and has the minimum aggregate cost will be chosen as the best solution.

An example is shown in Fig. 6. 2 of 19 possible split points are selected. Including the source and destination points we have 4 points which are the end points of 3 segments that we will represent by one of the 3 representations, i.e. line, or a second or third degree curve. Here, a 2nd degree curve, a line and a 3rd degree curve forms the approximate trajectory.

We applied pruning [50] method to reduce the running time by pruning non-optimal solutions whenever possible. Specifically, we begin from the source point and go up to the destination point by giving 0 or 1 to each bit. When we give 1, we branch our searching tree for every type of representation. When we put a representation, we calculate the total error and the total cost so far (including the preceding segments). If we have already exceeded the error bound or the cost of the current best solution (if there is any), we stop looking for a solution having the current bit sequence. Because it is obvious that whatever value the rest of the bits get, we will not be able to find a good solution.

5.2. Genetic algorithm

Genetic algorithms (GAs) are search algorithms based on the mechanics of natural selection and genetics [51]. We have a population of members that each member has a chromosome which stores an approximate trajectory with the representations to be used and a fitness value showing the quality of the member. In every step, a new generation is generated by coupling the current members. After some time, the best member is selected as the best solution.

Assume we have N possible split points, as shown in Fig. 7, each of which can be selected for approximating the trajectory. Similar to the exhaustive search approach, the first bit is the source point's followed by N bits for N possible split points, and another bit follows them for the

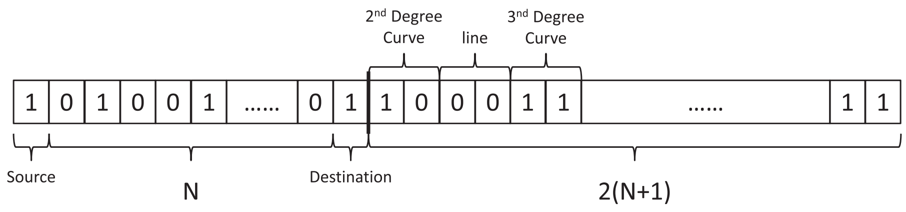


Fig. 7. A sample chromosome in GA solution to the trajectory approximation problem.

destination point. The bits for source and destination are always 1. We included them in the chromosome for the ease of computation and implementation purposes. The remaining part of the chromosome has 2 bits for each of the possible split points and the source point, which makes total of $2(N+1)$ bits. These 2 bits define which representation will be used between the consecutive split points. 00 and 01 represent a line, while 10 represents a 2nd degree curve and 11 represents 3rd degree curve. If there were more than four possible representations, then we would need more than 2 bits for each point. For example, in Fig. 7, the first segment of the approximate trajectory is a 2nd degree curve between the source point and the first split point. A 3rd degree curve follows it between the first and the second split points. Line is omitted.

The initial generation is filled with members who have a chromosome of $3N+4$ random binary numbers. The members of the next generation are generated as follows: We choose 2 parents with roulette selection [51], i.e., the probability of the members to be chosen is proportional to their fitness values which are the aggregate cost of that member + handicap for the members exceeding the error bound. This way, better members are more likely to be chosen for crossover. We apply single point crossover and obtain two child members, and the best two among the parents and children are selected for the next generation. We keep doing this until the next generation is fully generated. Furthermore, we applied one bit mutation after the crossover to increase diversity.

The feasible members in the population are the ones which satisfy the error bound. There are also unfeasible members which exceeds. These have a handicap in their fitness values which makes them less likely to be chosen for the next generation. We keep them in the population because they might lead to a very good solution with only very little changes. Besides, in some of the generations (especially the initial ones) there might not even be any feasible solutions. When the stopping condition is satisfied, the best feasible member having the least cost is selected as the best solution.

5.3. Greedy Heuristic 1: Equal Error Heuristic (EEH)

This algorithm is the fastest and the blindest one. The aim is to distribute the target error bound for the overall trajectory to its parts as equally as possible. We apply a recursive procedure of finding the best fitting representations and, if necessary, splitting the trajectory into smaller pieces. If the error bound is passed for a piece of the trajectory even if the best fitting representation is being used, then that piece of the trajectory is split into equal length smaller pieces so that it may become possible to approximate the smaller pieces within the error bound. Since the error bound is defined as a percentage, it is possible to evaluate each piece of the trajectory independently and test if the approximation for that piece of the trajectory stays within the overall error target. The intuition is that if each part of the trajectory is approximated with at most, for instance, 5% error, then the whole trajectory is assured to be approximated with at most 5% error.

First, we try to approximate the whole trajectory at once with the best possible representation. If the trajectory is above the error bound, we divide the trajectory into two or more pieces, where the number of pieces might be decided according to the error of that segment. For example, a segment with a very high error might be divided into more than 2 pieces at once. Every piece is equal in size on the *x*-axis. Then, we find the best representations to fit on each segment. We keep dividing the segments recursively that are over the error bound which is the same for all segments in percentage.

Fig. 8 shows an example application of the Equal Error Heuristic (EEH) on a sample trajectory. In (a), the heuristic first tries to find a representation for the whole ideal trajectory that stays within the error bound of 20%. It fails to do so, and then in (b), it splits the whole trajectory into two segments. Since both segments are under the error bound, the heuristic terminates with the approximate trajectory produced. If any of these two segments failed the error bound criteria, they would have been further split into two sub-segments as well.

Algorithm 1: Pseudocode for Equal Error Heuristic (EEH)

```
EqualError ($idealtrj, errorbound$)
 1: $trj[0] \leftarrow$ BestFit ($idealtrj.x_1, idealtrj.x_2, errorbound$)
    {An apprx. trajectory for the whole ideal trajectory}
 2: $n \leftarrow 1$
 3: $s \leftarrow 0$
 4: while $s < n$ do
 5:    if OffLimit ($trj[s], errorbound$) then {If over the
       error bound, divide into two sub-segments}
 6:       for $i \leftarrow n$ down to $s+1$ do {Shift segments
          right in $trj$}
 7:          $trj[i+1] \leftarrow trj[i]$
 8:       end for
 9:       $n \leftarrow n+1$
10:       $seg \leftarrow trj[s]$
11:       $trj[s] \leftarrow$ BestFit ($seg.x_1, (seg.x_1 + seg.x_2)/2, errorbound$) {Left sub-segment}
12:       $trj[s+1] \leftarrow$ BestFit ($(seg.x_1 + seg.x_2)/2, seg.x_2, errorbound$) {Right sub-segment}
13:    else
14:       $s \leftarrow s+1$
15:    end if
16: end while
```

Algorithm 1 shows a pseudocode for the iterative implementation of Equal Error Heuristic, where the EqualError heuristic function gets two parameters, the ideal trajectory (*idealtrj*) and the error bound in percentage (*errorbound*), and produces an approximate trajectory for the ideal trajectory under the error bound. *trj* is the approximate trajectory formed of segments while *n* is the number of segments in *trj*. At line 1, BestFit function returns the least cost representation under the error bound between $idealtrj.x_1$ and $idealtrj.x_2$ which are the *x* coordinates of the end points of the ideal trajectory. If there are no representation possible under the error bound, it returns the one with the least error. The loop at line 4 goes

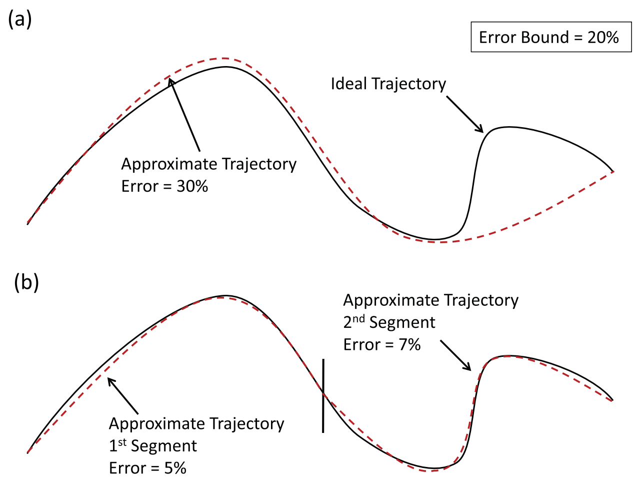


Fig. 8. Equal Error Heuristic divides the approximate trajectory into two pieces since approximation for the whole trajectory exceeds the error bound.

through all the segments in *trj*. Line 5 checks if a segment is over the error bound. If it is, then this segment is to be divided into two smaller segments. Lines 6–9 shift the segments in *trj* to open space for one more segment. Lines 10–12 divide the segment into two equal sub-segments in length on the *x*-axis and puts them in *trj*. Later they will be checked if they are over the error bound or not. Line 14 moves to the next segment if the current segment is under the error bound.

5.4. Greedy Heuristic 2: Longest Representation Heuristic (LRH)

This algorithm places the segments of the approximate trajectory one by one. The main idea is to choose the longest possible representation below the error bound for the next segment until we reach the end of the ideal trajectory. Starting from the shortest interval (in terms of number of split points) for the next segment, the interval is increased every time until we cannot find a suitable fit anymore. The last suitable one is chosen for that segment. Then, we try to find a representation for the segment starting from the end point of the last segment we have decided.

For example, in Fig. 9, for the first segment we initialize the interval to [0,5]. We assume that the step size is 5 for the sake of a clear figure. As long as we find a representation to fit into the current interval we increase it by 5, i.e., [0,10], [0,15] and [0,20]. When we realize that the representation in [0,20] exceeds the error bound, we store the representation in [0,15] as the first segment of the approximate trajectory. Then, we start from the end of the first segment and keep looking for the next segment in [15,20], [15,25], [15,30] etc. Since the segment in [15,35] is not suitable, the second segment is going to be the one in [15,30]. The algorithm runs until we reach to the end of the ideal trajectory. The step size of 5 for this example can be larger. This way the algorithm will run faster, but since the resolution will be reduced the algorithm will less likely to find better solutions. On the other hand, reducing it will make the algorithm run slower for better approximation.

Algorithm 2: Pseudocode for Longest Representation Heuristic (LRH)

```
LongestRepresentation ($idealtrj, errorbound$)
 1: $n \leftarrow 0$
 2: $last \leftarrow idealtrj.x_1$
 3: $seg \leftarrow$ BestFit ($idealtrj.x_1, idealtrj.x_1 + 1, errorbound$) {Init. first segment}
 4: for $x \leftarrow idealtrj.x_1 + 1$ to $idealtrj.x_2$ do
 5:   $oldseg \leftarrow seg$ {Store the last feasible segment}
 6:   $seg \leftarrow$ BestFit ($last, x, errorbound$) {Produce a longer segment}
 7:   if OffLimit ($seg, errorbound$) then {If over the error bound, use the previous one}
 8:     $trj[n] \leftarrow oldseg$
 9:     $n \leftarrow n + 1$
10:     $last \leftarrow x - 1$
11:     $seg \leftarrow$ BestFit ($last, x, errorbound$) {Init. next segment}
12:   end if
13: end for
14: $trj[n] \leftarrow seg$ {Add the last segment}
15: $n \leftarrow n + 1$
```

Algorithm 2 shows a pseudocode for the Longest Representation Heuristic (LRH), where LongestRepresentation function gets two parameters, the ideal trajectory (*idealtrj*) and the error bound in percentage (*errorbound*), and produces an approximate trajectory for the ideal trajectory under the error bound. *trj* is the approximate trajectory formed of segments while *n* is the number of segments in *trj.last* keeps the position of the end point of the last segment placed in *trj*. Lines 1–2 initialize *n* and *last*. Line 3 initializes *seg* to a line of length 1 on *x*-axis. This type of line is assumed to be always under the error bound. The loop starting at line 4 goes through the split points on the ideal trajectory from $idealtrj.x_1 + 1$ to $idealtrj.x_2$. Line 5 stores the last feasible segment in *oldseg*. Line 6 sets *seg* to the representation returned by the BestFit function which is the least cost representation under the error bound between *last* and *x*. If there is no representation possible under the error bound, the function returns the one with the least error. Line 7 checks if *seg* is over the error bound or not. If so, lines 8–9 place the *oldseg* into *trj* and lines 10–11 produce a segment for the interval right after *oldseg*. Finally, lines 14–15 place the last segment into *trj* which was ignored in the loop.

5.5. Analysis

The proposed algorithms have three parameters affecting their time complexity as well as solution quality: (i) *N*, the number of possible split points (a.k.a. the resolution at which our approximation algorithms work within), (ii) *E*, the error bound given by the application, (iii) α, the complexity of the ideal trajectory, and (iv) *d*, the distance in pixels between two consecutive split points on the *x*-axis. *Nd* gives a measure of the length of the ideal trajectory.

5.5.1. Resolution difference

We first examine and formalize the resolution difference and its effects on the algorithms. The distance be-

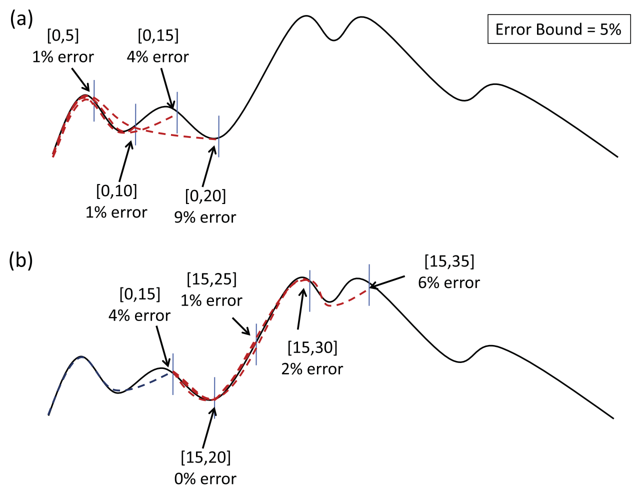


Fig. 9. (a) Longest Representation Heuristic tries to find the longest representation under the error bound starting from the left most point. (b) After the first segment is set, it looks for the next segment starting right after.

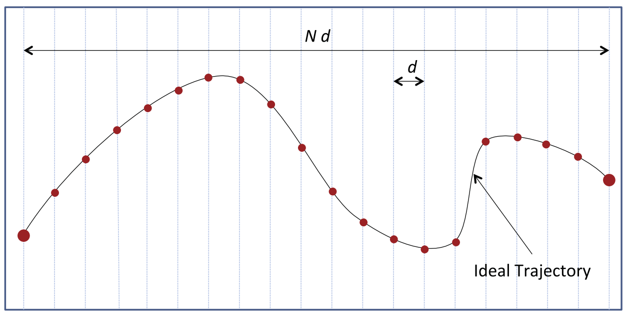


Fig. 10. The resolution difference between the ideal trajectory and the space where the approximate trajectory is computed.

tween two consecutive split points, d, is measured on the resolution of the space where the ideal trajectory is defined. For an ideal trajectory spanning $N-1$ split points, the total x-axis length will be Nd. This resolution difference is illustrated in Fig. 10.

The effect of this resolution difference can be formalized if the ideal trajectory complexity, α, is taken into account. Assuming that the ideal trajectory's derivative on the y-axis is bounded by $\tan\alpha$, we formulate the maximum possible error between two consecutive split points, as shown in Fig. 11. In order to sketch a scenario where the maximum possible error takes place between the ideal and the approximate trajectories, we should be using the simplest representation, a line, between the split points S_i and S_{i+1}. Since the derivative on the y-axis is bounded by $\tan\alpha$, the largest area between the ideal and the approximate trajectories can happen only when the ideal trajectory diverges by the longest possible vertical distance and joins back to the other end of the line where the next split point resides. The ideal trajectory should reach the farthest vertical point by the middle of the line on the approximate trajectory, so that it can travel back to the line before reaching the next split point. Then, the longest vertical divergence will be:

$$B = \frac{d}{2}\tan\alpha. \tag{5}$$

Lemma 1. *The maximum approximation error between two consecutive split points, e^{max}, is $O(d^2\tan\alpha)$, when the derivative of the ideal trajectory on the y-axis is bounded by $\tan\alpha$.*

Proof. Depending on the derivative characteristics of the ideal trajectory on the x-axis, the largest error (i.e. the area between the ideal and the approximate trajectories) is bounded by the rectangle with edges d and B. Since such a rectangle would be unrealistic, we assume that the ideal trajectory is elliptic.[1] The area of such an half ellipse is πAB, which evaluates to

$$e^{max} = \pi\frac{d^2}{4}\tan\alpha \tag{6}$$

as can be observed from Fig. 11. $\square$

Theorem 1. *The maximum approximation error, E^{max}, for an ideal trajectory spanning $N-1$ split points is $O(N^2d^2\tan\alpha)$, when the derivative of the ideal trajectory on the y-axis is bounded by $\tan\alpha$.*

Proof. The proof is an extension of Lemma 1. When the whole ideal trajectory is considered, the largest possible area between the ideal and the approximate trajectories can be modeled as a scaled up version of the half ellipse shown in Fig. 11. The length of the straight line from the source to the destination of the ideal trajectory will be Nd. This makes the area of the half ellipse as

$$E^{max} = \pi\frac{(Nd)^2}{4}\tan\alpha \tag{7}$$

which is $O(N^2d^2\tan\alpha)$. $\square$

The approximation error stated by Theorem 1 does not indicate the performance with respect to the optimum solution to our trajectory approximation under application-based accuracy constraints. Rather, the theorem gives insight about how close the approximate trajectory can be away from the ideal trajectory. Depending on the application-specific accuracy constraint E and the representation costs of the individual segments of the approximate trajectory, the optimum solution can be such that the representation cost (i.e. routing state) is very small but the approximation error is very high since the application allows such high approximation errors by defining a large E.

5.5.2. Time complexity

Intuitively, N is directly related to the running time of the algorithms. However, the effect of E and α to the running times is probabilistic and not intuitive.

5.5.2.1. Exhaustive search. The exhaustive method gives two different values for each of the possible split points and tests the solution. This brute force algorithm is obviously running in $O(2^N)$. However, the pruning for the algorithm is what makes it tractable for the problem and reduces the running time dramatically. The possibility of pruning happening is dependent on the values of E and α.

[1] The rest of the analysis works in the same manner if we had chosen a rectangle. The conjectures out of the analysis do not change because of the choice of elliptic curve.

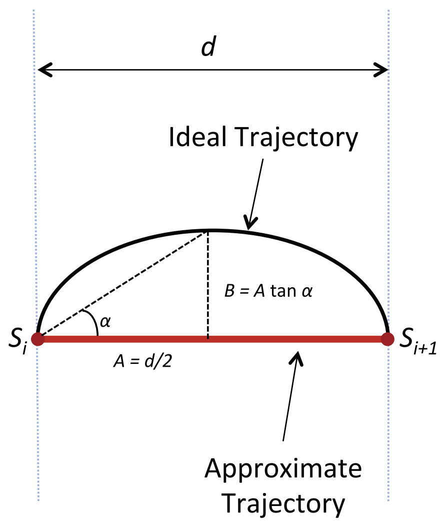


Fig. 11. Maximum error between two split points when the ideal trajectory's complexity is bounded with the angle α.

Although we experimentally observed that pruning helps the running time significantly, the worst case time complexity for exhaustive search is $O(2^N)$.

5.5.2.2. Genetic algorithm. The GA's parameters are mostly constants. The number of elements in the pool and the number of generations are constant values. However, the length of a chromosome is $3N + 2$. Since all the operations on the chromosomes (i.e., the crossover and generation of an approximate trajectory) are linear, the time complexity of GA is $O(N)$.

5.5.2.3. Greedy heuristics. Our heuristics aim to find a good solution within a very short period of time. So, the time complexity is crucial for their design.

Theorem 2. *LRH running time is O(N).*

Proof. The LRH algorithm inspects each split point by calculating the error and cost representing the segments of the end-to-end trajectory. For each split point, it calculates the error and the cost from the source node. The trajectory complexity α or the error bound E do not affect the running time of the LRH algorithm. This is illustrated by loop at lines 4–13 of Algorithm 2, which iterates N times regardless of the error bound E or the trajectory complexity α. Thus, the running time is of $O(N)$ for LRH. □

The EEH algorithm looks at a segment of the ideal trajectory and recursively decides if it needs to divide the segment further or not based on the approximation error of the segment. Assuming that the finest points where EEH can divide are the split points, the worst case scenario is when EEH divides at every possible split point. This intuitively gives a running time complexity of $O(N\log N)$. However, more stringent bounds on the time complexity of EEH are possible since the probability that EEH will divide its segment is less than 1.

Lemma 2. *The probability that EEH will divide the ideal trajectory segment at its ith step, ρ_i, is dependent on the ratio $\Gamma = E/E^{max}$.*

Proof. The EEH algorithm tries to distribute the application-specific error bound, E, equally across the approximate trajectory. So, if it needs to divide an ideal trajectory segment at its ith step into smaller segments, it also divides the error bound to allocate the error to the smaller segments. Assuming that EEH divides by two at each step, the number of recursive steps (i.e., levels) will be $\log_2 N$ and the error bound for a segment at the ith step of the algorithm will be:

$$E_i = \frac{E}{2^i}, \quad i = 0 \ldots \log_2 N. \tag{8}$$

Let e_i be the random variable for the actual amount of error between the ideal trajectory segment and the approximate trajectory at the ith step of EEH. If e_i is greater than E_i, then EEH needs to divide the segment further into two at the ith step. We, then, write the probability that EEH will divide the segment in two smaller parts at the ith step as:

$$\rho_i = P[e_i > E_i] = 1 - P[e_i \leqslant E_i] = 1 - \int_0^{E_i} f(e_i)de_i, \tag{9}$$

where $f(e_i)$ is the probability density function for e_i.

For an ideal trajectory with an x-axis length of Nd, the x-axis length of the segment at the ith step will be $Nd/2^i$, $i = 0,\ldots,\log_2 N$. Revising (7) with $Nd/2^i$ as the base of the half-ellipse, the maximum error from the ideal trajectory segment at the ith step will be:

$$E_i^{max} = \frac{\pi}{4}\left(\frac{Nd}{2^i}\right)^2 \tan\alpha = \frac{E^{max}}{2^{2i}}, \quad i = 0 \ldots \log_2 N. \tag{10}$$

This means that e_i is in the range $[0, E_i^{max}]$. Assuming that $f(e_i)$ is Uniform in $[0, E_i^{max}]$ and $E_i \leqslant E_i^{max}$, ρ_i evaluates to the following:

$$\rho_i = 1 - 2^i \frac{E}{E^{max}}, \quad i = 0 \ldots \log_2 N, \tag{11}$$

which proves that ρ_i is directly dependent on the ratio $\Gamma = E/E^{max}$ in addition to i. □

The assumption that $E_i \leqslant E_i^{max}$ is conservative and does not invalidate the analysis for the worst case running time as ρ_i is supposed to be zero for the cases when $E_i > E_i^{max}$. Further, the assumption of Uniform distribution of the error e_i makes the worst-case analysis conservative as well since any other distribution with a skew towards smaller error values would result in a lower ρ_i.

Lemma 2 verifies a few intuitions. It is clear from (11) that the probability of dividing into smaller segments tends to 1 either the error bound E or the ideal trajectory complexity α increases. More specifically, $\lim_{E\to 0}\rho_i = 1$ and $\lim_{\alpha\to\pi/2}\rho_i = 1$. Furthermore, larger Γ causes ρ_i to reduce, which means smaller running time complexity for EEH. The ratio Γ expresses how the error bound, E, and the amount of possible unit error due to the complexity of the ideal trajectory, E^{max}, stand with each other. A large E may not always result in a larger ρ_i, or vice versa. Rather, the ratio between E and E^{max} determines the behavior of ρ_i.

Lemma 3. *The number of computations at the ith recursive step of EEH is* $\phi(i) = 1 + 2\rho_i\phi(i + 1)$.

Proof. Given the probability that EEH divides the segment into two at its i recursive step, ρ_i, from Lemma 2, it becomes possible to formulate the number of computations that needs to take place at the ith recursive step, $\phi(i)$, as a recurrence relation. First, at each step, there will be one computation to evaluate if EEH needs to divide the segment further down or not. If EEH does need to divide the segment further, then it will embark two recursive branches down, hence the factor of 2. This recursive relationship intuitively, then, becomes:

$$\phi(i) = 1 + 2\rho_i\phi(i+1), \quad i = 0 \ldots \log_2 N. \qquad \square \tag{12}$$

Theorem 3. *EEH running time is* $O(N\log N - \gamma)$*, where* $\gamma = E/e^{max}$.

Proof. Solving the recurrence relation (12), the number of computations EEH will perform, Φ, can be derived as:

$$\Phi = 1 + \sum_{i=0}^{\log_2 N} 2^{i+1} \prod_{j=0}^{i} \rho_j, \tag{13}$$

Conservatively assuming that the product of probabilities above is a sum, Φ evaluates to:

$$\Phi = 3 + N\log_2 N - \Gamma\left[\frac{16N^2}{3} - 4N + \frac{2}{3}\right], \tag{14}$$

$$= 3 + N\log_2 N - \gamma\left[\frac{16}{3} - \frac{4}{N} + \frac{2}{3N^2}\right], \tag{15}$$

which clearly indicates that EEH running time is bounded by $O(N\log N - \gamma)$. $\square$

Similar to Γ, the ratio γ is a reverse-indicator of the difficulty of the trajectory approximation problem. Larger γ means that the application allows larger error bound with respect to the unit approximation error, e^{max}, and that the difficulty of the problem decreases. So, Theorem 3 verifies the intuition that EEH running time reduces as the problem's difficulty decreases. The theorem also clearly shows that EEH running time is strictly bounded $O(N\log N)$ no matter how difficult the problem is. That is $\lim_{E\to 0}\Phi = N\log_2 N$ and $\lim_{\alpha\to\pi/2}\Phi = N\log_2 N$.

Theorem 4. *EEH running time is* $O(1)$ *if* $\Gamma \geqslant 1$.

Proof. The EEH running time is $O(1)$ if the inequality $\Phi \leqslant 1$ satisfies. Substituting (14) for Φ and solving for Γ, we obtain:

$$\Gamma \geqslant \frac{12N\log_2 N + 6}{16N^2 - 12N + 2}. \tag{16}$$

The right hand side (RHS) of (16) goes to 0 as N tends to ∞. Further, the RHS is less than or equal to 1 as long as $N \geqslant 1$, which is always true as there has to be at least one split point for the algorithm to work. This completes the proof. $\square$

The result of Theorem 4 is that the running time complexity of EEH will not be dependent on N if the application-defined error bound E is greater than or equal to the maximum possible approximation error that can be caused by the complexity of the ideal trajectory, E^{max}. That is, if the application is allowing $\Gamma = 100\%$ approximation error, then EEH runs with $O(1)$. More aggressive numbers can be obtained from the RHS of (16) when a minimum value for N is considered. Table 1 outlines such cases where minimum *Gamma* and N values are calculated to assure $O(1)$ complexity for EEH. The table shows that EEH needs to work with a resolution of 101 split points (i.e., it needs to be able to divide the ideal trajectory at 101 different points) to find a solution in $O(1)$ time if the application is given the ability to define an approximation error bound as little as 5%.

6. Performance evaluation

We performed performance evaluation of the four algorithms from Section 5 by applying them on several trajectories with varying complexity. The goal of our experiments is twofold:

- *Algorithm performance comparison:* Observe performance of our heuristics and the genetic algorithm in terms of quality of their trajectory approximation and their running time.
- *Path customization to different networks:* Illustrate that our optimization framework can be customized for a power-scarce or memory-scarce network by changing the weight of the packet header cost in the aggregate routing state cost.

6.1. Experimental setup

In our experiments, we used a 400×400 pixel area, where the trajectories are placed. The source node of a trajectory is at a random place on the y-axis with $x = 0$. Similarly, the destination node is at a random place on the $x = 400$ line. The trajectory between the source and destination nodes is made up of small line segments, concatenated to construct the complete trajectory. We parameterized the trajectory generation procedure in terms of how variant the trajectory should be. We call this parameter as the *complexity* of the trajectory which can take values in [0,180]. The complexity parameter defines the maximum angle value (in "degrees") between each pair of consecutive line segments. The higher this parameter gets the more complex (zigzag) the trajectory will be. If

Table 1
Various conditions satisfying $O(1)$ complexity for EEH.

Minimum Γ	Minimum N
1.00	1.0
0.50	3.8
0.25	11.4
0.10	41.0
0.05	100.6
0.01	711.4

this parameter is set to 0 then the angle between the line segments will be 0, which will produce a straight line.

In the experiments we have tried different error bound and trajectory complexity parameters. We used 4 different error bound values (i.e., 5, 10, 25 and 50) and 18 different trajectory complexity values (i.e., from 10 to 180 increasing with a step of 10). For each error bound and trajectory complexity pair, we produced 32 different trajectories with random seeds (which are used to randomize the angle between two consecutive line segments of a trajectory) and applied the four trajectory approximation algorithms on each of them. We report show 90% confidence intervals on our results over these 32 trajectories. We now present the rest of our experimental setup specific to each of the four algorithms.

6.1.1. Exhaustive search

In the exhaustive search, we set the number of possible split points to 19. So, the maximum number of segments that the trajectory can have is 20, which means that the shortest segment can be 20 pixels wide. We could not increase the number of split points because of the time limitation. Even with this set up it sometimes took many hours just for one run.

6.1.2. Genetic algorithm (GA)

First thing to note about GA is the fact that it includes a few random factor, unlike the other three algorithms. So, we ran our GA 32 times on each trajectory and report the average result of those 32 realizations. The other three algorithms do not need this since they do not have a random factor.

We used the following parameters to tune our GA implementation:

- Population size: 300.
- Crossover probability: 0.99.
- Mutation probability: 1.

These are values that we found out after running many experiments and believe are the near optimum for our problem. The mutation probability we found to be the best is surprisingly high, which yields more diverse solutions faster (i.e. better exploration of the search space). To make our GA implementation comparable with the exhaustive search method, we used 19 possible split points. However, as will be seen in the results, it is possible to use a lot more split points (e.g., 200) and get good solutions in a very short time unlike the exhaustive search.

We have observed the GA results for different stopping conditions. We ran the experiments for up to 20, 50, 100, 200, 300, 1000, and 10000 generations. In most of the cases the GA finds the best solution before 50 generations. After 100 generations we observed that almost nothing changes. So, we decided to stop a GA run after 300 generations.

6.1.3. Greedy Heuristic 1: Equal Error

We divided the segments into 2 pieces every time and used the representation which gives the smallest error for the corresponding piece of the trajectory.

6.1.4. Greedy Heuristic 2: Longest Representation

We set the step size to 1 to get the best result. Since, running time is not that much of an issue for this heuristic, we kept the step size small. For long or complex trajectories, one might want to increase the step size to reduce the computation time, though with a loss in optimality of the path approximation.

6.2. Results and Discussion

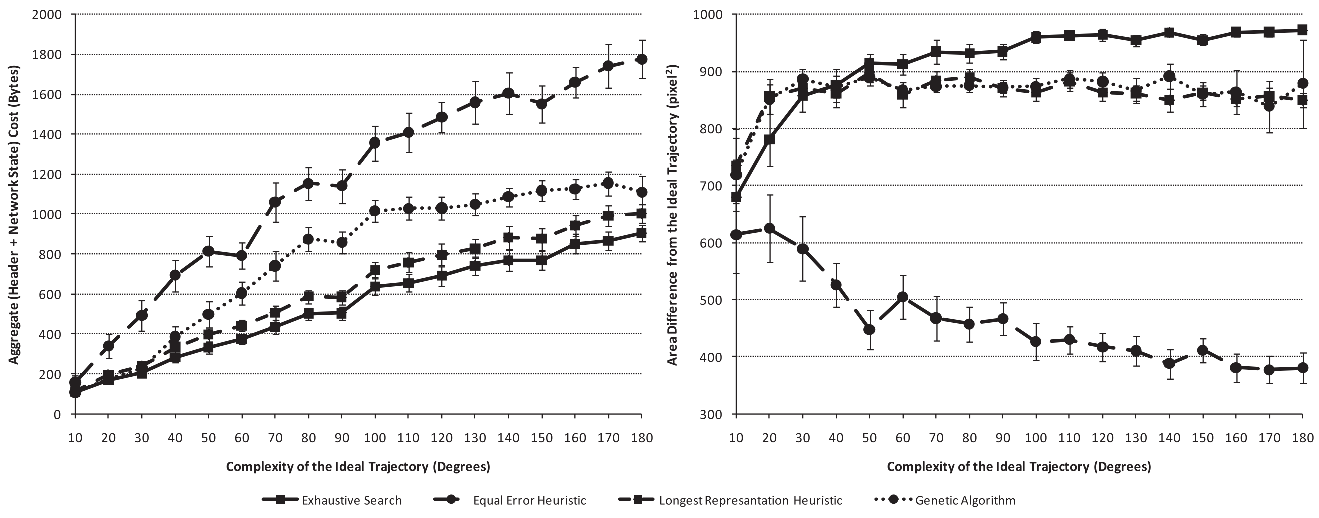
6.2.1. Comparison of Exhaustive Search, GA, and Heuristics

We compared the algorithms in three measures: the aggregate cost, the area difference from the ideal trajectory, and the running (computation) time. We present the results for error bounds 5%, 25% and 50% only as the results for other error bounds exhibit similar trends. We ran the experiments for a resolution of 20 split points over the 400×400 pixel area, which means that the approximate trajectory can at most have 19 segments.

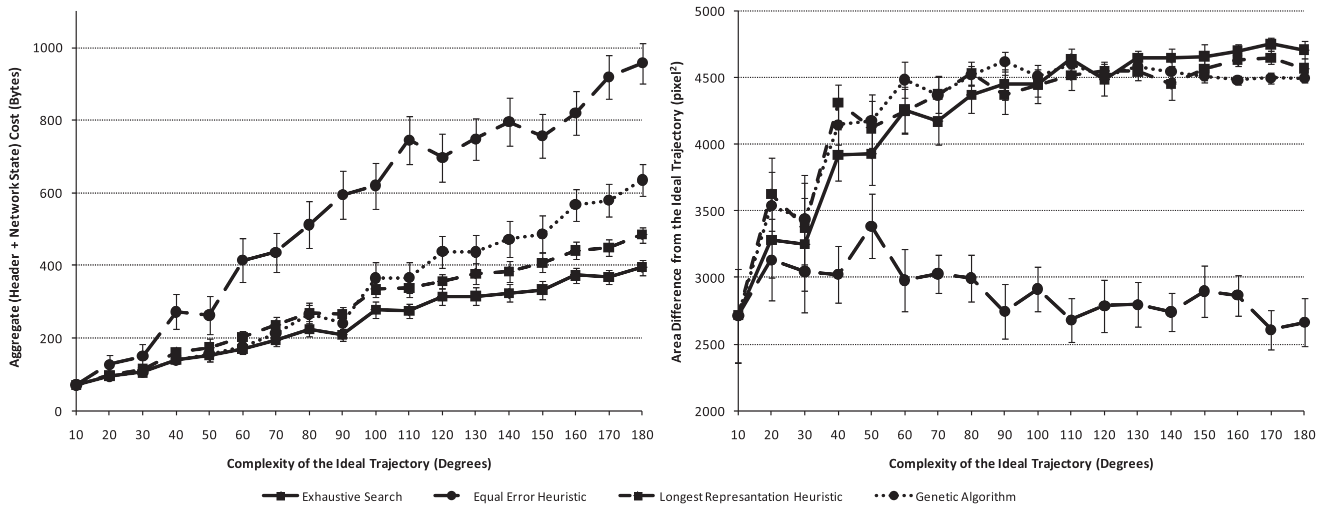
In Fig. 12, the plots at the left column show the aggregate cost for trajectories with different complexity values. We see that the Exhaustive Search (ES), genetic algorithm (GA) and the Longest Representation Heuristic (LRH) give similar results. We know that, for the given resolution, ES finds the optimum solution with the smallest cost under the given error bound. Since GA and LRH have the same resolution with ES, we can say from the graphs that they perform pretty well. Our LRH performs actually slightly better than GA (in Fig. 12(a) and (b)), particularly for larger complexity in the ideal trajectory. However, the Equal Error Heuristic (EEH) performs noticeably worse than the other three algorithms, with emphasized disadvantage as the complexity increases.

By comparing Fig. 12(a)–(c), a clear trend we can see, for all algorithms, is that *increased complexity of the ideal trajectory increases the cost of the approximated trajectory*. The reason is that the optimization process has to split the ideal trajectory to smaller and more number of segments in order to be able to represent each segment with a smaller cost representation. This causes increased number of segments with similar complexity, and thus an increased total cost for the approximating the ideal trajectory. Further, tighter error bound (determined by the application) clearly worsens the optimum solution to the trajectory approximation problem, i.e. *decreased error bound increases the cost of the approximated trajectory*. This is due to the fact that smaller error bound forces complexity of segment representations to be higher, which then increases the aggregate cost.

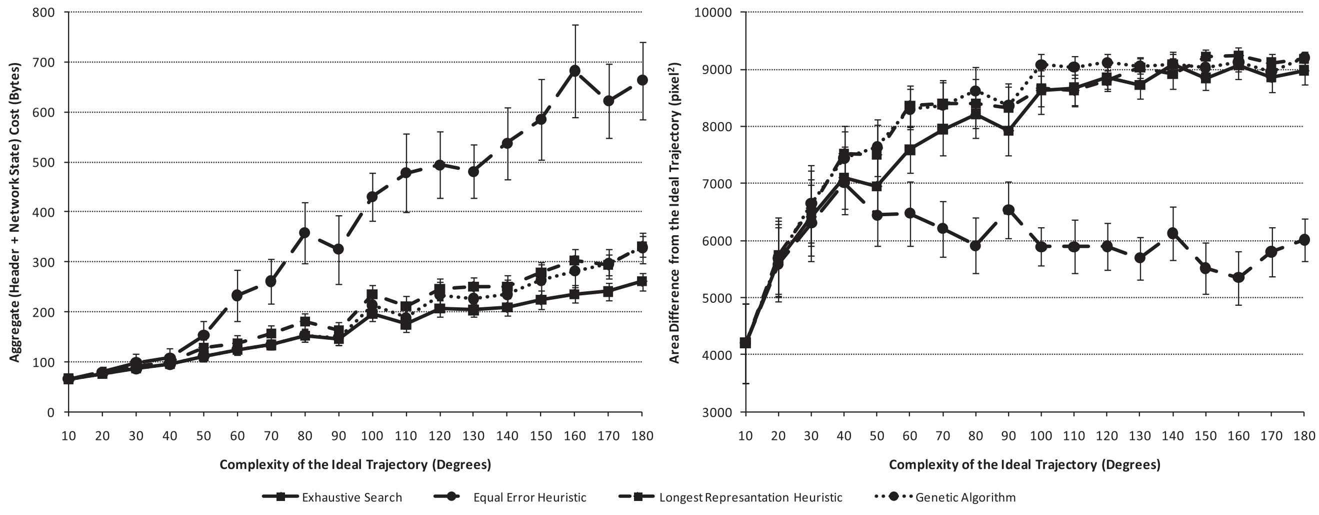
The graphs at the right column of Fig. 12 show the deviation area (in pixel^2) of the approximate trajectory, i.e. the area between the approximate and the ideal trajectory. The deviation area is actually showing the approximate trajectory's error from the ideal trajectory. Thus, it is a measure of how close the approximate trajectory is to the ideal one. Remember that the trajectory approximation problem is not to minimize the deviation area, but to minimize the cost of representing the approximate trajectory which is deviating from the ideal trajectory up to an error bound defined by the application. So, the optimum solution to the trajectory approximation should, intuitively, lie in the re-



(a) Error Bound 5%



(b) Error Bound 25%



(c) Error Bound 50%

Fig. 12. Results of trajectory approximation: Left – aggregate routing state cost vs. Trajectory complexity; Right – The area difference from the ideal trajectory vs. Trajectory complexity.

gion where the approximate trajectory deviates from the ideal one with a deviation error as close as possible to the error bound. The deviation area results in Fig. 12 verifies this intuition since all the three well performing algorithms (i.e., ES, GA, LRH) yield high deviation areas while the worst (in terms of the cost) algorithm (i.e. EEH) yield

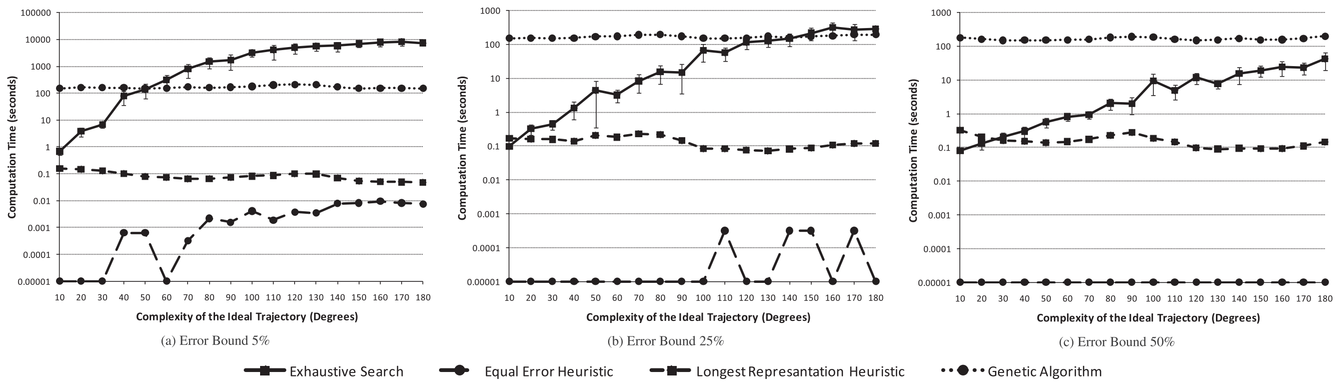


Fig. 13. Results of trajectory approximation: computation time vs. trajectory complexity.

low deviation areas. The difference is noticeable, of course, only when the complexity of the ideal trajectory is high.

Lastly, Fig. 13 shows the running time of the algorithms for different error bounds. It is clear that ES is not practical with several seconds of running time, though it finds the optimum solution in terms of routing state cost. Under 5% error, the ES takes several hours to run, particularly for cases with high complexity in the ideal trajectory. The other extreme is EEH, which runs in almost no time. EEH may not be a suitable choice because its performance (in terms of routing state cost) is too low. However, it might be a viable choice where time is very crucial and routing state cost is not much of an issue. The other heuristic, LRH, runs in a very short period of time, mostly under 100 ms. LRH achieves very good (sometimes better than GA) routing state costs too. So, LRH seems to give best results for the time spent for computation. With the current experimental set up, GA takes about 100s since it is running 300 generations regardless of the results found. It is possible to tune the GA such that smaller amount of time spent for computation with an increase in the routing state costs. Such a well-tuned GA can be quite useful for the initial approximation of the trajectory. Overall, the LRH algorithm runs in reasonably short time, while giving pretty good performance in minimizing the routing state cost.

6.2.2. Path customization to power-scarce networks

Being able to customize the end-to-end path for different environments is a key capability. We evaluated our optimization framework to observe if it can generate customized paths depending on how the packet header and the network state costs are calculated. We ran experiments for a customized objective (1) of cost and observed how the solution is adjusted to the new definitions. Specifically, we focused on the tradeoff between packet header cost and network state cost. It is well-known that data transmission consumes larger power than storing the data [52,53]. Thus, we aim to customize our framework such that the approximate trajectory is calculated while trying to reduce the amount of packet header state. We include the length of the trajectory segments the packets have to traverse as a weighting factor in our objective (1).

This means that we have two cases to compare in calculating our routing state cost objective: *length independent calculation* and *length dependent calculation*. The length independent calculation is what we used in the above experiments, i.e., for every segment of the approximate trajectory, the packet header cost is the same as the base cost of the representation no matter what the length of that segment is. If it is a straight line then $C_P = 32$ bytes. In the length dependent calculation, we assume that for long segments of trajectories the packets have to go through more than one node and we calculate cost for every node passed through. This per segment cost is, then, dependent to the density of the nodes, which we assumed equal to the resolution of our trajectory approximation. For example, for the resolution of 20 split points, we assumed that for every 20pixels the packets visit another node. This means that for a 2nd degree curve which is 148 pixels long, 8 nodes will be passed, and so C_P will be $32 \times 8 = 256$ bytes.

Comparing the length dependent vs. independent cases, we observed the number of segments in the approximate trajectory and the average complexity of the representations used. To obtain finer granularity results, we increased the resolution of the approximation problem to 100 split points, which means that the approximate trajectories can have up to 99 segments at maximum. Since the ES is intractable for such a high resolution,[2] we used our GA implementation to calculate the approximate trajectories. To make sure that the GA finds good solutions to the trajectory approximation at this high resolution, we increased its run time to 5000 generations.

For ideal trajectories with different complexities and different error bounds, we measured and plotted the number of segments as well as the complexity of segment representations (in terms of the number of bytes used for the representation) in the approximate trajectories, all of which are shown in Fig. 14. We see that whatever the error bound is, the length dependent calculation (the dashed lines) causes the trajectories to have more segments and be formed by less complex representations. The reason is that, in this calculation the weight of packet header cost is increased and the optimization framework tends to reduce the packet header cost by increasing the network state cost. As an example, in Fig. 15, the left picture shows the approximate trajectory generated with length indepen-

[2] Even a single run of the trajectory approximation using ES takes weeks to complete at this resolution.

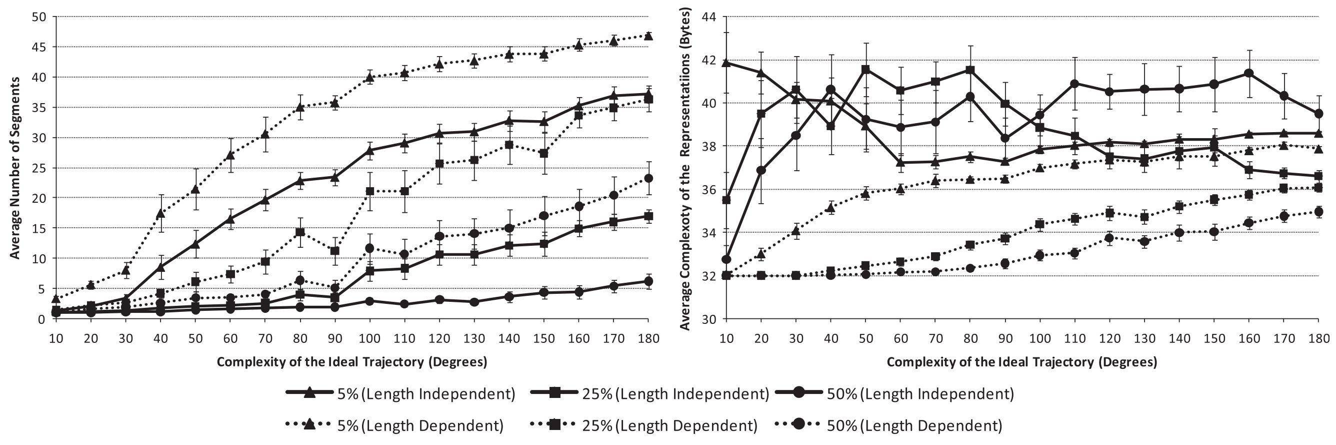


Fig. 14. Customization of the routing state costs for a power-scarce network. Cost of packet header state can be made dependent on the length of trajectory piece the packets have to travel.

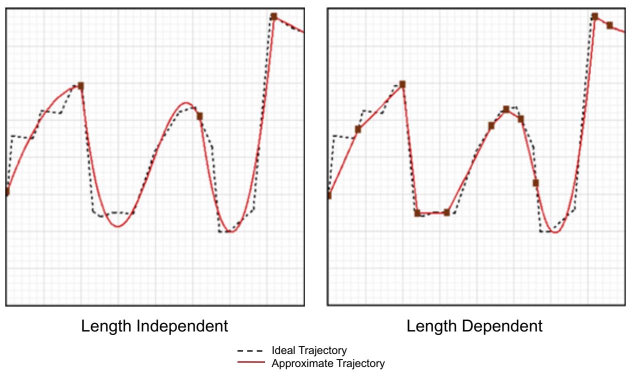


Fig. 15. Sample runs for different cost calculations. Trajectory complexity: 120, error bound: 25%.

dent calculations. It is made up of 3 curves and 1 line. On the other hand, the approximate trajectory generated with the length dependent calculations, on the right, has more segments and most of them are lines. This clearly shows that it is possible to customize our trajectory approximation framework such that the approximate trajectory generated by the optimization process is tuned for different resource constraints.

We, now, test if our path approximation methodologies can "effectively" customize the solution for different *weights* of packet header state and network state costs. Notice that such weight assignment is not the same as the length dependent or independent calculations we explored in the previous subsection. The choice of length dependent or independent was a binary parameter; however, these weights are much more expressive parameters to the optimization process.

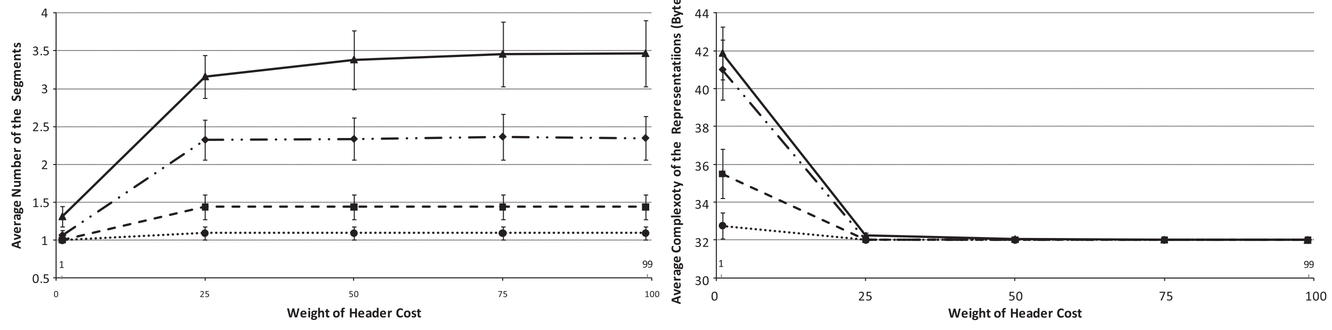
The weight setting capability allows the applications to feed these weights depending on the circumstances where the routing is taking place. If, for example, the network components are running on highly limited power, then the application could assign a larger weight value to the packet header cost in (1). Likewise, if the network components do not have enough memory to store routing state information, then the application could give a larger weight value to the network state cost.

Using a resolution of 200 split points, we ran our GA optimizations (with 5000 generations) for different weight values and observed if the number of segments per trajectory and the complexity of representations in a trajectory change based on the weights. We particularly want to see if it is possible to customize the characteristics of the approximated trajectory by simply changing one parameter in the optimization formulation, which is the *weight of the packet header cost*. We tried the weight settings of (1,99), (50,50), and (99,1), where each pair's first and second elements correspond to the packet header the network state costs respectively. Fig. 16 shows the results of these experiments for three different complexity values of the ideal trajectory.

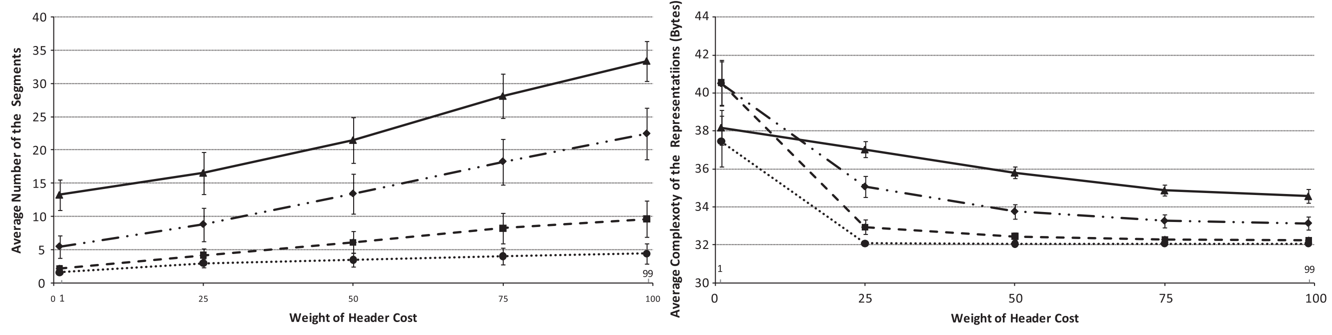
Overall, we can observe the trends that the number of segments in the approximate trajectory increases as the weight of the packet header cost increases. Likewise, increase in the weight of the packet header cost causes the complexity of the segment representations to reduce. Though the negative correlation between the complexity of the segments and the weight of the packet header cost is not that apparent, the positive correlation between the number of segments and the weight of the packet header cost is considerably apparent. This is more noticeable in the cases with smaller error bounds and larger complexity in the ideal trajectories. Smaller error bounds force the approximation process to search in a narrower set of possible solutions where finding the tradeoff between the number of segments and the complexity of the segment representations become a critical one. Similarly, higher complexity in the ideal trajectory narrows the search space for solutions where this tradeoff is emphasized.

7. Summary and future work

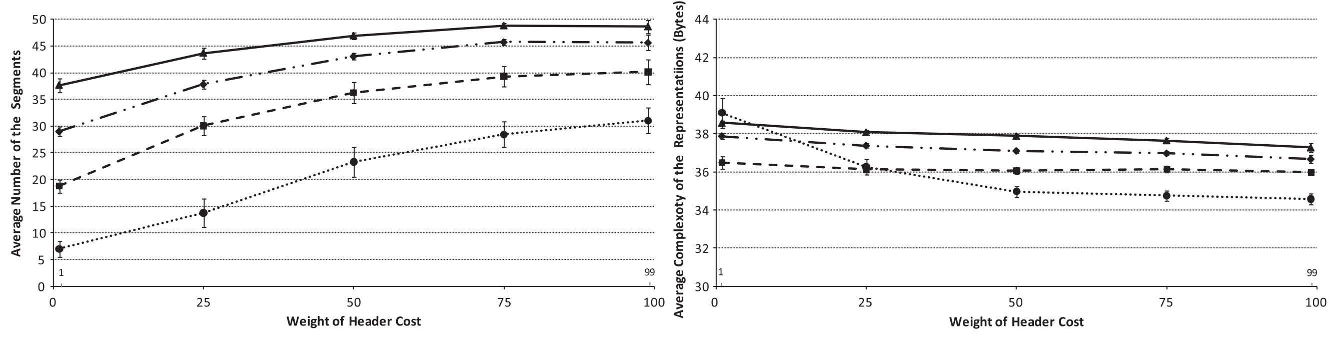
In this paper we presented an optimization framework minimizing routing state under application-based constraints. We formulated the problem of generating an approximate trajectory within an application-defined error bound such that the total state cost of performing Trajectory-Based Routing over a multi-hop wireless network minimized. We showed that this approximation problem



(a) Complexity of the ideal trajectory: 10 degrees



(b) Complexity of the ideal trajectory: 50 degrees



(c) Complexity of the ideal trajectory: 180 degrees

▲ Error 5% ◆ Error 10% ■ Error 25% ● Error 50%

Fig. 16. Path customization: Correlation between the number and complexity of segments in the approximate trajectory and the weight of header cost.

is NP-complete and hard to solve with regular brute force methods. To solve the problem, we devised four algorithms each having its advantages and disadvantages, and compared their performance and applicability in a multi-hop wireless network. We also showed that our problem is customizable for different contexts such as a power-scarce sensor network or memory-scarce network.

The work in this paper is only one important part of the whole framework, and many new dimensions and extensions can be performed as future work. For instance, inclusion of these path approximation methodologies into the larger framework with trajectory-based forwarding under a real user application is of high interest and deserves a full inspection. The whole framework can be tested where there are more than one connection at a time and node mobility exists. Considering the trajectory approximation problem, there are still points to be improved as well. New representations like B-spline and Bézier curves can be used for more flexible and optimum approximations.

Another key future work is to explore the cases where it is possible to tackle all stages of our approach at once. In our formulation, we focused on the trajectory approximation problem with the assumption that an ideal trajectory is available and the underlying network is reasonably well populated to achieve forwarding very close to the approximate trajectory generated by our methods. The actual positions of the nodes will change the optimum solution to the trajectory approximation problem. However, such treatment of the trajectory approximation problem necessitates the node positions to be stationary (or very close to stationary) and available at the source node. Considering a network with sizable dynamism in its topology (and the node positions), we chose to stage the problem into three parts (see Fig. 1): (i) definition of the ideal trajectory, (ii) trajectory approximation, and (iii) trajectory-based forwarding (TBF). It is of high interest to integrate these stages and solve them jointly for cases where they may be applicable.

References

[1] M.O. Kilavuz, M. Yuksel, Minimizing multi-hop wireless routing state under application-based accuracy constraints, in: Proceedings of IEEE MASS, Atlanta, GA, September 2008.

[2] B.T. Loo, J.M. Hellerstein, I. Stoica, R. Ramakrishnan, Declarative routing: Extensible routing with declarative queries, in: Proceedings of ACM SIGCOMM, 2005.

[3] D. Chu, L. Popa, A. Tavakoli, J.M. Hellerstein, P. Levis, S. Shenker, I. Stoica, The design and implementation of a declarative sensor network system, in: Proceedings of ACM Conference on Embedded Networked Sensor Systems (SenSys), 2007.

[4] D. Niculescu, B. Nath, Trajectory based forwarding and its applications, in: Proceedings of ACM MOBICOM, 2003.

[5] M. Yuksel, R. Pradhan, S. Kalyanaraman, An implementation framework for trajectory-based forwarding in ad-hoc networks, Ad Hoc Networks, Elsevier Science 4 (1) (2006) 125–137.

[6] B. Karp, H. Kung, GPSR: Greedy perimeter stateless routing for wireless networks, in Proceedings of ACM/IEEE MobiCom 2000, August 2000.

[7] L. Popa, A. Rostami, R. Karp, C. Papadimitriou, I. Stoica, Balancing the traffic load in wireless networks with curveball routing, in Proceedings of ACM MOBIHOC, 2007.

[8] G. Veltri, Q. Huang, G. Qu, M. Potkonjak, Minimal and maximal exposure path algorithms for wireless embedded sensor networks, in: Proceedings of ACM SenSys, November 2003.

[9] V. Muthusamy, M. Petrovic, H.-A. Jacobsen, Effects of routing computations in content-based routing networks with mobile data sources, in: Proceedings of ACM MOBICOM, 2005.

[10] D.B. Johnson, D.A. Maltz, Dynamic source routing in ad hoc wireless networks, in: Imielinski, Korth (Eds.), Mobile Computing, vol. 353, Kluwer Academic Publishers., 1996.

[11] C. Perkins, E. Belding-Royer, S. Das, Ad hoc on-demand distance vector (aodv) routing, IETF RFC 3561, July 2003.

[12] P. Sinha, S. Krishnamurthy, S. Dao, Scalable unidirectional routing with zone routing protocol, in: Proceedings of Wireless Communications and Networking Conference (WCNC), 2000, pp. 1329–1339.

[13] S. Basagni, I. Chlamtac, V. Syrotiuk, B. Woodward, A distance routing effect algorithm for mobility (DREAM), in: Proceedings of ACM/IEEE MobiCom 98, October 1998, pp. 76–84.

[14] J. Li, J. Jannotti, D. De Couto, D. Karger, R. Morris, ‘‘A scalable location service for geographic ad-hoc routing, in: Proceedings of the 6th ACM International Conference on Mobile Computing and Networking, August 2000, pp. 120–130.

[15] C.E. Perkins, P. Bhagwat, Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers, ACM SIGCOMM CCR (1994).

[16] G. Pei, M. Gerla, T.-W. Chen, Fisheye state routing in mobile ad hoc networks, in Proceedings of Workshop on Wireless Networks and Mobile Computing, 2000, pp. D71–D78.

[17] I. Stoica, R. Morris, D. Karger, F. Kaashoek, H. Balakrishnan, Chord: A scalable Peer-To-Peer lookup service for internet applications, in: Proceedings of the 2001 ACM SIGCOMM Conference, 2001, pp. 149–160.

[18] C. Santivanez, R. Ramanathan, Hazy sighted link state (hsls) routing: A scalable link state algorithm, BBN Technologies, Cambridge, MA, Tech. Rep. BBN-TM-1301, 2001.

[19] G.B. Folland, Real analysis: Modern Techniques and Their Applications, Wiley, New York, 1999.

[20] Tropos debuts spectrum and application based routing engine for wireless mesh networks, <http://www.convergedigest.com/Wireless/broadbandwirelessarticle.asp?ID=20664>.

[21] J. Follows, D. Straeten, Application-Driven Networking: Concepts and Architecture for Policy-Based Systems, IBM Red Book, 1999.

[22] S. Bhattacharjee, K. Calvert, E. Zegura, Active networking and end-to-end arguments, IEEE Network Magazine (1998).

[23] Q. Mahmoud, Cognitive Networks: Towards Self-Aware Networks, Wiley-Interscience, 2007.

[24] X. Yang, D. Clark, A. Berger, NIRA: a new inter-domain routing architecture, IEEE/ACM Transactions on Networking 15 (2007) 775–788.

[25] T.G. Griffin, J.L. Sobrinho, Metarouting, in: Proceedings of ACM SIGCOMM, 2005.

[26] M. Yuksel, A. Gupta, S. Kalyanaraman, Contract-switching paradigm for internet value flows and risk management, in: Proceedings of IEEE Global Internet Symposium, 2008.

[27] N. Milosavljevic, A. Nguyen, Q. Fang, J. Gao, L. Guibas, Landmark selection and greedy landmark-descent routing for sensor networks, in: Proceedings of IEEE INFOCOM, 2007.

[28] M. Perillo, W. Heinzelman, DAPR: A protocol for wireless sensor networks utilizing an application-based routing cost, in Proceedings of IEEE WCNC, June 2004.

[29] A. Alippi, G. Vanini, Application-based routing optimization in static/semi-staticwireless sensor networks, in: Proceedings of IEEE PERCOM, 2006, pp. 47–51.

[30] C. Gui, P. Mohapatra, Virtual patrol: A new power conservation design for surveillance using sensor networks, in: Proceedings of IPSN, 2005.

[31] C.R. Lin, M. Gerla, Adaptive clustering for mobile wireless networks, IEEE Journal on Selected Areas in Communications 15 (7) (1997) 1265–1275.

[32] A.B. McDonald, T.F. Znati, A mobility-based framework for adaptive clustering in wireless ad hoc networks, IEEE Journal on Selected Areas in Communications 17 (8) (1999) 1466–1487.

[33] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, S. Shenker, Ght: A geographic hash table for data-centric storage in sensornets, in: Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA), 2002.

[34] J. Bruck, J. Gao, A.A. Jiang, Map: Medial axis based geometric routing in sensor network, in: Proceedings of ACM MOBICOM, 2005.

[35] B. Leong, S. Mitra, B. Liskov, Path vector face routing: Geographic routing with local face information, in: Proceedings of ICNP, 2005, p. 147158.

[36] F. Kuhn, R. Wattenhofer, A. Zollinger, Worst-case optimal and average-case efficient geometric ad-hoc routing, in: Proceedings of 4th ACM Int. Symposium on Mobile Ad-Hoc Networking and Computing (MobiHoc), 2003.

[37] Y.-B. Ko, N.H. Vaidya, Location-aided routing (LAR) in mobile ad hoc networks, in: Proceedings of MOBICOM, 1998, pp. 66–75.

[38] G. Pei, M. Gerla, X. Hong, Lanmar: Landmark routing for large scale wireless ad hoc networks with group mobility, in: Proceedings of Mobihoc, 2000.

[39] L. Blazevic, J.-Y.L. Boudec, S. Giordano, A scalable routing scheme for self-organized terminode network, in: Proceedings of the Communication Networks and Distributed systems modelling and Simulation conference (CNDS), 2002.

[40] L. Blazevic, S. Giordano, J.-Y.L. Boudec, Anchored path discovery in terminode routing, in: Proceedings of the Second IFIP-TC6 Networking Conference (Networking 2002), 2002.

[41] G.B. Giannakis, Modeling and optimization of stochastic routing for wireless multi-hop networks, in: Proceedings of IEEE INFOCOM, 2007.

[42] U. Acer, S. Kalyanaraman, A. Abouzeid, Weak state routing, in: Proceedings of ACM MOBICOM, 2007.

[43] R.G. Garroppo, S. Giordano, L. Tavanti, A survey on multi-constrained optimal path computation: exact and approximate algorithms, Computer Networks 54 (17) (2010) 3081–3107.

[44] G. Xue, W. Zhang, J. Tang, K. Thulasiraman, Polynomial time approximation algorithms for multi-constrained qos routing, IEEE/ACM Transactions on Networking 16 (2008) 656–669.

[45] S. Chen, M. Song, S. Sahni, Two techniques for fast computation of constrained shortest paths, IEEE/ACM Transactions on Networking 16 (1) (2008) 105–115.

[46] Z. Li, D. Li, M. Liu, Interference and power constrained broadcast and multicast routing in wireless ad hoc networks using directional antennas, Computer Communications 33 (12) (2010) 1428–1439.

[47] W. Yang, W. Liang, J. Luo, W. Dou, Energy-aware online routing with qos constraints in multi-rate wireless ad hoc networks, in: Proceedings of International Wireless Communications and Mobile Computing Conference, 2010.

[48] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, Numerical Recipes in C, second ed., Cambridge University Press, 1992.

[49] E.P. Crescenzi, V. Kann, A compendium of np optimization problems, <http://www.nada.kth.se/viggo/problemlist>.

[50] Z. Michalewicz, D.B. Fogel, How to Solve It: Modern Heuristics, Springer, 1999.

[51] D. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison Wesley, MA, 1989.

[52] J.-C. Chen, K.M. Sivalingam, P. Agrawal, Performance comparison of battery power consumption in wireless multiple access protocols, Wireless Networks 5 (6) (1999) 445–460.

[53] C.E. Jones, K.M. Sivalingam, P. Agrawal, J.C. Chen, A survey of energy efficient network protocols for wireless networks, Wireless Networks 7 (4) (2001) 343–358.

Mustafa O. Kilavuz is a graduate assistant pursuing Ph.D. at the CSE Department of the University of Nevada – Reno (UNR), Reno, NV. He received a B.S. degree from Computer Engineering Department of Bilkent University, Ankara, Turkey in 2006. He received an M.S. degree from Computer Science and Engineering Department of UNR in 2009. His research interests are in the areas of wireless routing and robotics.

Murat Yuksel is an Assistant Professor at the CSE Department of The University of Nevada – Reno (UNR), Reno, NV. He was with the ECSE Department of Rensselaer Polytechnic Institute (RPI), Troy, NY as a Postdoctoral Research Associate and a member of Adjunct Faculty until 2006. He received a B.S. degree from Computer Engineering Department of Ege University, Izmir, Turkey in 1996. He received M.S. and Ph.D. degrees from Computer Science Department of RPI in 1999 and 2002 respectively. His research interests are in the area of computer communication networks with a focus on protocol design, network economics, wireless routing, free-space-optical mobile ad hoc networks (FSO-MANETs), and peer-to-peer. He is a senior member of IEEE, life member of ACM, and a member of Sigma Xi and ASEE.