University of Nevada, Reno

Minimizing Multi-Hop Wireless Routing State under Application-Based Accuracy Constraints

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

by

Mustafa Omer Kilavuz

Dr. Murat Yuksel/Thesis Advisor

May, 2009

University of Nevada, Reno
Statewide • Worldwide

THE GRADUATE SCHOOL

We recommend that the thesis
prepared under our supervision by

MUSTAFA OMER KILAVUZ

entitled

Minimizing Multi-Hop Wireless Routing State
Under Application-Based Accuracy Constraints

be accepted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

Murat Yuksel, Ph. D., Advisor

Kostas Bekris, Ph. D., Committee Member

M. Sami Fadali, Ph. D., Graduate School Representative

Marsha H. Read, Ph. D., Associate Dean, Graduate School

May, 2009

Minimizing Multi-Hop Wireless Routing State under Application-Based Accuracy Constraints

Mustafa Omer Kilavuz

University of Nevada, Reno, 2009

Supervisor: Dr. Murat Yuksel

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. 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 optimization framework, we investigate the tradeoff between the packet header size and the network state. We, then, apply our framework 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 packet header size and network state. We show that the problem of accurate representation of a trajectory with the objective of minimizing the cost incurred due to header size and network state is difficult to solve optimally. Finally, we develop heuristics solving this problem and illustrate their performance.

Annem, Babam ve Abime

Contents

List of Figures

Chapter 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 emphasized in sensor networks, especially in routing functions provided by the network. To accommodate various application-specific routing needs, the *expressiveness of the routing interface* must be at a sufficient level. 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 numerous 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 [3, 7] 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 [8, 35] by using network-specific properties such as geographic routing [14] capability. Such routing functions enabled application-specific traffic engineering [17], 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 its constraints to meat specific goals. Some recent work pinpointed the complexity of this issue within limited contexts, e.g., mimimal/maximal exposure path selection [10].

Although provisioning of rich routing building blocks to multi-hop wireless networking applications is of high interest, application-specific requirements and constraints emphasize the difficulty 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 thesis. 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 make the right routing choices are much needed, as the time to make such routing decisions is very minimal for mobile nodes [30]. We outline an optimization framework where various inter-layer design problems can be formulated, e.g., minimization of routing state under application-specific path constraints. Our optimization framework opens the doors for 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 the 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) [8, 35] under the application-specific constraint of "path accuracy". That is, we consider an

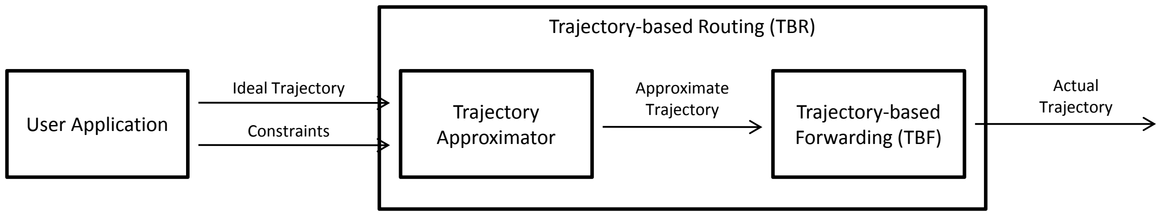

Figure 1.1: Trajectory-Based Routing (TBR) framework.

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 and be used in our optimization framework. In TBR, as shown in Figure 1.1, 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 to such that the packet follows the ideal trajectory as closely as possible. Although TBR is a good routing option, the decision to select the next neighbor optimally can be time consuming, which violates the basic premise of simple packet forwarding. Thus, the nodes must work on an "approximate trajectory" rather than the ideal trajectory which can be quite complicated depending on the application. The approximate trajectories can be a concatenation of several segments of "easy to handle" trajectories such as a line, a third order polynomial curve, or a Bézier curve. Then, this approximate trajectory is used to make forwarding decisions for the "actual trajectory/route", as illustrated in Figure 1.2.

When we examine the structure of TBR, we identify three major problems that must be solved in order to make it work: (i) to provide the ideal trajectory, (ii) approximating the ideal trajectory and (iii) packet forwarding on the approximate trajectory.

The ideal trajectory can be directly provided by the user, an application or

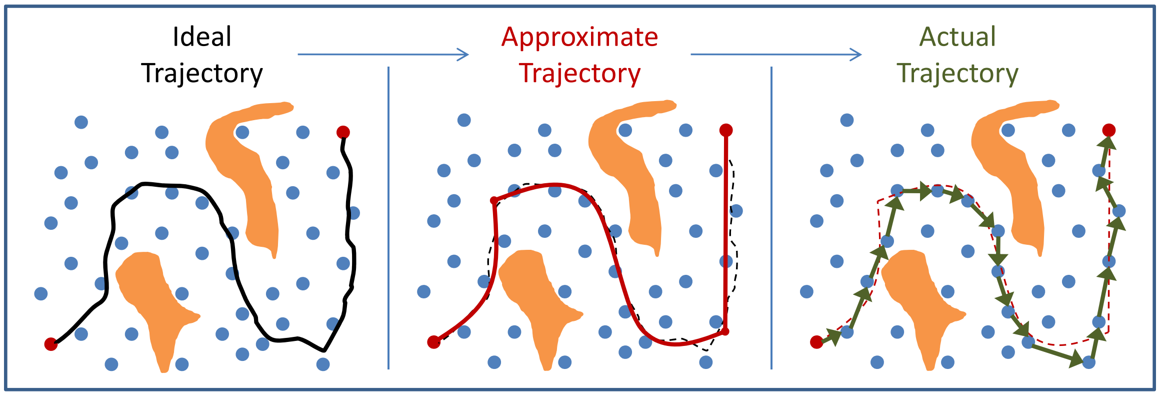

Figure 1.2: Ideal, approximate, and actual trajectories.

upper layers, e.g., transport layer. Although the ideal trajectory can be provided manually, a method is needed to generate it automatically considering the application-specific needs such as quality of service, obstacle avoidance, and load balancing. We present an approach to this problem by using path planning algorithms, which is explained in Chapter 5.

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 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 critical. Thus, to reduce the computational time requirements of our solution, we map the problem to a genetic algorithm and design two heuristics as well. The approximation problem is covered in Chapter 3

The *packet forwarding* phase of TBR is called Trajectory-based forwarding

(TBF) and fortunately it has already been studied by several groups [8,35]. They are briefly explained towards the end of the next chapter.

The rest of the thesis is organized as follows: We start with the literature survey in Chapter 2 and give some information about some of the very popular routing protocols with their strengths and drawbacks. Trajectory-based forwarding methods and Trajectory-based Routing are also explained here. We outline the problem of approximating an ideal trajectory under application-specific accuracy constraints and four methods to solve it in Section 3. We present results of our simulation experiments in Chapter 4. We describe our trajectory planning algorithm for providing the ideal trajectory in Chapter 5. Finally summarize our work in Chapter 6.

Chapter 2

Literature Survey

There has been considerable recent interest in mobile ad-hoc networks (MANETs) [4, 12, 14, 16, 23, 27], and sensor networks. The main focus of MANET research has been on scalability and complexity issues due to underlying dynamism, specifically in routing. Routing tackles the problem of establishing an *indirection* from a *persistent name (or ID)* to a *locator*. In today's internet routing, this indirection translates to IP address prefix being the ID and next hop router being the locator. Routing protocols such as AODV [23], DSDV [24], 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 [22,27] or consistent hashing [16,28] or filtering [26]). MANET routing has grown into two subclasses: *proactive* routing [24] (using early binding) and *reactive or on-demand* routing [12, 23] (using late-binding).

2.1 Proactive vs. Reactive Routing Protocols

Proactive routing protocols always store routing information tables in nodes and inform the other nodes of the topological changes. Storing large tables and sharing these information via packets causes the nodes to consume too much energy if the nodes are wireless. In particular, highly mobile nodes may cause the topology to be more dynamic and keeping up with the topology changes becomes prohibitively costly.

Destination-Sequenced Distance Vector (DSDV) [24] is a popular highly proactive loop-free routing protocol. Every DSDV node stores a routing table that maps a neighbor to each destination node, which makes that neighbor node the next hop to reach the destination. A source node sending a data packet, or an intermediate node forwarding one, looks up the table for the corresponding entry and sends or forwards the packets to the next hop. DSDV uses distance vectors to find the shortest path to a destination. In order to have the up-to-date information stored in the routing tables about the dynamic network topology, the nodes periodically broadcast or multicast new updates about the important changes on the distance values to the reachable destinations. When the network is highly dynamic, these updates should be transmitted more often to establish consistent and loop-free routing.

Each entry in the routing table has a metric and a sequence number besides the destination and the next hop. The metric gives information about the path length to the destination through the next hop and the sequence number tells how old the information is. Higher sequence numbers mean newer information. If there are two entries for a destination, the one with the higher sequence number is used and the older one is discarded. If the sequence numbers are the same, the one with a lower metric value is chosen. When a link is broken, the corresponding metric value is set to ∞ and update messages are sent to the other nodes.

Reactive routing protocols, however, use route discovery methods before transmission of data packets instead of storing state information in routing tables. The nodes do not need to inform each other about the topological changes periodically; but instead, there is delay in discovering a route before the data transmission starts. Moreover, the route is not guaranteed to be a feasible one.

Ad hoc On Demand Distance Vector (AODV) [23] is a scalable, loop-free and reactive routing protocol for MANETs, capable of both supporting unicast and multicast routing. When a source node wants a route to a destination that does not already exist in its routing table, it initiates a route discovery phase. It broadcasts a route request (RREQ) packet including its and the destination's IP address, its sequence number, destination sequence number, broadcast ID and time to live (TTL) number. The receiving node, except the destination node, checks its routing table to see if it has any routing information about the destination. If there is such an entry, it checks the destination sequence number to see if its information is newer. If this route information is newer than whatever the source node knows or this is the destination node, a route reply (RREP) packet is prepared and sent back to the source node. Broadcast ID field in the packet is used to prevent any duplicate receptions of the same packet. After receiving the RREP packet, the source node may start the data traffic through that route. If another RREP packet is received for the same request with less number of hops, the route is updated with the new information.

When an intermediate node cannot forward the packet to the destination, it sends a route error (RERR) packet to the source node. Every node receiving RERR packet on the route updates its routing table for that destination node. If the source node still wants a route to the destination, it reinitiates the route discovery phase. Every node on a route stores the information of the next hop for the destination. This information is only written to the table when a route is discovered and stored as long as the route is active.

Greedy Perimeter Stateless Routing (GPSR) [14] is a reactive geographic routing protocol for wireless networks that uses the positions of the routers and the packet's destination. The protocol allows routers to be nearly stateless, which means they store very few information about the network. Each node only needs its neighbors' positions instead of other topological information. Every node is assumed to have a mechanism, maybe a GPS device [21], to identify its own location. Also, the source node knows the location of the destination node. The tradeoff GPSR or other geographic routing protocols make is to equip each node with a localization device capability and practically eliminate the need for storing and maintaining the global state. In other words, the *global routing state* is replaced with a new *global invariant* of "coordinate system".

GPSR consists of two packet forwarding methods. It uses "greedy forwarding" when possible and "perimeter forwarding" otherwise. Greedy forwarding forwards the packet to the neighbor which is closest to the destination and stops when the packet reaches its destination. This method does not work when a node has no neighbors closer to the destination than itself, which means there is a "void area" in the network and the packet is at the tip of a peninsula. In such situations, the packet should be forwarded to a farther node around the void area. Perimeter forwarding provides algorithms to choose the node to forward and pass over these void areas.

Distance Routing Effect Algorithm for Mobility (DREAM) [4] is also a geographic routing protocol and can be considered as both proactive and reactive.

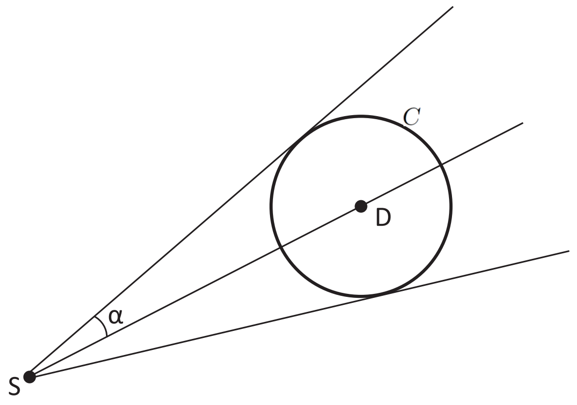

Figure 2.1: The circular area C that D could be located in and the angle α that covers it.

DREAM was designed for mobile ad-hoc networks based on two novel observations. The distance effect is the first observation that two nodes having the distance greater in between seems like moving slower with respect to each other. This makes the nodes which are farther from each other need less updates about each other. Secondly, the nodes moving faster need to send update messages more frequently. So, the update messages sent are dependent to the mobility rate and also the distance to the other nodes. These messages include a distance threshold value which limits the distance they travel. Most of these messages have a short limit. Thus, farthest nodes receive only smaller number of the messages while the closer nodes receive much more.

A source node S, wants to send a packet to a destination node D, already has information about D's location and the packet is forwarded to D directionally. However, since the nodes are mobile and the updates are sent at every time interval, S cannot know the exact position of D. But, it can determine the approximate velocity of D by looking at the last couple update messages and also it is assumed that it knows the possible maximum value of D's velocity. By using this information, as seen in Figure 2.1, a circular area C is determined where the center is D's last known location and the radius is the maximum distance D could have traveled. C is used as the destination area that D is located in. The data packets are sent to all neighbors in the direction of this area. If the velocity is not known, an application specific probability value p is given and the minimum angle α is calculated by using the probability mass function so that the probability of finding D in that direction within the range is p.

DREAM can make more than 80% of the packets reach the destination directly, and the rest are also guaranteed to be delivered with a recovery policy. However, no specific method exists about how this is supposed to work, as this depends on the characteristics of the network.

2.2 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. Providing routing expressiveness and flexibility to user applications has been of great interest [9]. In the general context of routing, defining application-specific custom routes through user-defined routes [32], through a declarative language [7,29], or through concatenations of several user-chosen contracts [34] have attracted interest.

The current internet protocols are arguably robust and efficient, but hard to change to accommodate the needs of new applications such as improved resilience and higher throughput [3]. Upgrading even a single router is hard, and implementation of a distributed routing protocol correctly is even harder, because every single node must be accessed and modified in order to upgrade a protocol already deployed.

There are several methods proposed to solve this problem. Overlay networks consider implementing routing protocols at the application layer by the third parties. This only moves the problems from the network layer to the upper layers and still the whole protocol has to be implemented or upgraded on the overlay nodes. Another method proposed is Active Networks, which does not need direct access to the nodes for upgrades. But, this solution is mentioned to have several problems such as performance, security and reliability.

Declarative Routing [3] is a design which keeps a better balance between the extensibility and the robustness by using the database techniques. The key idea is to implement a routing protocol by writing simple database queries and running them in some or all of the routers. Declarative Routing can be used in a variety of ways. ISPs could be given the control of their own routers and they can use declarative queries to implement new protocols or manage the existing ones. Another possible way could be the flexibility of the end-user usage. The end-users or their applications might be able to decide the properties of their routes of their own traffic by using declarative queries. The most important premise of declarative routing is the capability of expressing other popular routing protocols within the declarative framework.

A specific design of declarative routing involves a query processor deployed in every node which gets queries and processes them. Basically, there are two tables in routers: the neighbor table and the forwarding table. The query processor is receiving queries from outside and the neighbor table updates and makes changes in the forwarding table with the result queries. This system is the only interaction between the query processor and the router's core forwarding logic.

Adapting the same declarative query system to the sensor networks was considered possible in [7]. Today, the sensornet programming is still too hard even for expert programmers. The sensornet programs and protocols are written in low-level embedded languages. Also, in sersornet programming, the programmer has to deal with issues such as wireless communication, limited resources and asynchronous event processing. It is a big challenge to design a general purpose, efficient and easy-to-use programming model. Authors present the design and implementation of a declarative sensor network (DSN) platform: a programming language, compiler and runtime system to support declarative specification of wireless sensor network applications. 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) [12], which gives a full flexibility to the user application to define the routes. The source makes a list of all the nodes starting from itself to the destination and puts this route information into the packets' headers. When a node receives the packet it knows where to forward the packet by using this information. In this protocol there are no periodic advertisements which makes it a reactive routing protocol. Whenever a source needs a route to a destination, it is dynamically discovered according to the cache information and route discovery phase.

Every node stores a "route cache". When a source wants to send a packet to a destination, it first looks for an existing route in the cache and uses it if exists. If not, it generates one by route discovery. A host initiating a route discovery broadcasts a route request packet which includes the information of the host and the target. Every node receiving the route request packet adds itself to the list on the packet and rebroadcasts. When the packet reaches the target, it will have the full route from the host to the target. Then, the target sends the route reply packet over that route or an existing route in the cache back to the host. During the data packet traffic, the host monitors the route to make sure that the packets reach the destination. If a node on a route is no longer operational, the route is fixed by initiating a route maintenance phase.

Since DSR could not scale to a sufficient number of nodes, schemes like Trajectory-Based Routing (TBR) [8,33] and landmark selection [20] 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 [18] and centralized optimization of routing for application-based goals [2,5].

Authors in [2] present an application-based optimization method based on the power consumption issues of routing algorithms on wireless sensor networks. The method is designed to work on static/semi-static nodes because the method does not update the nodes that frequently.

The method starts with a random deployment of the nodes on a field. The nodes send their Received Signal Strength Indicator (RSSI) and the power level to a central unit to have an RF mapping of the deployment. RF-based localization algorithm generates a relative topological spatial configuration of the network by using this information. According to the localization information, only a subset of nodes is chosen as sensing nodes as the first stage of optimization. The main stage of the optimization is the simulation of the routing algorithm in the central unit with several different parameters. After finding out the best parameter set for the current topology, all the nodes are configured according to the optimized parameters. The optimizations and the updates are done periodically in a loop. As a result, a saving of about 40% in power consumption is calculated over the MultiHopRoute routing algorithm in TinyOS [1].

Distributed Activation based on Predetermined Routes (DAPR) [18] is a distributed, integrated sensor management and routing protocol which is designed to maximize the lifetime of the sensor network while satisfying the application-based demands. The focus is on the network field coverage preservation which is the application requirement in this case. The protocol starts with assigning values to every node according to their importance in the network. There are two parameters used in calculating this value the residual energy of the node itself, and the neighboring nodes covering the area around that node. The lower the residual energy of the node, the more important it is, because we want the nodes to live as long as possible. Also, if any part of the sensing area of a node is not well covered or not even covered by neighboring nodes, then this node is also important for being one of a few nodes, or the only node, covering that area. The sensing, routing and sleeping roles are assigned to the nodes according to their importance. While more important nodes are assigned to be sensing nodes, less important nodes are assigned to be routing nodes. When there is a packet to be sent over a path, the path is chosen to have the minimum cost which depends on the importance of the nodes over the path. This method selects less important nodes to be on the path spending their energy to forward packets. Finally, there are some nodes called redundant nodes that will not be used for a while because there are already many neighboring nodes.

The protocol works in rounds, each with a constant duration. Each round starts by floating "Round Start" messages around by a central unit. Each round starts with a route discovery phase. The rounds are selected so that the minimum cumulative cost paths are used. In the next phase the nodes decide to sleep or wake up. And then, until the next round they continue their normal operation.

2.3 Trajectory-Based Forwarding

As a generic routing framework for accommodating application's desires at the routing level, Trajectory-Based Forwarding (TBF) was proposed in [8]. TBF is based on forwarding packets on a predefined trajectory. The trajectory is set by the source and the forwarding decision is done according to the trajectory that is encoded in each packet. TBF does not require communication overhead or infrastructure support for route maintenance. But it assumes every node knows its position by a GPS system for example. TBF decouples routes from routers and does not rely on the nodes, but on the route. It even becomes possible to perform "source routing" in mobile environments.

TBF relieves the nodes from storing large forwarding tables and flooding for route discovery. On the other hand, an encoded trajectory increases the packet header overhead. The overhead is not related to the length of the trajectory but rather to its complexity.

In TBF, the parametric equations (e.g., $X = X(t), Y = Y(t)$) can be used to represente the trajectories. The time parameter, t, can be quite useful in forwarding process as it represents how far the packet has traveled on the trajectory. Functional representation (e.g., $Y = f(X)$) is not sufficientle flexible to represent all types of curves and equational representation (e.g., $X^2 + Y^2 = R^2$) may be hard to solve and require the knowledge of points on the curve. Complex trajectories can have a large number of parameters to encode into the packet header so they can be divided into simpler parts to reduce the overhead.

For the forwarding decisions, the t parameter of the closest point of the curve to a node defines the place of that node according to the curve. If t_1 of node N_1 is less than t_2 of node N_2, then, N_2 is at a more forward position than N_1 is. This means N_2 is closer to the destination according to the curve. The t parameters for each node are used to provide some forwarding methods.

In [35], six forwarding methods were presented and evaluated. The first method is a simple random method that might be useful for wireless networks with nodes having low computational and transmission power. Closest to curve (CTC) is also a computationally simple method that always forwards the packets to the closest node to the trajectory. This method works well if the accuracy constraint is crucial. Least advancement on curve (LAC) forwards packets to the node with the lowest t parameter. This method may be good, for example, if all the nodes around the trajectory must be traversed. Hybrid of CTC and LAC (CTC-LAC) chooses the closest node to itself whose distance from the trajectory is less than D. If there are no nodes satisfying the restriction, then, the distance D is increased with a step ΔD until such a node is found. Most advancement on curve (MAC) chooses the node closest to the destination. Lowest deviation from curve (LDC) calculates an area for each node and forwards to the node with the lowest area. This area is calculated as follows: Assume N_0 wants to forward a packet and N_1 is its neighbor and let P_0 and P_1 be the closest points on the trajectory to the corresponding nodes. The deviation is the area between the line segment $\overline{N_0N_1}$ and the curve segment $\overline{P_0P_1}$.

2.4 Trajectory-Based Routing

In Trajectory-based Routing (TBR) [8, 35], the source node encodes the trajectory into packet headers, and 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 Figure 1.2, 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 must still determine the best way of attaining the goal of forwarding the packets along this ideal trajectory.

Since the trajectory is encoded into the packet headers, the ideal trajectory must 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 simple such as a straight line, a polynomial curve, or a Bézier curve. Hence, the ideal trajectory may not be representable exactly, depending on the complexity of the ideal trajectory. Instead of representing the exact ideal trajectory, we focus on generating an “approximate trajectory” which is easier to represent, but differs slightly from the ideal trajectory. To give the user application a knob on how the routing function formulates this approximate trajectory, we consider the limits of the difference between it and the ideal trajectory as an “accuracy constraint”.

Although the approximate trajectory is optimized under the constraints, it is sometimes too complex to encode the whole trajectory into the packet headers. Hence, the approximate trajectory is divided into small segments so that each segment can be encoded into packet headers separately. To manage this, some Special Intermediate Nodes (SINs) [35] are selected around the points where the approximate trajectory is divided. The SINs store the information of the trajectory segment starting from that point only. The packets departing from the source receives only the information about the first segment. When they reach the end of this segment, they acquire the next segment’s routing information from the corresponding SIN. The packets

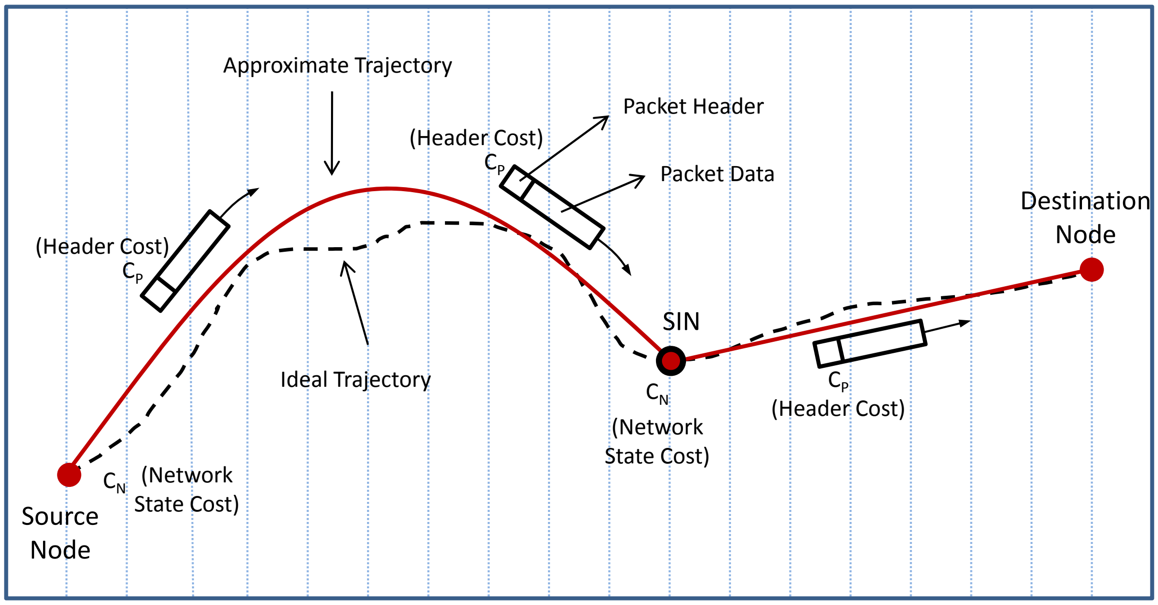

Figure 2.2: Packet forwarding in TBR.

keep changing the routing information in their headers whenever they visit another SIN until they reach the destination. For example, in Figure 2.2, the approximate trajectory is divided into two segments, 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 the 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 *the packet header cost* and by the routing information stored in the SINs *the network state cost*.

Various Trajectory-based Forwarding (TBF) techniques were proposed [8,35] to manage the packet traffic over the 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 Figure 1.2. In this thesis, 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.

Chapter 3

Trajectory Approximation

3.1 System Model

When a trajectory is approximated by a series of representations, each segment of the trajectory is associated with a cost in terms of packet header length and network state. We construct the problem of minimizing the 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 segment of the approximate trajectory contributes an error to the total error of the whole approximate trajectory. The accuracy constraint restricts 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 segment of the trajectory. These are denoted as $r_1, r_2, ..., r_k$. Here r_i is a straight line, a polynomial curve, or a Bézier curve, etc.
- The space in which the trajectory exists is discretized, and there are a maximum

of m points (excluding source and destination), or $m+1$ segments into which the trajectory can be split.

- Since each segment 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, which denotes the representation used for a particular segment 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 segment 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 part of the ideal trajectory with the least error possible by using linear programming [31].

- There is a packet header cost C_P and a network state cost C_N for each segment of the trajectory, depending on the representation used for the segment of the trajectory being considered.

We can now formulate the following binary program:

$$\min \sum_{i=1}^{m} \sum_{j=1}^{k} C_P(j) Q_{ij} + C_N(j) Q_{ij} \tag{3.1}$$

$$\text{Subject to:} \tag{3.2}$$

$$\sum_{i=1}^{m} \sum_{j=1}^{k} e(Q_{ij}) \leq E \tag{3.3}$$

$$\sum_{j=1}^{k} Q_{ij} \leq 1 \;\; \forall\, i = 1 \cdots m \tag{3.4}$$

$$Q_{ij} \in \{0, 1\} \tag{3.5}$$

In the formulation (3.1-3.5), the constraints (3.3) denote the error associated with the representation of each segment of the trajectory. Note that, we do not necessarily select all of the m points available, which is captured in constraints (3.4). In constraints (3.3), the sum of the errors must not exceed an application-defined error E which we assume is an input to the problem. Constraint (3.5) states that Q_{ij} is a binary variable.

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 $(3.1 \cdots 3.5)$ is *node* based, the edge formulation is implicit. For example a solution which selects some $l \leq k$ 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 [25]. This is represented diagrammatically in the Figure 3.1. In Figure 3.1, 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.

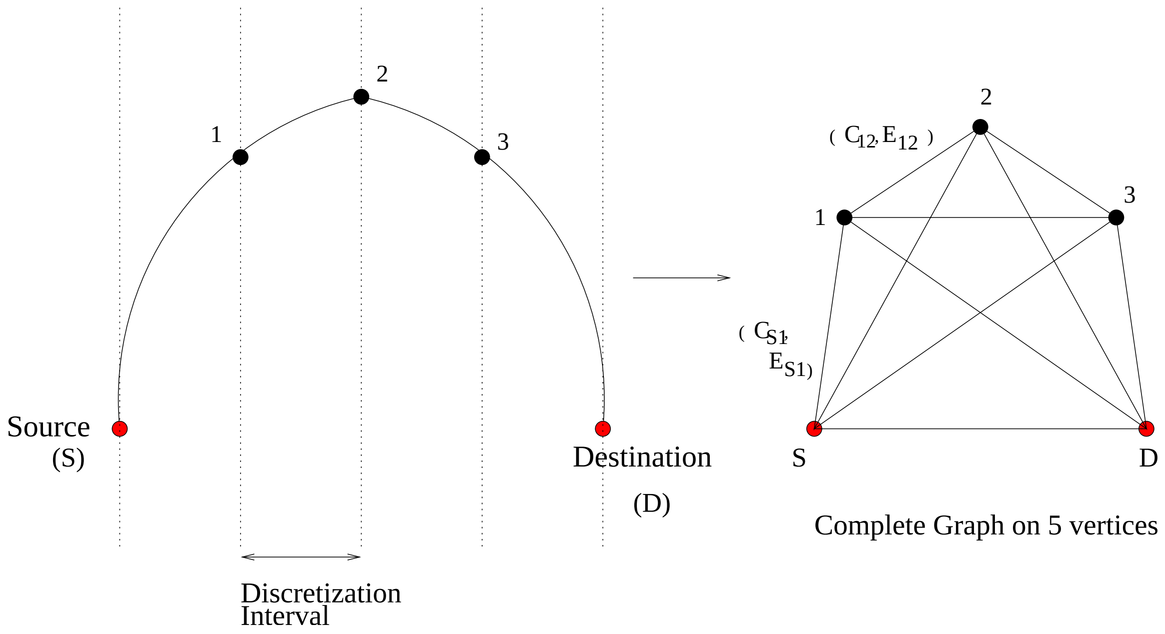

Figure 3.1: A sample graph formulation of the discretized trajectory approximation problem.

3.2 Cost Calculations

The costs C_P and C_N in the objective (3.1) 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 thesis, we consider three different representations. More complex representations are, the more costly they will be, but complex representations fit better andreduce the error. 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 variable. Considering each double takes 8 bytes, the space needed to express a line is 4×8=32 bytes.

- *Quadratic polynomial curve, $y = ax^2 + bx + c$:* For a quadratic curve, we must store x_1, x_2, a, b, and c. We do not need y_1 and y_2 since they can be calculated by substituting x_1 and x_2 into the equation. Thus, the space needed for a quadratic polynomial curve is 5×8=40 bytes.

- *Cubic polynomial curve, $y = ax^3 + bx^2 + cx + d$:* Similarly, for a cubic 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 6×8=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 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 $c_pR_j + c_NR_j$, 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 states, we assume that $c_P = c_N = 1$. We will also show how different weights effect the structure of the approximate trajectory.

3.3 Error Measures for Trajectory Approximation

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 segment. 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. Clearly, we must define what "100% error" is, which we think 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. We defined the distance between the ideal trajectory and a projection as 50 pixels in our experiments. So, 100% error is $(50 + 50) * width$ unit area. This distance can vary according to the size of the space and might be defined as a percentage of its height. For instance, 1/4 or 1/8 of the height of the space.

3.4 Methods to Solve the Problem

As trajectory approximation is an NP-hard problem, it requires fast solutions. In practice, this problem is solved quite 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. We now outline four techniques to perform 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.

3.4.1 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

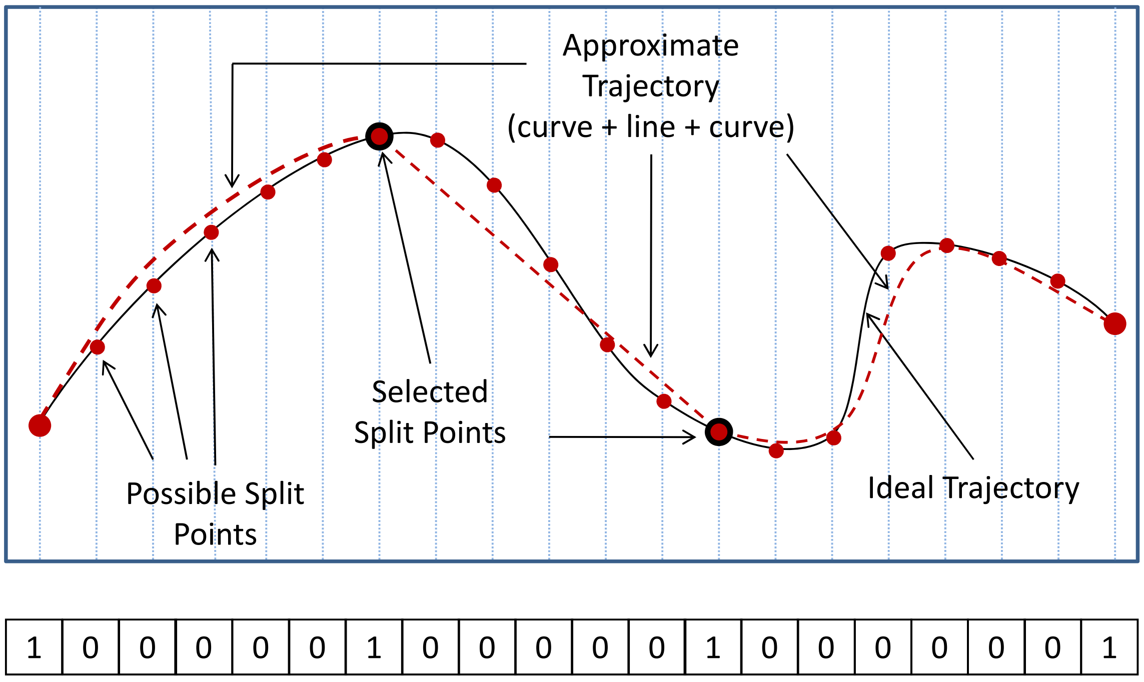

Figure 3.2: Split points, trajectories, and selection bits.

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. 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 is chosen. The approximate trajectory formed by these representations which is below the tolerated error bound and has the minimum aggregate cost is chosen as the best solution.

An example is shown in Figure 3.2 where 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 has one of the 3 representations, i.e. a line, a quadratic or a cubic curve. Here, a quadratic curve, a line and a cubic curve forms the approximate trajectory.

We applied the pruning method [19] 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 assigning a 0 or 1 to each bit. When we assign 1, we branch our searching tree for every type of representations. When we select 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 found any), we stop searching for a solution having the current bit sequence. Obviously, regardless of the remaining bits, the current sequence will not yield an optimal solution.

3.4.2 Genetic Algorithm

Genetic algorithms (GAs) are search algorithms based on the mechanics of natural selection and genetics [11]. We have a population of members and 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 Figure 3.3, each of which can be selected for approximating the trajectory. Similar to the exhaustive search approach, the first bit represents the source point, the next N bits are for N possible split points, and the final bit is for the destination point. The bits for the source and

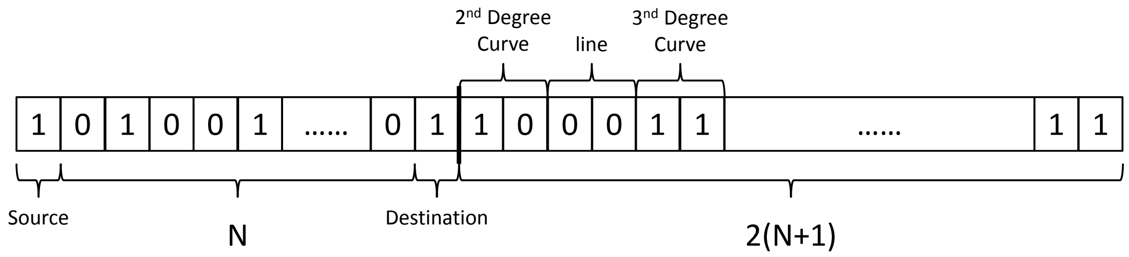

Figure 3.3: A sample chromosome in GA solution to the trajectory approximation problem.

the destination are always set to 1. We included them in the chromosome for 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, for a total of $2(N+1)$ bits. These 2 bits define which representation are used between the consecutive split points. 00 and 01 represent a line, while 10 represents a quadratic curve and 11 represents cubic curve. If there were more than four possible representations, then we would need more than 2 bits for each point. For example, in Figure 3.3, the first segment of the approximate trajectory is a quadratic curve between the source point and the first split point. A cubic curve follows it between the first and the second split points. The line in between 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 [11], i.e., the probability of the members to be chosen is proportional to their fitness values which are the aggregate cost of that member plus a handicap for the members exceeding the error bound. Thus, 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 repeat this process 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 and the unfeasible members which exceed it. The latter 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 can lead to a very good solution with minor changes. Besides, in some of the generations (especially the initial ones) there may be no feasible solutions. When the stopping condition is satisfied, the best feasible member having the least cost is selected as the best solution.

3.4.3 Greedy Heuristic 1: Equal Error

This algorithm is the fastest one. First, we fit a representation to the whole trajectory. If the approximate trajectory is above the error bound, we divide the trajectory into two or more segments, where the number of segments 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 segments at once with Every segment equal in size on the x-axis. Then, we find the best representations to fit on each segment. We keep dividing segments that are over the error bound recursively until all the segments satisfy the error bound.

3.4.4 Greedy Heuristic 2: Longest Representation

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 each segment until we reach the end. Starting from the shortest interval (in terms of number of split points) for a segment, the interval is increased every time until no suitable fit exists. The last suitable fit is chosen for that segment. We then continue our process for the following segments starting from the end point of the last segment decided.

For example, for the first segment we initialize the interval to [0,1]. As long as we can find a representation to fit into that interval we progressively increase it by one, i.e., [0,2] ..., till [0,88], say, if we cannot find a representation for the interval [0,89] and the representation selected for [0, 88] is a cubic, then we begin the search for the next segment from 88 with a cubic representation for [0, 88]. The step size of 1 for this example can be larger. This way speeds up the algorithm, but reduces its resolution and the algorithm is less likely to find better solutions.

Chapter 4

Experiments

4.1 Performance Evaluation

We performed performance evaluation of the four algorithms from Section 3.4 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 the quality of their trajectory approximation and their running time.

- *Customization to a power-scarce network:* Illustrate that our optimization framework can be customized for a power-scarce network by changing the weight of the packet header cost in the aggregate routing state cost.

4.1.1 Experimental Setup

We used a 400×400 pixel area in our experiments. The source node is located randomly on the y-axis while the destination node is randomly located on the $x = 400$ line. The trajectory between the source and destination nodes is made up of small line segments and has a *complexity* parameter in [0,180]. The complexity parameter defines the maximum value in "degrees", the maximum angle between each pair of consecutive line segments. The higher this parameter, the more complex (zigzag) the trajectory will be. If 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 and applied the four trajectory approximation algorithms on each of them. We ran the genetic algorithm 32 times on each trajectory and report the average result of those 32 realizations. The other three algorithms do not require multiple runs because they do not have a random factor. We now present the rest of our experimental setup specific to each of the four algorithms.

Exhaustive Search

In the exhaustive search, we set the number of possible split points to 19. Hence, 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 setup, it sometimes took many hours just for one run.

Genetic Algorithm (GA)

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 selected after running many experiments and believe to be near the 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 our results show that, it is possible to use a lot more splitting 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 define the stopping condition to stop if a solution better than the current best is not generated within 100 generations.

Greedy Heuristic 1: Equal Error

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

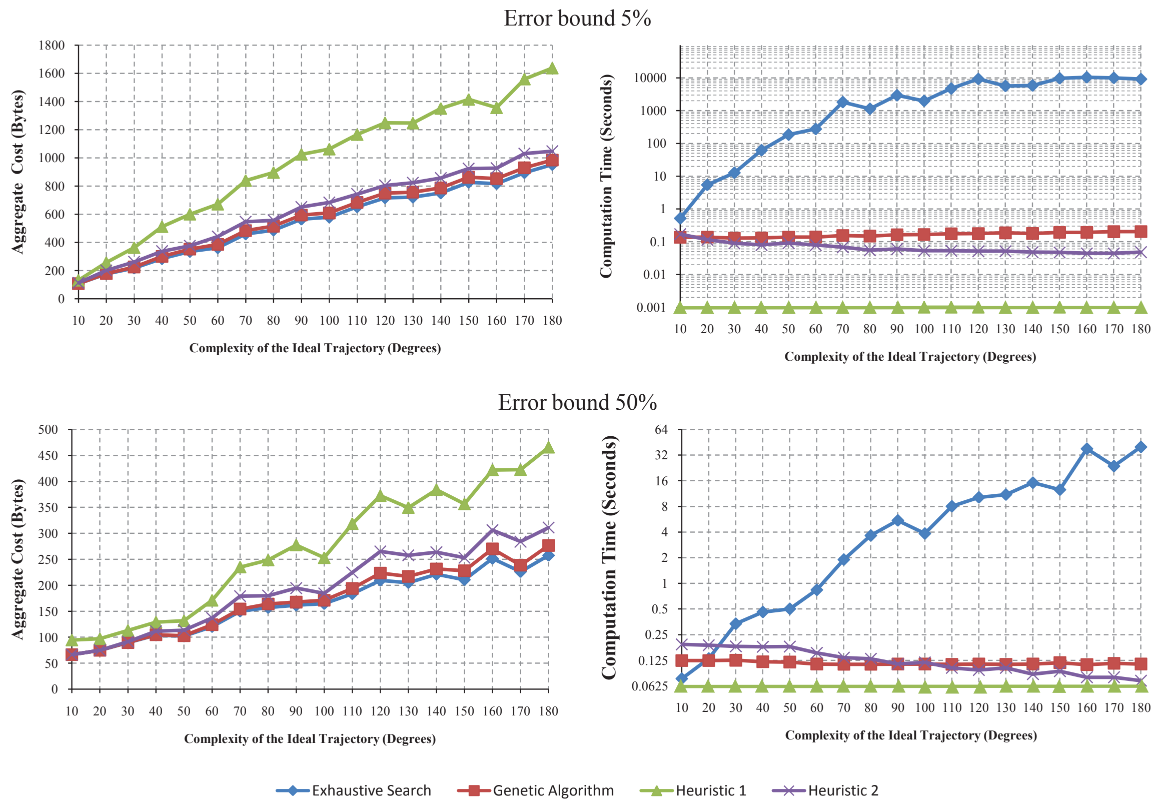

Figure 4.1: Results of trajectory approximation. Left two: Aggregate routing state cost vs. Trajectory complexity. Right two: Calculation time vs. Trajectory complexity.

Greedy Heuristic 2: Longest Representation

We set the step size to 1 to get the best result. Since running time is not much of an issue for this heuristic, we kept the step size small. In larger areas it may be necessary to increase the step size to reduce the computational time.

4.2 Results & Discussion

Comparison of Exhaustive Search, GA, and Heuristics

We compared the algorithms in two measures: the aggregate cost and the running (calculation) time. We present the results for error bounds 5% and 50% only as the results for the other two 10% and 25% exhibit similar trends. In Figure 4.1, the plots at the left column show the aggregate cost for trajectories with different complexity values. We see that the Exhaustive Search and Genetic Algorithm (GA) give very close results. For the given resolution, Exhaustive Search finds the optimum solution. Since GA has the same resolution with Exhaustive Search, we can say that GA performs well. Also, while Heuristic 2 acceptable, the same is not true for Heuristic 1. Another point we observe is that both the complexity of the trajectory and low error bound increases the cost. The reason is that the trajectory cannot be divided into a small number of simple representations. More complex representations increase the aggregate cost.

The plots at the right column of Figure 4.1 show the running time of the algorithms. It is clear that Exhaustive Search is not practical with several seconds of running time, Although it finds the optimum solution in terms of the routing state cost. Under 5% error, the Exhaustive Search takes several hours to run in some cases. Heuristic 1 runs in almost no time, but is not a suitable choice because its performance is unsatisfactory. Heuristic 1 might be a choice only where time is crucial and routing state cost is not an issue. Overall, GA and Heuristic 2 run in a reasonably short time, while giving good performance in minimizing the routing state cost. With this set up, GA runs in less than 0.2 seconds, and it is quite useful for the initial approximation of the trajectory.

Customization to Power-Scarce Networks

We also ran experiments for a customized objective (3.1) of cost and observed how the solution is adjusted to the new definitions. Specifically, we focused on the trade-off between packet header cost and network state cost. It is well-known that data transmission consumes larger power than storing the data [6, 13]. 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 segment the packets have to travel as a weighting factor in our objective (3.1).

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 trajectory, the packet header cost is the same with 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 has to go through more than one node and we calculate cost for every traversed node. Because this cost is dependent on the density of the nodes. We assumed that for every 20 pixels the packets visit another node. For example, for a quadratic curve which is 148 pixels long, 8 nodes are passed, and so C_P is 32×8=256 bytes.

Comparing both cases, we observed the number of segments that the trajectory was split into and the average complexity of the representations used. We used our GA implementation only. For simplicity we used 1 for straight line, 2 for quadratic curve and 3 for cubic curve for the complexity values of the representations. We plotted these measures for trajectories with different complexities and different error

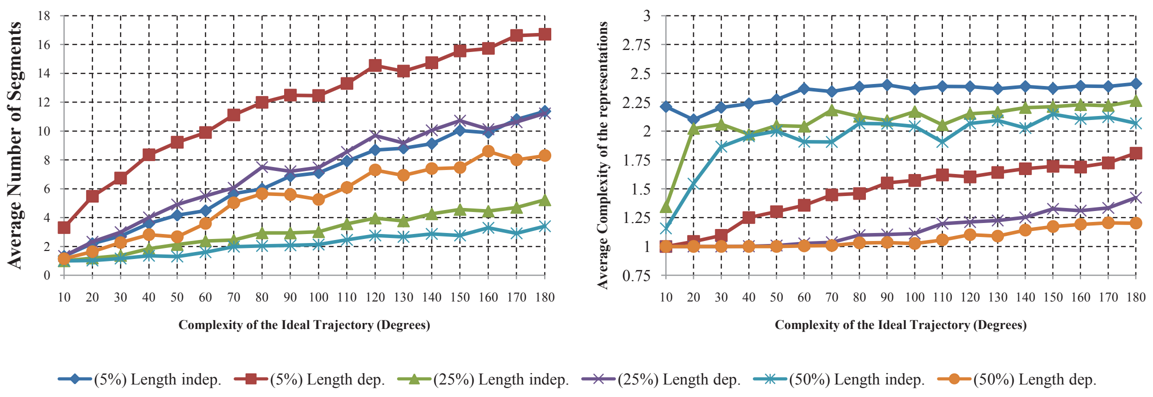

Figure 4.2: Customization of the routing state costs for a power-scarce network. Cost of packet header state can be made dependent on the length of trajectory segment the packets have to travel.

bounds as in Figure 4.2.

We observe that, regardless the error bound, the length dependent calculation 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 Figure 4.3(a), the left picture shows the approximate trajectory generated with length independent calculations. It is made up of 3 curves and 1 line. On the other hand the approximate trajectory generated with the length dependent calculations, in Figure 4.3 (a), has more segments and most of them are lines. We can conclude that the approximate trajectory generated by GA is completely flexible and is adjusted for different customizations of the ideal trajectory and the constraints.

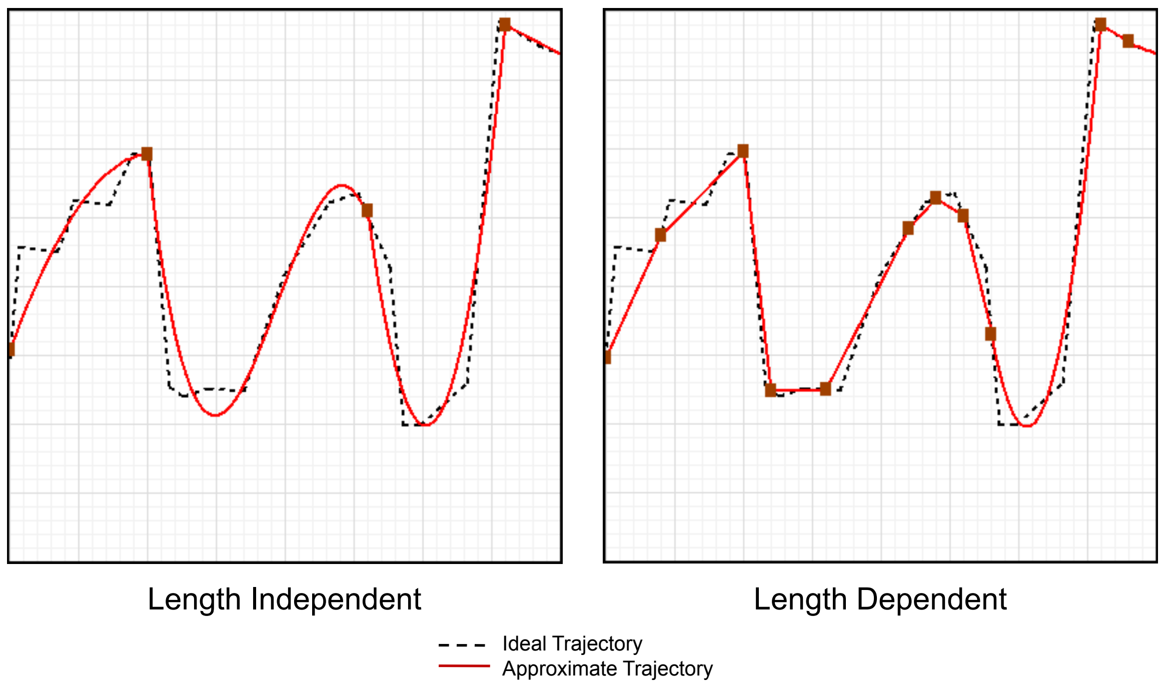

Figure 4.3: Sample runs for different cost calculations. Trajectory complexity: 120, Error bound: 25%

Chapter 5

Trajectory Planning

The ideal trajectory is one of the TBR inputs to initiate the data traffic from the source node to the destination node following that trajectory. The ideal trajectory can be directly provided by a user, an application or upper layers, e.g., transport layer. Although the ideal trajectory can be provided manually, a method is needed to generate it automatically considering the application-specific needs such as quality of service, obstacle avoidance, and load balancing.

In this chapter, we present a method that generates trajectories for TBR. The major concern about these trajectories is that they should avoid obstacles or void areas to deliver the packets successfully. We assume static nodes are deployed on a planar field and the nodes has no information about the environment, so the method should learn which parts of the field are good or bad for a connection. Our method is also scalable and flexible.

TBR works on the trajectories instead of the intermediate nodes (hops) between the source and the destination except SINs. It does not know which nodes were used in the transfer and does not keep track of them. Hence, our method uses a

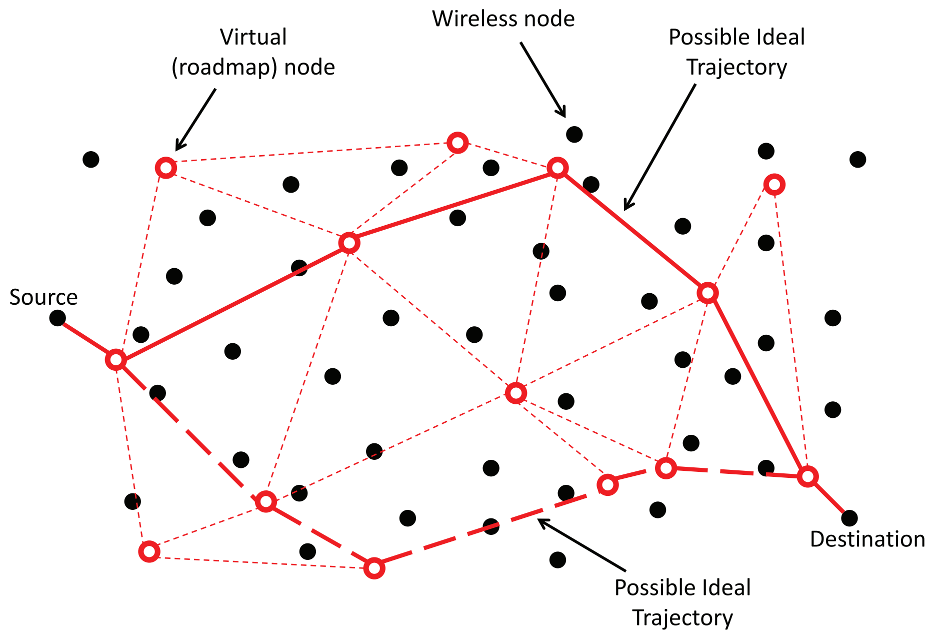

Figure 5.1: Probabilistic roadmap and generated ideal trajectories.

probabilistic roadmap [15] in the workspace that the network is deployed instead of the network topology to generate trajectories. We start by sampling roadmap nodes and connect each node to k-nearest neighbors or the nodes which are closer than a specified distance. Each edge is given a weight proportional to the distance between its two nodes. The lower the weight of an edge, the higher its preference. To generate a trajectory between a source and a destination node, these nodes should be connected to the roadmap as the roadmap nodes were connected to each other. The shortest path calculated, which is basically a sequence of line segments, is the ideal trajectory. Figure 5.1 shows the roadmap and the possible trajectories. In order to use all the nodes in the network, they all must be close enough to any edge in the roadmap. Roadmap should be constructed to guarantee that for each node there exist an edge which is closer than a specified distance. Also, nodes in the roadmap should not be too close to increase efficiency.

Since we assumed that we have no information about the environment, the trajectory might be blocked by an obstacle or a void area. If a node on the trajectory cannot forward the packet, it sends feedback including its location to the source node that the trajectory is not feasible. The edge in the roadmap that corresponds to that location gets a very high weight, so that it is not used again. Another trajectory is generated using the shortest path algorithm.

To provide load balancing, we need to prevent the use of specific nodes more often than the others. This could be done by using different nodes for different trajectories, changing the trajectory after some time for long lasting connections, and by dividing the traffic into several trajectories. While a trajectory is in use, the edges corresponding to that trajectory get their weights increasing in time. The shortest path is periodically recalculated to see if a shorter path exists. This forces the trajectories to be distributed over the area and balances the overall traffic.

The queue sizes of the nodes on a trajectory give important information about the traffic load on it. The nodes with high queue sizes might be located at the crossroad of several trajectories and cause delay. A query packet is sent over the trajectory to collect information about the queue sizes. If the number is high, the corresponding edges on the roadmap are given high weights temporarily. Unlike as in blocked trajectories, these edges are used again after some time.

Probably sharing the roadmap information periodically with the other nodes is the best way to improve this method. Hence, they learn about less used nodes in the network to balance the traffic load and the other trajectories to prevent collisions. However, the transmission cost of this improvement is high.

Chapter 6

Conclusion and Future Work

In this thesis, 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 Trajectory-Based Routing state is minimized. We showed that this approximation problem is NP-complete and is therefore 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 real MANET setting. We first solved the problem with the exhaustive search. This algorithm finds the optimum solution in the discretized space, but takes too much time. Although it is not applicable for this reason, it is a good reference to evaluate the other three algorithms. The genetic algorithm and the longest representation heuristic have close results to the exhaustive search. Their low running times make them good choices for this problem. On the other hand, equal error heuristic does not perform well. We also showed that our problem is customizable for different contexts such as a power-scarce sensor network. Addition to the trajectory approximation problem, we introduced a method to generate ideal trajectories which avoids obstacles and void areas and also helps load balancing.

The work focused in this thesis is only one important part of the whole framework. Inclusion of this work into the larger framework with ideal trajectory planning and trajectory-based forwarding under a real user application is of high interest and merits far than consideration. Regarding the trajectory approximation problem, there are still points to be improved as well. Representations such as B-spline and Bézier curves can be used for more flexible and optimum approximations. There are many ways to improve the trajectory planning method. Parameters such as node density and packet traffic density could be taken into account and the nodes might share their planning information with the others. The whole framework can be tested where there are more than one connection at a time and in mobile scenarios.

Bibliography

[1] Tinyos. `http://www.tinyos.net`.

[2] A. Alippi and G. Vanini. Application-based routing optimization in static/semi-staticWireless Sensor Networks. In *Proceedings of IEEE International Conference on Pervasive Computing and Communications (PERCOM)*, pages 47–51, 2006.

[3] B. T. Loo and J. M. Hellerstein and I. Stoica and R. Ramakrishnan. Declarative Routing: Extensible Routing with Declarative Queries. In *Proceedings of ACM SIGCOMM*, 2005.

[4] S. Basagni, I. Chlamtac, V. Syrotiuk, and B. Woodward. A distance routing effect algorithm for mobility (DREAM). In *Proceedings of ACM/IEEE MobiCom 98*, pages 76–84, October 1998.

[5] Chao Gui and Prasant Mohapatra. Virtual patrol: a new power conservation design for surveillance using sensor networks. In *Proceedings of IEEE International Symposium on Information Processing in Sensor Networks (IPSN)*, 2005.

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

[7] David Chu and Lucian Popa and Arsalan Tavakoli and Joseph M. Hellerstein and Philip Levis and Scott Shenker and Ion Stoica. The Design and Implementation of A Declarative Sensor Network System. In *Proceedings of ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2007.

[8] Dragos Niculescu and Badri Nath. Trajectory-based forwarding and its applications. In *Proceedings of ACM MOBICOM*, September 2003.

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

[10] G. Veltri and Q. Huang and G. Qu and M. Potkonjak. Minimal and Maximal Exposure Path Algorithms for Wireless Embedded Sensor Networks. In *Proceedings of ACM Conference on Embedded Networked Sensor Systems (SenSys)*, November 2003.

[11] Goldberg, David E. *Genetic Algorithms in Search, Optimization & Machine Learning.* Addison Wesley, 1989.

[12] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[13] C. E. Jones, K. M. Sivalingam, P. Agrawal, and J. C. Chen. A survey of energy efficient network protocols for wireless networks. *Wirel. Netw.*, 7(4):343–358, 2001.

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

[15] L. E. Kavraki, P. Svestka, L. E. K. P. Vestka, J. claude Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12:566–580, 1996.

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

[17] Lucian Popa and Afshin Rostami and Richard Karp and Christos Papadimitriou and Ion Stoica. Balancing the Traffic load in Wireless Networks with Curveball Routing. In *Proceedings of ACM MOBIHOC*, 2007.

[18] M. Perillo and W. Heinzelman. DAPR: A protocol for wireless sensor networks utilizing an application-based routing cost. In *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC)*, June 2004.

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

[20] Nikola Milosavljevic and An Nguyen and Qing Fang and Jie Gao and Leonidas Guibas. Landmark Selection and Greedy Landmark-descent Routing for Sensor Networks. In *Proceedings of IEEE INFOCOM*, 2007.

[21] B. Parkinson and J. J. Spiker. *Global Positioning System: Theory and Applications, Volume 1*, volume 163. 1996.

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

[23] C. Perkins. Ad hoc on demand distance vector (aodv) routing. IETF, Internet Draft, draft-ietf-manet-aodv-00.txt, November 1997, 1997.

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

[25] E. Pierluigi Crescenzi, Viggo Kann. A compendium of np optimization problems. *http://www.nada.kth.se/ viggo/problemlist.*

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

[27] P. Sinha, S. Krishnamurthy, and S. Dao. Scalable unidirectional routing with zone routing protocol. In *Proceedings of Wireless Communications and Networking Conference (WCNC)*, pages 1329–1339, 2000.

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

[29] Timothy G. Griffin and Joao L. Sobrinho. Metarouting. In *Proceedings of ACM SIGCOMM*, 2005.

[30] Vinod Muthusamy and Milenko Petrovic and Hans-Arno Jacobsen. Effects of Routing Computations in Content-Based Routing Networks with Mobile Data Sources. In *Proceedings of ACM MOBICOM*, 2005.

[31] William H. Press, Brian P. Flannery, Saul A. Teukolsky, William T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, 2002.

[32] Xiaowei Yang and David Clark and Arthur Berger. NIRA: A New Inter-Domain Routing Architecture. *IEEE/ACM Transactions on Networking (ToN)*, 15(4):775–788, August 2007.

[33] M. Yuksel, J. Akella, S. Kalyanaraman, and P. Dutta. Free-Space-Optical Mobile Ad Hoc Networks: Auto-Configurable Building. Submitted to Wireless Networks (after major revisions), 2006.

[34] M. Yuksel, A. Gupta, and S. Kalyanaraman. Contract-Switching Paradigm for Internet Value Flows and Risk Management. In *Proceedings of IEEE Global Internet Symposium*, 2008.

[35] M. Yuksel, R. Pradhan, and S. Kalyanaraman. An Implementation Framework for Trajectory-Based Routing in Ad-Hoc Networks. *Ad Hoc Networks, Elsevier Science*, 4(1):125–137, January 2006.