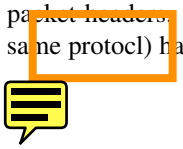
Simulation and evaluation of Bezier-spline Trajectory Based Routing

J. Rick Ramstetter
Rutgers University Dept. of Computer Science
Piscataway, NJ
jramstet@cs.rutgers.edu

April 19, 2010

Abstract

In dense sensor networks, Trajectory Based Routing (TBR) allows a source node to encode in some packet's header a mathematical (e.g. polynomial) curve defining that packet's route. Intermediary nodes make a best-effort to route the packet along this curve. Bezier curves, used widely in computer graphics, allow message trajectories to be described by complex and non-invertable curves. In this work, I implement in Matlab a generalized simulator for TBR protocols in dense sensor networks. Atop that simulator, I implement multiple variations of the same Bezier-spline routing protocol. By using a Bezier-spline representation of trajectories, I am able to examine the tradeoff betweeen a high-degree Bezier curve representation of a desired route (e.g. a single, high degree function piece) and a multi-piece spline (consisting of lower degree or linear pieces) approximating that same high-degree curve. I consider three cases: 1) many-piece-linear splines, 2) many-piece-cubic splines, and 3) single-piece, high-degree curves. Each of these cases is evaluated on multiple variations of the same Bezier-spline protocol. I find that many-piece-cubic splines have the lowest cost in representing the vast majority of tested trajectories, and allow for the smallest-size representation of trajectories in packet headers. I also find that simple optimizations (resulting in the aforementioned variations on the same protocl) have a significant impact on performance and energy usage.

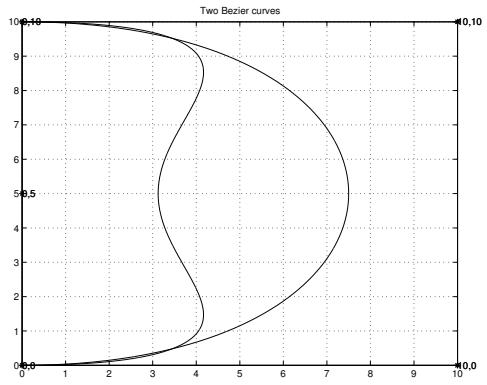


1 Given Problem Statement

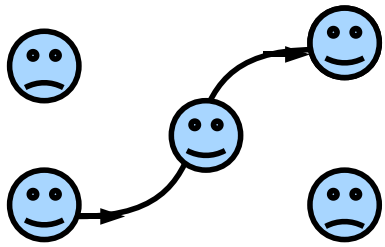
"Rapid geometric forwarding and routing requires rapid selection of the next set of nodes to forward a packet to. In this work, you would implement some of the trajectory based forwarding algorithms using a parametric RPN representation. The goals are two-fold: first, the RPN representation should fit into small packet headers (e.g. 40 bytes or less). Second, the hop selection process should also be very rapid; a few hundred instructions at most. Your RPN parametric equations should be able to describe lines, curves, and more complex structures such as square and hexagonal coverage areas." – Richard Martin

2 Introduction

Trajectory Based Routing (TBR), originally postulated at Rutgers by Badri Nath and Dragos Niculescu, allows a source node to tersely encode in a packet's header a general path that the packet should follow in traveling to its destination. In the past, the only means of accomplishing this was enumeration of individual



(a) Two Bezier curves. The first curve is a cubic curve with control points $(0,0),(10,0),(10,10),(0,10)$. The second curve is a 5^{th} degree curve with control points $(0,0),(10,0),(0,5),(10,10),(0,10)$. The point at $(0,5)$ is doubly weighted. Importantly, a Bezier-curve is only guaranteed to pass through its first and last control points.



(b) An example of using curve-based routing. All smiling nodes receive the message, whereas frowning nodes do not receive the message. The message's header **does not** include an enumeration of the intermediary nodes along the route to its destination. Rather, the header contains a curve, which all intermediary nodes make a best effort in following.

hosts in a packet's header. This is unsuitable for sensor networks, as individual nodes may changes locations (e.g. due to wind) or enter a power-conserving sleep mode. Further, a packet routed across a sensor network may have a large number of network hops to make, resulting in a large packet header size. Trajectory Based Routing solves these problems by utilizing a mathematically defined curve in describing a packet's route. A source node encodes such a curve in a packet's header, and intermediary nodes make a best effort in routing the packet along that curve. Curve based route representations work even as nodes migrate or enter sleep modes; the route is not intrinsically tied to any specific node (or any specific node's location). Likewise, curve based route representations do not necessarily grow in size as the number of intermediary nodes grows.

Bezier curves are one method of representing such a routing path. Widely used in the area of computer graphics, Bezier curves allow specification of curves via functions which are complex and possibly non-invertible. Bezier curves are defined by a set of control points, and are completely contained in the convex hull of these points. A Bezier curve does not necessarily pass through its intermediary control points; it is only guaranteed to pass through the first and lost control points. In a Bezier-spline, each piece of the spline can be a linear function with only two control points, thus causing the spline to pass through all enumerated control points. Bezier curves can be represented via either a linear combination of the Bernstein polynomials or a single, high-degree polynomial.

For example, the following equation can be used to represent a Bezier curve of any degree:

$$Q(t) = \sum_{i=0}^{N} \binom{N}{i} (1-t)^{N-i} t^i P_i \tag{1}$$

...where P_i is control point i, N is the total number of control points, and $\binom{N}{i}$ is the standard Binomial coefficient.

In this work, I have implemented in Matlab a simulation of a Bezier-spline based routing protocol. A Bezier-spline is composed of multiple Bezier curves, each of varying degree. This protocol represents each

spline piece as a linear combination of Bernstein polynomials. The next section contains a full description of the protocol.

3 Protocol

The Bezier-spline routing protocol I use makes the following assumptions:

1. Transmission of some value x from node A to node B is an order of magnitude more costly than B computing the value for itself. Given the rapid pace of technological advances in the semiconductor (e.g. integrated circuit) industry, and the fairly static physical requirements of Radio Frequency transmissions, this assumption should hold well into the future. A similar assumption is made in [1].

2. All nodes are arranged in a 2-dimensional grid of size $WIDTH$ by $HEIGHT$. Nodes are at all times aware of their own location. In the face of node mobility, this can be solved via various localization technologies.

3. Spline pieces are continous at their meeting points. I do not assume derivative continuity at junctions.

The protocol's header is arranged into two distinct *portions*: a fixed length portion and a variable length portion. A key issue in desinging the protocol is deciding on the number of bits to represent each header value with. The protocol assumes that a potentially large number of header fields will be encoded with fixed lengths; these fixed lengths are necessary to ensure a message's receiving node can properly seperate the message header's fields. Though I have performed some simulation testing to verify my choice of parameter-representation-bits, I would not characterize this simulation testing as thorough.

3.1 Fixed length header portion

This is the first data to be received from a remote peer.

1. 16 bits representing a sending node's t, where t is the same as in equation (1). To be clear, the t received by some node B will be the t transmitted by some sender node A. When B attempts to forward a message to node C, B will include his t in the message.

2. 8-bit checksum of the sending node's (e.g. the received) t value.

3.2 Variable length header portion

This follows the fixed length header immediately, and can have arbitrary length.

1. 16 bits encoding the number of control points which define a spline piece.

2. 16 bits encoding the maximum value of t for which a spline piece is relevant.

3. 16 bits encoding a checksum for the **entire** spline piece, including all control points.

4. 15 bits encoding the X axis coordinate for node P_i.

5. 15 bits encoding the Y axis coordinate for node P_i.

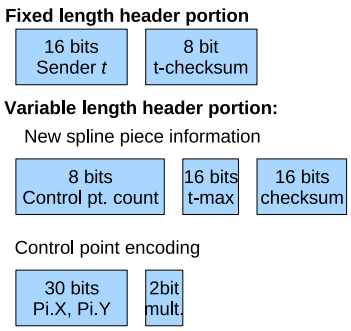


Figure 1: Illustration of the protocol."Sender t" is the input value to Q used by a sending node. The first "checksum" is a 3 bit hash of the received t value. The "Control pt. count" specifies the number of control points in a spline piece. The value t_{max} encodes the maximum values of t for which a spline piece is relevant. The second "checksum" is a per-piece checksum; each spline piece thus has an associated checksum. $P_{i,X}$ and $P_{i,Y}$ encode the X and Y axis locations of a spline piece's control point i. "Mult" encodes the multicplity of that control point. The values $P_{i,X}$, $P_{i,Y}$, and $mult$ are repeated for all N control point in a spline piece.

6. 2 bits encoding the multiplicity of control point P_i. Setting this value to 0 causes the point to be ignored.

Many spline pieces can be flexibly defined inside the variable length portion of the protocol header. The 8 bit "control point count" field encodes the number of control points in a piece. Given that each control point occupies 32 bits of space, a spline piece's length in bits can be easily computed as $(\text{control point count}) * 32 + 16 + 16$.

For a spline piece j, each of its control points are encoded one after another, starting with $i = 0$ and counting up to $i = (\textit{control point count})$. Thus, there will be 2 or more occurences of $P_{i,X}$, $P_{i,Y}$, and $P_{i,mult}$ for each of the spline's pieces. When a receiving node finds that the number of control points received (e.g. i) has reached its maximum value (e.g. *control point count*), the receiving node assumes that the definition of another spline piece follows.

Given the above layout of the header, the protocol represents a 3-piece Bezier-spline as follows:

$$Q_{spline}(t) = \begin{cases} P_{1,i} = \sum_{i=0}^{N_1} \binom{N_1}{i} (1-t)^{N_1-i} t^i P_{1,i} & \text{for } 0 < t \leq t_{1,max} \\ P_{2,i} = \sum_{i=0}^{N_2} \binom{N_2}{i} (1-t)^{N_2-i} t^i P_{1,i} & \text{for } t_{1,max} < t \leq t_{2,max} \\ P_{3,i} = \sum_{i=0}^{N_3} \binom{N_3}{i} (1-t)^{N_3-i} t^i P_{1,i} & \text{for } t_{2,min} < t \leq t_{3,max} \end{cases} \quad (2)$$

In the above equation, the motivation for requiring continuity at boundary points between splines becomes clear. To be specific, requiring continuity allows $t_{i,max}$ to be re-used as $t_{i+1,min}$, thereby saving 8 bits of header space.

This routing protocol can be used for routing based on explicit node enumeration. A Bezier curve must pass through its first and last control points. As such, a spline consisting of many 2-control-point Bezier curves is essentially an enumeration of nodes along a route.

3.3 Intermediary node next-hop decisions

In this protocol, intermediary nodes are expected to make a best effort in routing a message along the path (e.g. spline or curve) specified by the message's originator. This requires that intermediary nodes have access to a cost function which can be used in choosing which node the message should next be forwarded to.

I have evaluated five cost functions for this protocol:

1. **Minimum Distance from Curve (MDC)**: For every neighboring node x within radial transmission range, compute at all time samples t_i a distance from the curve for node x. Forward the packet to the in-range node with the minimum distance from the curve at any sample $t_{new} > t_{current}$ (where $t_{current}$ is the current node's minimal-curve-distance sample).

2. **Limited Minimum Distance from Curve (LMDC)**: similar to MDC but labels as out of transmission range nodes whose minimum distance to the curve $Q(t)$ occurs at a sample t_i where $t_i \leq t_{current}$. Of the remaining nodes, forward the packet to the in-range node with the minimum distance from the curve at any sample $t_{new} > t_{current}$.

3. **Limited Minimum Distance from Curve 2 (LMDC2)**: The same as LDMC, but nodes will not allow the packet to move along the negative gradient of curve $Q(t)$.

4. **Sample Bounded LMDC2 (SBLMDC2)**: similar to LMDC, but limits the number of samples (or increments of t) a message can make between any two nodes. This can force a better fit to very curvy trajectories. The parameter α specifies the limit of the number of sample increments.

5. **Most Advance on Curve (MAC)**: For the set of neighbors found in LMDC, choose the node which advances furthest along the curve (e.g. choose the highest in-range t_i).

A future cost function would be Closest to Centroid of LMDC Set (CCLMDC). That is, find the centroid of the set of neighbors left visible in LMDC2, and forward the packet to the node that is closet to the centroid at a sample t_{new} where $t_{new} > t_{current}$.

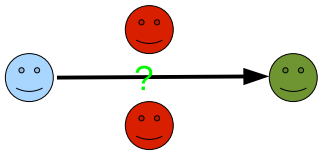


Figure 2: Illustration of a quandary faced in choosing a protocol cost function. The blue node has received a message. The header of that message indicates the message should next be moved in the direction of the green node. However, the blue node must find an intermediary node (one of the two red nodes) to forward the message to, as the green node is not directly reachable. The two red nodes are equidistant from the desired route; the blue node must pick between them.

3.4 The checksum

I use a Longitudinal Redundancy Check (LRC) as a hash of the various header fields. The LRC checksum works by laying out the bits of all header fields in their expected order, and breaking these bits into groups

of a fixed size c. These created groups do not necessarily reflect the seperation of encoded fields in the headers; an individual field (e.g. an X coordinate for some control point) may have its bits split across multiple groups. The LRC, a binary value of length c, is calculated by taking the exclusive-or of all header groups. Undersized groups are padded with zeros to bring them to length c.

In this protocol, only the checksum of the t value must be updated at intermediary nodes. The checksums included in the definition of each spline piece does not change at intermediary nodes, as the pieces themselves do not change at intermediary nodes.

3.5 Example header

The header for a Bezier-spline with 2 spline pieces, each piece having three control points, will look as follows:

1. 16 bits encoding the sender's t value
2. 8 bit checksum of sender's t value
3. 8 bits encoding the count of control points in the first piece (0x011 in this case)
4. 16 bits encoding the maximum t for which the first spline piece is relevant
5. 16 bits checksum of the entire first piece, including all control points
6. 15 bits encoding the X coordinate for the first piece's first control point
7. 15 bits encoding the Y coordinate for the first piece's first control point
8. 2 bits encoding the multiplicy for the first piece's first control point
9. 15 bits encoding the X coordinate for the first piece's second control point
10. 15 bits encoding the Y coordinate for the first piece's second control point
11. 2 bits encoding the multiplicy for the first piece's second control point
12. 15 bits encoding the X coordinate for the first piece's third control point
13. 15 bits encoding the Y coordinate for the first piece's third control point
14. 2 bits encoding the multiplicy for the first piece's third control point
15. 8 bits encoding the count of control points in the second piece (0x011 in this case)
16. 16 bits encoding the maximum t for which the second spline piece is relevant
17. 16 bit checksum of the entire second piece, including all control points
18. 15 bits encoding the X coordinate for the second piece's first control point
19. 15 bits encoding the Y coordinate for the second piece's first control point
20. 2 bits encoding the multiplicy for the second piece's first control point

21. 15 bits encoding the X coordinate for the second piece's second control point
22. 15 bits encoding the Y coordinate for the second piece's second control point
23. 2 bits encoding the multiplicy for the second piece's second control point
24. 15 bits encoding the X coordinate for the second piece's third control point
25. 15 bits encoding the Y coordinate for the second piece's third control point
26. 2 bits encoding the multiplicy for the second piece's third control point

The total number of bytes transmitted by this header is 40

4 Simulator

In evaluating the above protocol, I have implemented a simulation thereof in MATLAB. Specifically, I have implemented a general purpose simulator for Trajectory Based Routing protocols. Evaluation of another TBR protocol atop this simulator would require only translation of the protocol's properties to the simulator's expected input, though I have not yet attempted to do this. For example, a plausible next step might be evaluation of a protocol which represents routing curves as a sum-of-sinusoids.

4.1 Evaluation metrics

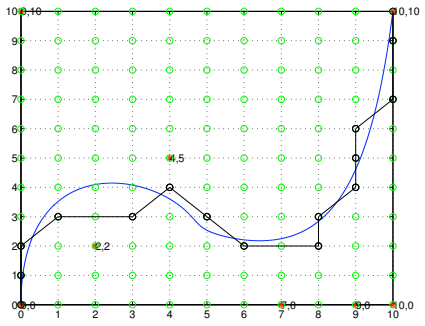
The simulator I have developed evaluates Trajectory Based Routing on two simple performance metrics. The first metric is "number of bits transmitted globally." This count considers all transmissions made by all nodes in the system. Because I assume that tranmissions are an order of magnitude more costly than computation, the "bits transmitted" metric can be roughly equated with power consumption aggregated across all nodes.

The second metric attempts to characterize the accuracy of intermediary nodes in routing a message along its encoded trajectory. Although an intermediary nodes X is assumed to make a best-effort attempt at forwarding a message along its encoded trajectory, such forwarding decisions are made on a loca basis, using only information available at X. These local, per-node forwarding decisions do not necessarily combine to form a globally optimum path. This metric can be summarized as follows:

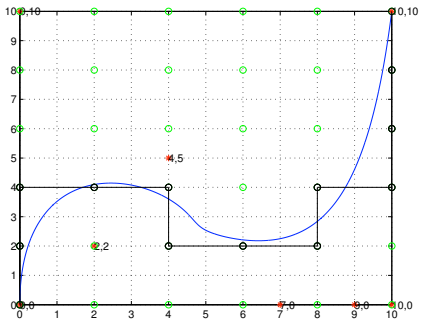
$$S = \sum_{t=0}^{t=1000} \sqrt{\left(Q_x(\frac{t}{1000}) - A_x(\frac{t}{1000})\right)^2 + \left(Q_y(\frac{t}{1000}) - A_y(\frac{t}{1000})\right)^2} \tag{3}$$

... where S, t, B, Q, and A take on the following meanings:

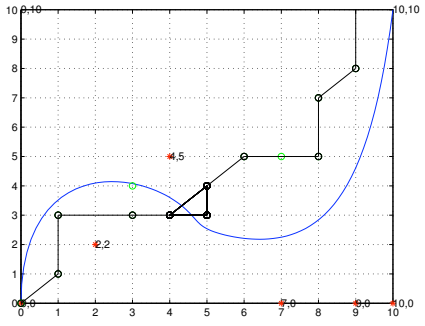
- S: The score. A lower score is better.
- t: Time as taken from 0 to 1000. The 1000 was chosen arbitrarily, and is reflective of a step size of $\frac{1}{1000}$. As t increases by some fixed step size, a message should be routed forward along its trajectory by intermediary nodes.
- $Q_i(t_2)$: The desired position of a message at time t_2 along axis i. For the protocol outlined in section 3, this would be the output of the spline function along axis i at time t_2.
- $A_i(t_2)$: The actual position of a message at time t_2 along axis i. For a densely populated network, this should approximate $Q_i(t_2)$.



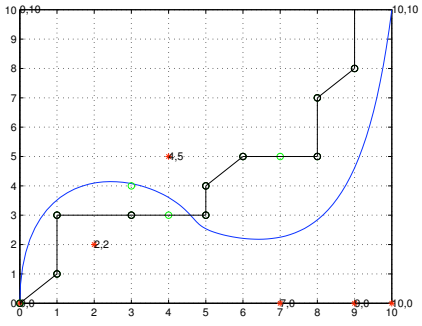
(a) A simulation run using LMDC. Nodes (green) are equally spaced at 1 unit apart along both axes. The Bezier curve $Q(t)$ is drawn in blue, and its control points in red. The black line represents the path a message takes from $(0, 0)$ to $(10, 10)$ in attempting to follow $Q(t)$. Control points are at (0,0), (0,10), (10,0), (2,2), (4,5), (7,0), (9,0), and (10,10). Nodes are able to transmit in a radius of 2.



(b) The same curve and node transmission range, but with nodes disperesed every two spaces.



(c) A simulation run using LMDC. Nodes are unevenly spaced. When the packet arrives at (5,4), the best next-hop would be (6,5). However, LMDC picks (5,5), then (4,3), then (5,4). This repeats until the increasing $t_{current}$ value causes $Q(t)$'s output to become quite close to the destination node (10,10). The message takes 41 hops to arrive at the desination.



(d) The same curve and node coverage using the LMDC2 algorithm. The algorithm converges in 12 hops.

Figure 3: Some results

Method	Global bytes sent	Hops required	Score (see subsection 4.1)
Minimum Distance from Curve (MDC)	1824	48	127.5
Limited MDC (LMDC)	1558	41	126.7
Limited MDC 2 (LMDC2)	456	12	92.03
Sample Bounded LMDC2 (SBLMDC2), $\alpha = 5$	456	12	92.03
Most Advancement on Curve (MAC)	342	9	140.3

Figure 4: Results using the curve, node dispersion, and broadcast range as described in 3(c). Nodes are at (0,0), (1,1), (1,3), (3,3), (3,4), (4,3), (5,3), (5,4), (6,5), (8,5), (7,5), (8,7), (9,8), (9,10), and (10,10). Control points are at (0,0), (0,10), (10,0), (2,2), (4,5), (7,0), (9,0), and (10,10). Nodes are able to transmit in a radius of 2.

Method	Global bytes sent	Hops required	Score (see subsection 4.1)
Minimum Distance from Curve (MDC)	608	16	23.5
Limited MDC (LMDC)	570	15	42.8
Limited MDC 2 (LMDC2)	456	12	92.03
Sample Bounded LMDC2 (SBLMDC2), $\alpha = 5$	456	12	94.5
Most Advancement on Curve (MAC)	398	12	90.3

Figure 5: Results using the curve, node dispersion, and broadcast range as described in 3(c). Nodes are equally spaced at 1 unit apart along both axes. Control points are at (0,0), (0,10), (10,0), (2,2), (4,5), (7,0), (9,0), and (10,10). Nodes are able to transmit in a radius of 2.

Future metrics might include:

1. Maximum distance from desired curve.
2. Average maximum distance from desired curve, considering only distances above a threshold.
3. Routing crossovers: the number of times the route a message follows crosses itself.
4. Power consumption: total power consumption of all nodes, including transmissions and processing.
5. Least remaining battery life, an attempt to quantify the effect of many messages following the same route.

References

[1] Dragos Niculescu and Badri Nath. Trajectory based forwarding and its applications. In *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 260–272, New York, NY, USA, 2003. ACM.