Multi-Hop Dynamic Geographic Routing

Julio C. Navas and Tomasz Imielinski

Computer Science Department

Rutgers University

Piscataway, NJ 08855

{navas,imielins}@cs.rutgers.edu

1 Introduction.

Geographic Routing is a *new process* for routing data packets through an internetwork of computers. Our process distinguishes itself from the current routing technology by using arbitrary geographical regions (denoted by longitude and latitude) instead of logical computer addresses as the criteria to route a packet. The purpose of this new technology is to extend the capabilities of the current IP internetwork by adding a new geographic message protocol to the current IP suite of protocols (which currently consists of UDP and TCP). This geographic protocol is many-to-many and connectionless

This is an *enabling technology*. This technology would be used by the application programmers and not by the common end- user. We provide an Application Programming Interface (API) that a programmer would use to interface with the geographic routing protocol in order to send and receive messages.

The system is designed to give the user as much flexibility as possible in designating the destination geographical region. Ideally, the user would interact with a graphical user interface that would contain a geographic map of the desired destination area. The user would then draw a bounded polygon around the targeted region and designate a message to send to that region. The application would then translate that polygon from the screen coordinates to geographic coordinates, which use longitude and latitude, and then utilize the API to send the message to those coordinates. The geographic routing system would now transport the message from geographic-router to geographic-router in a multi-hop fashion until the networks in the destination area are reached. At this point, the routers will broadcast or multicast the message to everyone within the target area.

Currently, we are in the process of evaluating a prototype implementation and setting up an experimental network capable of routing the geographic messages. This network would be deployed in stages: first throughout the Computer Science Department, then the Rutgers University network, and, finally, to participating universities and government/military facilities. This is currently a DARPA-sponsored Integrated Technology Demonstration (ITD) within the GloMo I (Global Mobile Information Systems) program.

2 Geographic Routing Methods

The Geographic Routing Methods use the geographic destination polygon in the geographic message header directly for routing. Geographic routing is will eventually be implemented in the Internet Protocol (IP) Network layer but, currently, is implemented in the Application layer in a manner similar to the way multicast routing was first implemented. At this time, the geographic routing prototype system acts as a virtual network that is overlaid onto the current IP internetwork and uses geographic addresses for routing. We accomplished this by creating our own software routers that understand geography. These routers use IP tunnels to transport data packets through areas that do not support geographic routing.

The router keeps a cache of the next-hop destinations of the most recent geographic message packets. When a router receives a geographic message packet, it will use the incoming packet's sender IP address and destination polygon together as a key into the cache. If this is not the first packet to arrive for this destination and if the timer on the cache entry has not yet expired, then the cache will return a list of all of the next hop addresses to which copies of the packet must be sent.

Experiments and evaluations performed on the prototype geographic routers and reported in [Mobicom'97] show that the main bottleneck (and, hence, the main research area) lies in how the next-hop list is created.

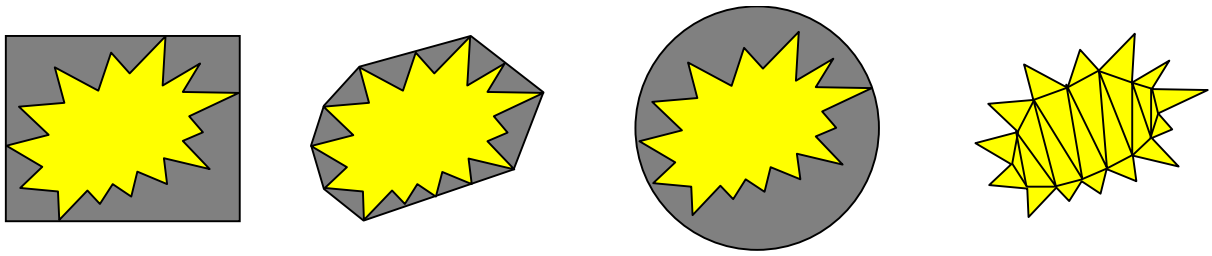
2.1 Approximation and Search Tree Method

The key attributes of this method are:

- No assumptions about network architecture. Automatic configuration into flat network model.
- Algorithm find the shortest paths the sender to all of the receivers
- All polygons are represented by simpler *approximations*.
  - Computational cost along each path is low.
  - Polygon approximation causes the creation of a routing tree that is overly large. Hence, extra branches must be pruned.
- Routing table is stored as a type of *range tree*, which is a type of search tree.
  - When forwarding a packet, the router can quickly find in $O(k + \log^2 n)$ steps (where n is the number of routing table entries and k is the number of destination candidates) only those routers that are destination candidates and ignore the rest.

This method of geographic routing differs from the version described in [Mobicom'97], the Intersection and Linear List method, in a few distinct ways. First, it does not make any assumptions about the architecture of the network, whereas the previous method relied on a hierarchical structure. The routers in this method can automatically configure themselves into a flat network architecture and can be manually configured hierarchically. Second, the original destination polygon is not used for routing purposes. Instead, a simpler approximation of the polygon is used that drastically reduces the computational overhead by reducing the number of vertices in the polygon. Thirdly, the routing table data structure has been changed to efficiently accommodate a larger number of routing table entries. Whereas the previous method used a linear list, this method adapts range trees, which are a type of search tree from computational geometry, to use with two-dimensional geography. Finally, this newer version of the geographic routing methods always finds the shortest path to all destinations.

2.1.1 Approximating a polygon



(A) (B) (C) (D)

Figure 1 Four methods to approximate a complex polygon. (A) Rectangle (B) Convex Hull (C) Smallest Enclosing Circle (D) Triangulation

Polygon intersection is a computationally expensive procedure whose cost linearly increases with the number of vertices in the polygons. This computational expense is reflected in the high routing times reported in [Mobicom'97]. By reducing the number of vertices involved by approximating the polygons with simpler polygons that have fewer vertices, the computational cost of the intersection could be reduced. For example, large cost savings could be realized

by approximating a 100-vertex polygon (with a routing time of 19,000 microseconds) with a 20-vertex polygon (with a routing time of 6,500 microseconds). Also, polygon intersection routines only work with convex or star-shaped polygons and not with concave polygons. Yet, the system should not restrict the type of polygon that the user enters. An approximation of the polygon should be constructed so that the approximation is always convex even though the underlying polygon may be concave. Furthermore, if a standard type of polygon can be used for the approximations, then more advanced computational geometry techniques, such as range trees (see section 2.1.2) could be employed instead of intersections. Caution must be used, however, to ensure that any approximation contains the original polygon or packets will not be delivered to destinations that should have received it. Because of the loss of specificity, packets may be forwarded to nodes that shouldn't have received it in the first place.

The four different polygon approximation techniques that have been evaluated are the Bounding Rectangle, Convex Hull, Smallest Enclosing Circle, and Triangulation. See Figure 1. Note that all four approximations have the property that the resulting approximation is always convex. Use of one of these approximation techniques involves trade-offs between the computational expense of creating the approximation, the accuracy of the approximation, and how easily the approximation lends itself to use by range tree methods. See Table 1.

Method	Cost to Create	Accuracy	Lends itself to use by Range Trees?
Bounding Rectangle	O(n)	Poor. An overly large routing tree is created which may require much pruning.	Yes, no change necessary.
Smallest Enclosing Circle	O(n)	Poor. An overly large routing tree is created which may require much pruning.	Almost. A bounding rectangle must be created, but this can be done in constant time.
Convex Hull	O(n log n)	Close. The intersection times are higher than the Bounding Rectangle or the Smallest Enclosing Circle, however, the resulting routing tree is closer to optimal and doesn't need much pruning.	Not easily. A bounding rectangle must be created for use in a range tree.
Triangulation	O(n log n)	Exact. Routing tree created is optimal. The intersection times are higher than the Bounding Rectangle or the Smallest Enclosing Circle, however. Also, the intersection times will always be linear with respect to the number of triangles.	Not easily. A bounding rectangle must be created for use in a range tree.

Table 1: Comparison of different polygon approximation techniques.

2.1.2 Range Tree Methods

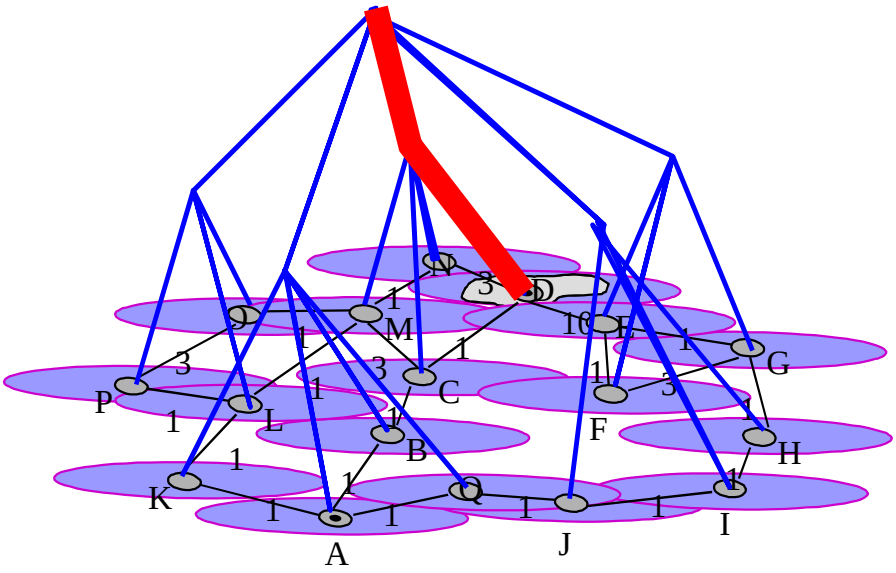


Figure 2: Searching through a Range Tree to discover the next routing hop(s)

One of the pitfalls of the Intersection and Linear List method is that the cost of routing a packet increases linearly with the size of the routing table. Intuitively, what is wanted is something similar to a binary search tree, but that can handle the two dimensional data of geography. Use of a search tree data structure would allow the routing algorithm to find the candidate destinations in $O(k + \log^2 n)$ time, where n is the number of routing table entries and k is the number of destination candidates, by automatically ignoring those table entries that do not apply. See Figure 2. In other words, the cost of routing a packet increases only logarithmically-squared with the size of the routing table. Range-search methods from computational geometry were adapted for use in geographic routing. In particular, variants of the range tree method, called point sets and interval sets were tried.

2.1.2.1 Using Point Sets

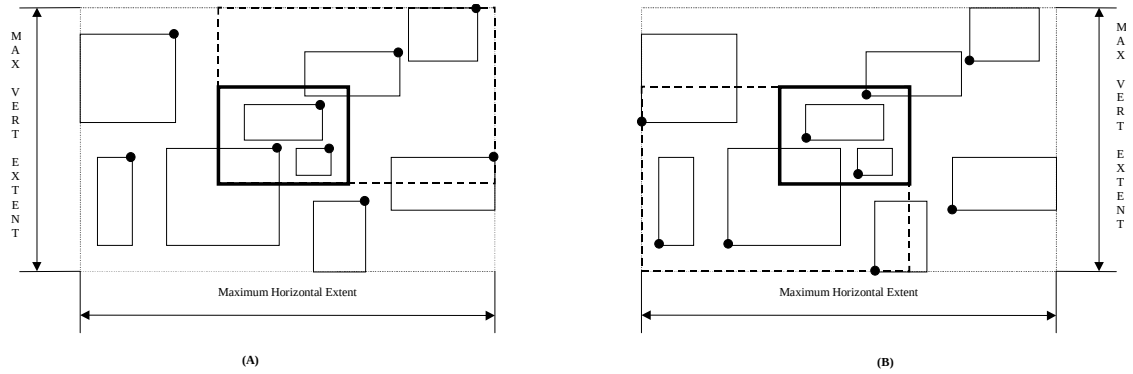


Figure 3: Using Point Sets to Discover Destination Candidates. The destination polygon is depicted with a thick bold line. A large dark circle highlights the points being tested. A dashed rectangle depicts the rectangular query that is taking place.

An instance S of a Point Set is a collection of items where each item is a two-tuple, <*p*,*i*>, where *p* is a two-dimensional point that is used as the key and *i* is the routing table information. The number of items in S is called the

size of S. A point set of size zero is said to be empty. Note that for each point *p* there is at most one item <*p,i*> in S. Point sets allow us to do rectangular range queries for all points within the rectangle. For the rectangle, <*x0,y0,x1,y1*>, a rectangular query would return all items <*p,i*> in S with *x0* <= *p.x* <= *x1* and *y0* <= *p.y* <= *y1*. The operations insert, lookup, and delete take time $O(\log^2 n)$ and range searching takes time $O(k + \log^2 n)$, where k is the size of the returned list of items and n is the current size of the point set. The space requirement is O(n log n).

What is needed is a way to find the bounding rectangles of those routing table entry polygons that intersect the bounding rectangle of the destination polygon. Since point sets work with two-dimensional points and since we are using bounding rectangles, two point sets are necessary in order to find the candidate destinations. The first point set stores the upper right hand corners of the bounding rectangles of each routing table entry's polygon (PS_{UR}). The second point set stores the lower left hand corners of the bounding rectangles (PS_{LL}).

In order to find those routing table entry polygons whose bounding boxes intersect with the destination polygon's bounding box the following algorithm is used:

Given:

- The bounding rectangle of the destination polygon, *D*, with upper right hand corner point, p_{UR}^{D}, and lower left hand corner point, p_{LL}^{D}.
- A point set, PS_{UR}, containing each routing table entry's (*Ri*) bounding rectangle's upper right hand corner points , p_{UR}^{Ri}.
- A point set, PS_{LL}, containing each routing table entry's (*Ri*) bounding rectangle's lower left hand corner points , p_{LL}^{Ri}.
- The rectangle enclosing all of the routing table entries, *U*, with upper right hand corner point, p_{UR}^{U}, and lower left hand corner point, p_{LL}^{U}.

Algorithm:

1. A rectangular query is performed on the point set PS_{UR} to find the list of *Ri* 's whose points p_{UR}^{Ri} lie within the rectangle with lower left-hand point p_{LL}^{D} and upper right hand point p_{UR}^{U}. This list will be called L_{UR}. See Figure 3 (A).
2. A rectangular query is performed on the point set PS_{LL} to find the list of *Ri* 's whose points p_{LL}^{Ri} lie within the rectangle with lower left hand point p_{LL}^{U} and upper right hand point p_{UR}^{D}. This list will be called L_{LL}. See Figure 3 (B).
3. Find list of destination candidates, $L_{DC} = L_{UR} \bigcap L_{LL}$.

2.1.2.2 Using Interval Sets

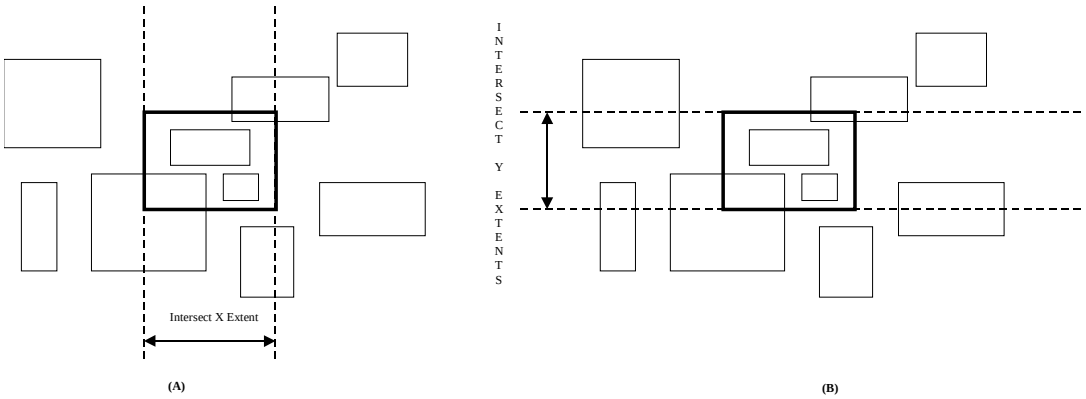


Figure 4: Using Interval Sets to Discover Destination Candidates. The destination polygon is depicted with a thick bold line. The interval between the dashed lines depicts the interval query that is taking place.

An instance S of an Interval Set is a collection of items <*x,y,i*>, where *[x,y]* denotes the interval that is used as the key and *i* denotes the routing table information. *x* is called the left boundary of the item and *y* is called the right boundary. The number of items in S is called the size of S. An interval set of size zero is said to be empty. For each interval *[x,y]* there is at most one item <*x,y,i*> in S. Interval Set allow us to do interval range queries, *[a,b]*, which return the list of all items <*x,y,i*> in S with *[x,y]* $\bigcap$ *[a,b] !=q emptyset*. The operations insert, lookup, and delete take time $O(\log^2 n)$ and range searching takes time $O(k + \log^2 n)$, where k is the size of the returned list of items and n is the current size of the point set. The space requirement is O(n log n).

Just like in the Point Set, what is needed is a way to find the bounding rectangles of those routing table entry polygons that intersect the bounding rectangle of the destination polygon. Since interval sets work with one-dimensional linear intervals and we are working with two-dimensional bounding rectangles, two interval sets are necessary in order to find the candidate destinations. The first interval set stores the extents along the x-axis of the bounding rectangles of each routing table entry's polygon (IS_X). The second interval set stores the extents along the y-axis of the bounding rectangles of each routing table entry's polygon (IS_Y).

In order to find those routing table entry polygons whose bounding boxes intersect with the destination polygon's bounding box the following algorithm is used:

Given:

- The bounding rectangle of the destination polygon, *D*, with the minimum x-coordinate of $p^D_{X\min}$, maximum x-coordinate of $p^D_{X\max}$, minimum y-coordinate of $p^D_{Y\min}$, and maximum y-coordinate of $p^D_{Y\max}$.
- A interval set, IS_X, containing each routing table entry's (*Ri*) bounding rectangle's minimum $p^{Ri}_{X\min}$ and maximum $p^{Ri}_{X\max}$ x-coordinates.
- A interval set, IS_Y, containing each routing table entry's (*Ri*) bounding rectangle's minimum $p^{Ri}_{Y\min}$ and maximum $p^{Ri}_{Y\max}$ y-coordinates.

Algorithm:

1. Perform an interval range query on the interval set IS_X and find the list, L_X, of all items < $p^{Ri}_{X\min}$, $p^{Ri}_{X\max}$, R_i> *with* [$p^{Ri}_{X\min}$, $p^{Ri}_{X\max}$] $\bigcap$ [$p^D_{X\min}$, $p^D_{X\max}$] *!= q emptyset*. See Figure 4 (A).

2. Perform an interval range query on the interval set IS_Y and find the list, L_Y, of all items $< p^{Ri}_{Y\,\min}, p^{Ri}_{Y\,\max}, R_i >$ *with* $[\, p^{Ri}_{Y\,\min}, p^{Ri}_{Y\,\max}\,] \bigcap [\, p^{D}_{Y\,\min}, p^{D}_{Y\,\max}\,]$ *!= q emptyset*. See Figure 4 (B).

3. Find list of destination candidates, $L_{DC} = L_X \bigcap L_Y$.

Note that Interval Sets have the added advantage of being able to be extended to three dimensions easily by simply adding one more interval set and interval range query. The Point Set, on the other hand, would require another two point sets and two more rectangle range queries.

2.1.3 GeoRIP, Shortest Path Routing Trees, and Pruning

Before geographic routers can determine where to forward an incoming packet, they must first have a routing table that contains information about the network topology. Several protocols already exist for discovering the network topology and automatically configuring a routing table. These protocols can be adapted to distribute a router's geographic location information in addition to the other information already passed along. In particular, for the purposes of this project, the popular RIP was augmented to include geographic location information. Using this protocol, which we call GeoRIP (see section 3.1.2), a router will have a routing table entry for each router in the network. Each entry will contain information on a router's geographic location, IP address, the shortest number of intermediate routers (hops) between this router and the current router, and the next router (or next hop) on the path from the current router to this entry's router.

When forwarding an incoming packet, a router uses the routing table information to determine which next-hop routers should be sent a copy of the packet. First, all of the final destinations for the packet are discovered. Then a list of the next-hop routers for these final destinations is created. Note that the list of next-hop routers may be smaller than the list of final destinations because several final destinations may share the same next-hop router. Care is taken not to send duplicate packets to the same next-hop router.

When a geographic message has been forwarded all of the way from the sender to all of the receivers, the routers will have created a *shortest-path routing tree* with the root at the sender and the leaves at all of the receivers. If the exact destination polygon is used, then the routing tree will be optimal. See Figure 5.

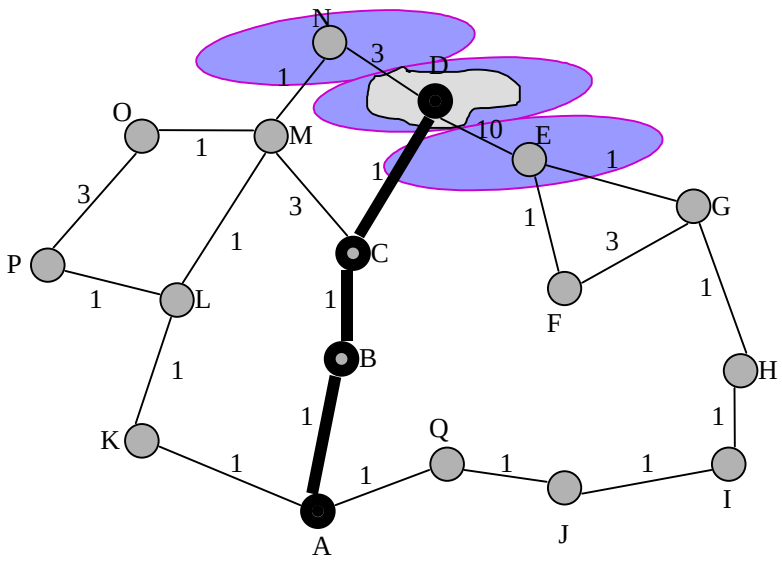


Figure 5: Routing through an internetwork using the original polygon. The sender at A is sending a message to a geographic region (rough polygon in gray) within D's cell. Since the exact polygon was used to determine the

next hop at each router, the exact routing tree from the sender to the receiver is created. The bold black lines denote the routing tree.

However, if an approximation to the destination polygon is used, then the resulting routing tree will contain extra erroneous branches that must be pruned. If a router detects that no downstream routers are viable destinations for a message and that it itself is not a destination of the message, then it will send a *prune* message upstream toward the source of the message. When a router receives a prune message for a specific geographic message, it removes that downstream router from the routing cache entry (if any) for that geographic message. If this causes the number of downstream routers to drop to zero, then the router will itself send a prune message upstream. See Figure 6.

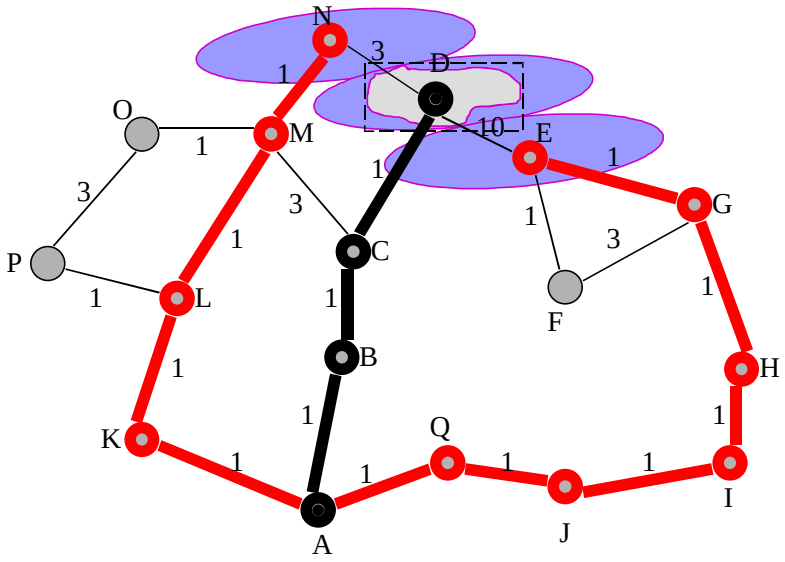


Figure 6: Pruning an inexact routing tree. The sender at A is sending a message to the geographic region (rough polygon in gray) within D's cell. The destination polygon has been approximated by a rectangle and this has caused the routing tree to include branches to N and E (marked in red). These branches must be pruned in order to reduce the tree to the optimal routing tree (shown in bold black).

3 Experimental Verification.

3.1 Prototype Implementation

The prototype implementation has the following attributes:

- Multi-hop geographic routing
  - Application-layer solution.
  - Known to work on Linux and Solaris.
- Auto-configuration into flat network (GeoRIP – see section 3.1.2)
  - dynamically adjust to network topology perturbations
  - good for campus-wide network (100-200 routers)
- Incrementally deployable. Able to tunnel through networks that do not support geographic routing.
- API for developers

3.1.1 The Components of the Prototype Geographic Routing System

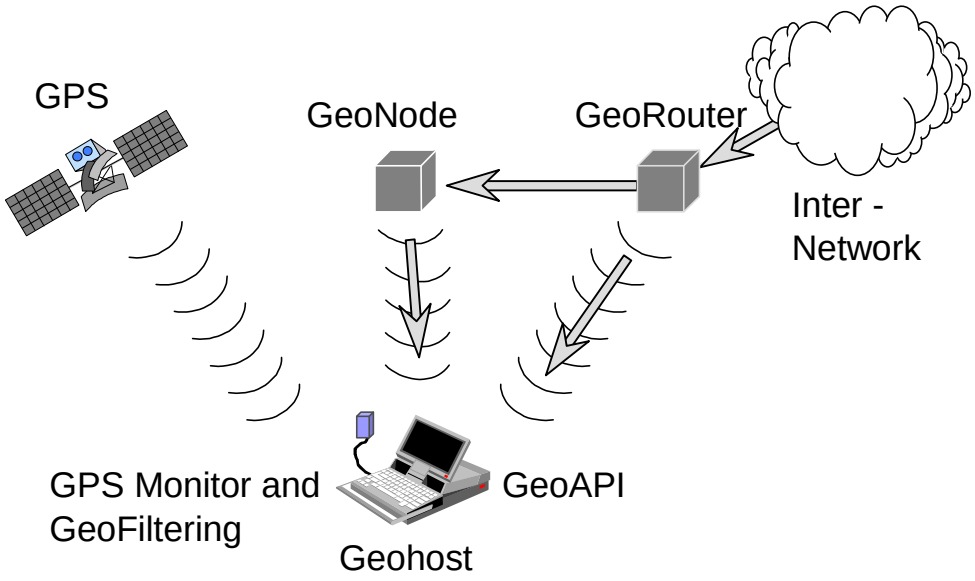


Figure 7- Components of the Geographic Routing Prototype

The system is composed of four main components: GeoHosts, GeoAPI, GeoNodes, and GeoRouters.

The GeoHost is located on all computer hosts that are capable of receiving and sending geographic messages. Its role is to notify all client processes about the availability of geographic messages, the host computer's current geographic location, the address of the local GeoNode, and the address of the local GeoRouter. If a GPS device is present in the mobile, the GeoHost will monitor that device and continually update its notion of the mobile host's location. Additionally, with the added accuracy of the GPS device, the GeoHost can further filter the messages that it receives and only accept those that directly correspond to its location. This may be necessary in those cases where the wireless base station covers a large geographical area.

The GeoAPI is the set of software library routines that allow a programmer to create applications that can send and receive geographic messages.

A GeoNode is a buffer for messages with lifetimes. The main function of the GeoNode is to store incoming geographic messages (which have lifetimes greater than zero) for the duration of their lifetimes and to periodically multicast them on all of the subnets or wireless cells to which it is attached. Each subnet and each wireless cell will have at most one GeoNode. The sender of the message specifies the lifetime of a geographic message. Message lifetimes may be necessary because the receivers of geographic messages may be mobile and may possibly arrive at the message destination some time after the geographic message first arrives.

Since, most likely, there will be several geographic messages residing in a GeoNode at one time, the multicasting of the various messages will be scheduled. The scheduling algorithm will take into account the size of the message, the priority of the message, and the speed of the subnet's transport medium. Clients wishing to receive geographic messages would then tune in to the appropriate multicast group to receive them.

Geographic routers (GeoRouter) are in charge of moving a geographic message from a sender to a receiver. GeoRouters are essentially routers which are geographically aware. Each router is charged with performing geographic routing functions for those networks to which it is directly attached. GeoRouters keep track of the geographic area that they service (called its service area) by calculating the union of the geographic areas covered by the networks attached to it. Its service area is represented as a single simple closed polygon whose vertices are denoted by geographic coordinates. GeoRouters build their routing tables by exchanging service area polygons.

3.1.2 Geographic Routing Information Protocol (GeoRIP)

3.1.2.1 Brief Description of the Routing Information Protocol (RIP)

RIP is a protocol in a series of routing protocols based on the Bellman-Ford (or distance vector) algorithm. This algorithm has been used for routing computations in computer networks since the early days of the Internet. The particular packet formats and protocol described here are based on the program "routed", which is included with the Berkeley distribution of Unix. It has become a de facto standard for exchange of routing information among gateways and hosts. It is implemented for this purpose by most commercial vendors of IP gateways. Note, however, that many of these vendors have their own protocols that are used among their own gateways.

RIP is intended for use within the IP-based Internet. The Internet is organized into a number of networks connected by gateways. The networks may be either point-to-point links or more complex networks such as Ethernet or the Internet. Hosts and gateways are presented with IP datagrams addressed to some host. Routing is the method by which the host or gateway decides where to send the datagram. It may be able to send the datagram directly to the destination, if that destination is on one of the networks that are directly connected to the host or gateway. However, the interesting case is when the destination is not directly reachable. In this case, the host or gateway attempts to send the datagram to a gateway that is nearer the destination. The goal of a routing protocol is very simple: it is to supply the information that is needed to do routing.

3.1.2.2 Format of a GeoRIP packet

GeoRIP is essentially the same as RIP, except that geographic location information is included with the routing information that is passed between routers. The major addition to RIP is the change in the routing information packet format. The format of a GeoRIP packet is as follows:

- Command (1 byte)
- Version (for GeoRIP, version = 3) (1 byte)
- Number of Entries (2 bytes)

Followed by one or more of the following:

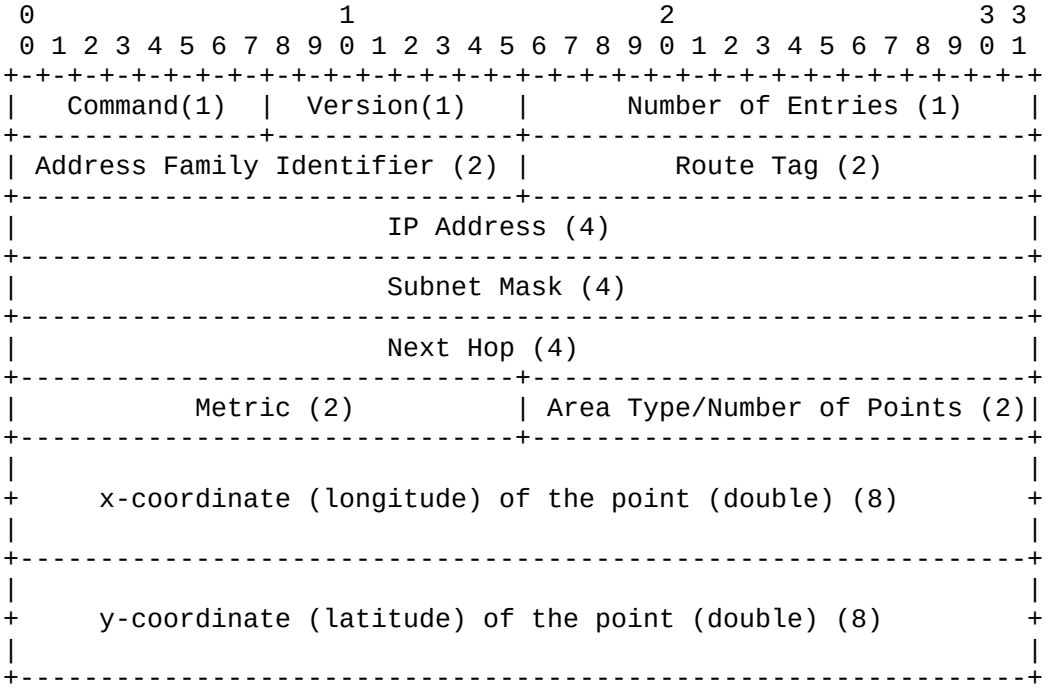
- An authentication section:
  - 0xFFFF (all bits set to 1) (2 bytes)
  - Authentication type (2 bytes)
  - Authentication (16 bytes)

- A route entry:
  - Address Family Identifier (2 bytes)
  - Route Tag (2 bytes)
  - IP Address (4 bytes)
  - Subnet Mask (4 bytes)
  - Next Hop (4 bytes)
  - Metric (2 bytes)
  - Area Type/Number of Points (2 bytes)

Followed by one of the following according to the Area Type:

- Type = 1 (Point)
  - Latitude (8 bytes)
  - Longitude (8 bytes)
- Type = 2 (Circle)
  - Latitude (8 bytes)
  - Longitude (8 bytes)
  - Radius (8 bytes)
- Type >= 3 (Polygon)
  - Bounding Box Bottom Left
    - Latitude (8 bytes)
    - Longitude (8 bytes)
  - Bounding Box Top Right
    - Latitude (8 bytes)
    - Longitude (8 bytes)
  - Latitude[i] (8 bytes)
  - Longitude[i] (8 bytes)

For example, if a GeoRIP packet contained one routing entry whose geographic area was described by a point, the packet would look like the following:



In this diagram, field sizes are given in octets. Unless otherwise specified, fields contain binary integers in normal network order with the most-significant octet first. Each tick mark represents one bit. The IP Address is the usual 4-octect IPv4 Internet address in network order.

3.1.3 How the Routing Table is Implemented

The routing table, which is the heart of the geographic router, is comprised of two main areas: a cache and a polygon describing the router's service area.

3.1.3.1 Cache

In order to allow for rapid access to the individual cache items while, at the same time, not wasting memory space by preallocating a set hash table size, the router's cache is implemented as a dynamic hash table with chaining. Individual cache items are really a three-tuple < (sender IP address, destination polygon), list of next hops, time-stamp >. The tuple, (sender IP address, destination polygon), that is copied from the geographic message packet header is used as the key into the hash table. The list of next hops is a list of those child or parent nodes to which this router should forward a copy of the packet. Finally, the time-stamp indicates the age of the cache item. The time-stamp is updated every time that the cache item is used. The cache is periodically checked for stale cache items. Stale cache items are those items whose time-stamp is older than some threshold. For the purposes of these experiments, the maximum age of a cache item is thirty seconds and the cache is checked every fifteen seconds for stale items.

3.1.3.2 Calculating a Router's Service Area

A router's service area is, theoretically, the union of the service areas of its child nodes. However, such a union may produce a polygon which has internal "holes" and which may have sections that are concave. Many algorithms that perform polygon intersection, however, require simple polygons that are convex. Therefore, instead of calculating the union of the child node areas, the convex hull was calculated instead. In order to calculate the convex hull, all of the polygonal and circular descriptions of the child node areas were first converted into a list of points. For circles, geometry was used to find ten equidistant points on the circle's perimeter.

3.1.4 Addressing Model

Two-dimensional GPS positioning offers latitude and longitude information as a two dimensional vector, < latitude, longitude >, where longitude ranges from -180 (west) to 180 (east), and latitude ranges from -90 (south) to 90 (north). Thus < 40.48640, -74.44513 > is an example of the GPS coordinates for the town of New Brunswick, New Jersey, U.S.A.

Currently, for ease-of-implementation reasons, we are using double-precision floating-point numbers to represent the longitude and latitude coordinates of all points.

In the future, once we have more experience using GPS receivers and are able to determine the optimal number of significant digits necessary for geographic routing, we will most likely switch to fixed-point 32-bit integers because of their greater computational efficiency.

3.1.4.1 Using a Geographic Destination Address

A geographic destination address would be represented by some closed polygon such as:

- point
- circle(center point, radius)
- polygon(point(1), point(2), ... , point(n-1), point(n), point(1))

where each vertex of the polygon is represented using geographic coordinates. This notation would be used to send a message to anyone within the specified geographical area defined by the closed polygon.

For example, if we were to send a message to city hall in Fresno, California, we could send it by specifying the geographic limits of the city hall as a series of connected lines that form a closed polygon surrounding it. Therefore the address of the city hall in Fresno could look like: polygon([36.80,-119.80], [36,85,-119.76], ...).

3.1.4.2 Format of a Geographic Message Header

A geographic message header has the following format and order:

- Version (1 byte)
- Flags (1 byte)
- TTL (1 byte)
- Priority (1 byte)
- Sender IP Address (4 bytes)
- Destination multicast or unicast IP Address (4 bytes)
- Destination Port Number (2 bytes)
- Sender Port Number (2 bytes)
- Lifetime (2 bytes)
- Area Type/Number of Points (2 bytes)

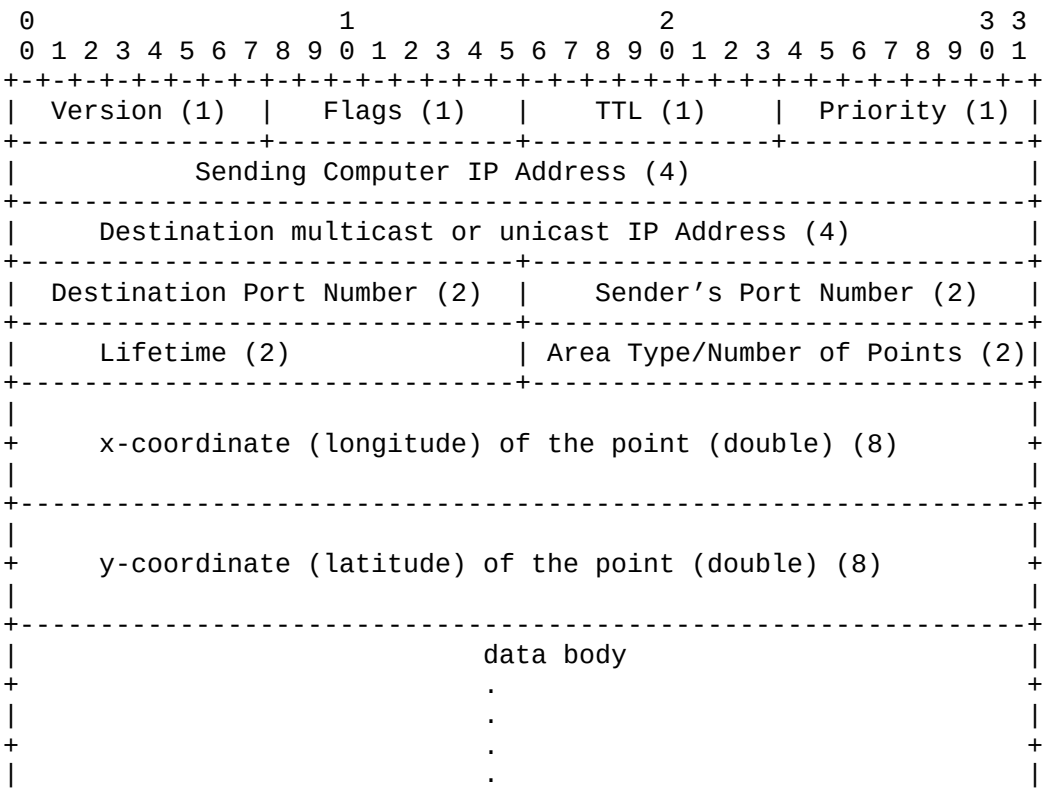
followed by one of the following according to the Area Type:

- Type = 1 (Point)
  - Latitude (8 bytes)
  - Longitude (8 bytes)
- Type = 2 (Circle)
  - Latitude (8 bytes)
  - Longitude (8 bytes)
  - Radius (8 bytes)
- Type >= 3 (Polygon)
  - Bounding Box Bottom Left
    - Latitude (8 bytes)
    - Longitude (8 bytes)
  - Bounding Box Top Right
    - Latitude (8 bytes)

- ❖ Longitude (8 bytes)
- Latitude[i] (8 bytes)
- Longitude[i] (8 bytes)

Followed by the message data body of arbitrary length and content.

For example, if a point describes the geographic destination, the header would look like the following:



In this diagram, field sizes are given in octets. Unless otherwise specified, fields contain binary integers in normal network order with the most-significant octet first. Each tick mark represents one bit. The Sending Computer IP Address is the usual 4-octect Internet address in network order.

3.1.5 Geographic Message Application Programming Interface (GeoAPI)

The GeoAPI allows programmers to access the functionality of the geographic messaging system. The API is a C++ library which is divided into two main classes: GeoMesg and GeoSocket. The GeoMesg class allows a programmer to create and manipulate geographic messages. The GeoSocket class allows programmers to create and manipulate IP sockets which can send and receive geographic messages.

3.1.5.1 GeoMesg: Geographic Message Class

3.1.5.1.1 Packet Manipulation

```
  virtual bool          MessageParse();
  virtual bool          ShapeParse();

  virtual bool          MessageCreate();

  virtual char *        ExtractRawPolygon( int & size );
```

3.1.5.1.2 Polygon Bounding Box Access

```
  virtual Point&        GetBottomLeft();
  virtual Point&        SetBottomLeft( Point & op );

  virtual Point&        GetTopRight();
  virtual Point&        SetTopRight( Point & op );
```

3.1.5.1.3 Header Fields Access Functions

```
  virtual char *        GetData( int *size = NULL );
  virtual char *        SetData( char *d, int size );

  virtual u_char        GetVersion();
  virtual u_char        SetVersion( char l );

  virtual u_char        GetPriority();
  virtual u_char        SetPriority( char l );

  virtual u_char        GetFlags();
  virtual u_char        SetFlags( char l );

  virtual ShapeType     GetType();
  virtual ShapeType     SetType( ShapeType l );

  virtual u_short       GetPort();
  virtual u_short       SetPort( short l );

  virtual u_short       GetLifetime();
  virtual u_short       SetLifetime( short l );


  virtual struct in_addr& GetDestAddr();
  virtual struct in_addr& SetDestAddr( struct in_addr& ia );

  virtual struct in_addr& GetSenderAddr();
  virtual struct in_addr& SetSenderAddr( struct in_addr& ia );


  virtual Shape*        GetShape();
  virtual Shape*        SetShape( Shape& sh );

  virtual void          Clear();
```

```
  virtual unsigned char *GetPacket(); // for backward compatability only
  virtual unsigned char *SetPacket( unsigned char *pkt, int sz );

  virtual int           GetSize();
```

3.1.5.2 Geographic Socket Class

3.1.5.2.1 Pan-Message Access Functions

```
  virtual u_short       GetPort();
  virtual u_short       SetPort( short l );

  virtual struct in_addr& GetDestAddr();
  virtual struct in_addr& SetDestAddr( struct in_addr& ia );

  virtual struct in_addr& GetSenderAddr();

  virtual Shape*        GetShape();
  virtual Shape*        SetShape( Shape& sh );
```

3.1.5.2.2 Socket End-point(s) Functions

```
  virtual bool          Connect( Shape & sh,
                                 short portnum );
  virtual bool          Connect( Shape & sh,
                                 short portnum,
                                 in_addr& maddr );
  virtual bool          Bind();
  virtual bool          Bind( short portnum );
  virtual bool          MulticastJoin( in_addr maddr );
  virtual bool          MulticastDrop( in_addr maddr );
```

3.1.5.2.3 Send/Receive Packets using the GeoMesg class

```
  virtual int           Read  ( GeoMesg& pi );
  virtual int           Recv  ( GeoMesg& pi );
  virtual int           RecvFrom( GeoMesg& pi );

  virtual int           Write ( GeoMesg& pi );
  virtual int           Send  ( GeoMesg& pi );
  virtual int           SendTo( GeoMesg& pi );

  virtual int           Read  ( char *buf, int size );
  virtual int           Recv  ( char *buf, int size );
  virtual int           RecvFrom( Shape & sh,
                                 u_short& portnum,
                                 in_addr& maddr,
                                 char *buf,
                                 int size );

  virtual int           Write ( char *buf, int size );
```

```
virtual int           Send  ( char *buf, int size );
virtual int           SendTo( Shape & sh,
                              u_short& portnum,
                              in_addr& maddr,
                              char *buf,
                              int size );
```

3.1.5.2.4 GeoHost / GeoNode communication commands

```
list< msg_info >      GetAllMsgsInfo();
msg_info              GetSpecMsgInfo( in_addr addr );
point                 GetPos();
bool                  RecvSpecGeoMsg( GeoMesg& gm,
                                      in_addr& multi_addr,
                                      u_short portnum );
int                   RecvGeoMsg( GeoMesg& gm );

iosockinet *          RecvFromGeoStart();
bool                  HandleDaemonMessage();
int                   RecvFromGeoEnd( GeoMesg& gm );
```

4 Future

We are consulting with Charles Hedrick and the RUCCS about deploying geographic routing throughout Rutgers. Initial inquiries have been made about the possibility of establishing geographic routers in U.C. Berkeley, U.C. Santa Cruz, and CECOM in Ft. Monmouth, New Jersey.

5 Related Work.

Linking an IP Address with a geographical location has been of interest for quite some time already. The recent redesign of the Internet Protocol (IP) (described in RFC-1883) and the advent of the Global Positioning System gave a new stimulus for this work.

The first serious attempt to relate IP-addresses to geographical locations was the UUMAP project in 1985 [uunet85]. This project attempted to collect in a central flat-file database the geographical locations of all of the computer hosts then on the Internet. However, because of the subsequent speed with which computers were added to the Internet and the difficulties of maintaining such a large central database, erroneous data found its way into the database. Some attempts have been made to decentralize the database through the use of UUCP-Zones

Later, two separate groups [RFC-1712] [RFC-1876], tried similar approaches based on the Domain Name System [RFC-1035]. These systems improved on the UUMAP project by decentralizing the geographic location information, thus relieving the burden on the system administrators and helping to ensure an up-to-date and correct database. In both cases, the DNS data-structure that contains the host computer information, the RR records (described in RFC-1101), were augmented with new fields to contain geographical location information in the form of longitude and latitude.

However, in [RFC-1712] and [RFC-1876], the DNS system can only return a geographic location given an IP address. In our work, we strive to perform the reverse function, which is to return IP addresses given a geographical location.

The first attempt to design a system that actually routes packets according to their geographic destination and the work that is closest to ours is Cartesian Routing by Gregory G. Finn at the University of Southern California [Finn87]. Cartesian Routing was an attempt to alleviate foreseeable future routing and addressing problems in large and dense metropolitan internetworks. The proposed solution called for point-to-point routing based on the geographic location of the source, the intermediate routers, and the destination. A new type of address was proposed for Cartesian routing

which would be a two-tuple: < location, id >, where location represents a geographic point defined by latitude and longitude, and id is an identifier which is unique across the entire network. In order to perform routing, all Cartesian routers record the geographic locations of all other directly connected routers. Then, in order to route a packet from a source to a destination, the router selects the neighbor router that is closest to the destination and sends the packet there. However, it is possible that no neighbor routers lie in the same direction as the destination. In this case, the router will search for another router at most n-hops away that makes progress toward the destination. It does this by using a flooding technique. If no router is found within n-hops, then the packet is considered undeliverable. Simulation studies were performed that showed that Cartesian Routing performed only marginally worse than regular IP routing in terms of hop counts. Our work attempts to provide the more general multi-point to multi-point geographic routing and to actually test its feasibility by implementing and evaluating a prototype geographic router.

In the proposed redesign of IP [RFC-1884], IP address type space was specifically allocated for geographic addresses. IP addresses would be assigned to subnets and hosts based on topological criteria, such as geography. The sender of a "geographic message" would be unicasting messages only to such hosts that have geographic addresses. Geographic routing attempts to provide the more general ability of sending a message to all recipients within a geographical area, regardless of whether or not the hosts have geographical addresses.

Recently, Ernst et. al. have developed a geographic radio broadcasting system [Ernst97]. This is a system for determining whether information broadcast by a general transmitter is relevant to a particular user based on the user's location, velocity, and/or time. One or more of the following would describe each message: a segment comprising a geographic region, a velocity, a time corresponding to an event, an event specific tag. The selection criteria may also include event specific tags. The receiver at the remote terminal receives the messages from the transmitter at the general broadcasting unit. A navigational receiver may also be used to acquire navigational information from an appropriate external source, such as GPS. A matching processor at the remote terminal evaluates the segment in the messages and determines if the segment sufficiently matches the stored selection criteria. If a match is found, then the message is disseminated.

[Ernst97] differs from geographic routing in that it provides no multi-hop capabilities. It assumes that all receivers are within range of the single transmitter. With geographic routing, as long as the receivers are within range of any transmitter, the messages will be forwarded, from the sender to the receivers through possibly several intermediate routers until they reach their destinations.

Teare and Walker used location as the key to limit access to a remote database [Teare93]. Their system used a GPS receiver to gain time-correlated data of a remote mobile node's actual position. The mobile node would then, in some manner, securely transmit the information as a position history to a central facility. At the central facility, a comparison is made between the received position history and predetermined signature data representing acceptable time-position histories. If a positive match is detected, a decryption key associated with the matched history is forwarded to the mobile node for decoding of the encrypted programming material that is also sent to it.

The method described in [Teare93], however, makes the remote database a bottleneck and a point-of-failure. Geographic routing, on the other hand, is a completely distributed and general routing system. It is an enabling technology which allows any information to be carried over it – unlike [Teare93] which is limited to data which can be stored in a database.

In [Mardus92], Mardus describes a method for route-selective reproduction of digitally encoded traffic announcements broadcast by a transmitter to a vehicle receiver, the announcements being decoded by the receiver. A comparison of the route-specific characteristics in the broadcast announcements with characteristics of the receiver's trip route is performed. If the characteristics agree to a predetermined extent, the driver is provided with the traffic announcement applicable to him, via a visual and/or acoustical output device. Route-specific characteristics include road types and their numerical designations, major route segments, and shorter route segments. The advantage of this system is that the driver is not distracted by a great number of traffic advisories that are not relevant for him.

Similar to [Teare93], [Mardus92] employs only a single transmitter and, thus, suffers the same bottleneck and point-of-failure problems. Furthermore, [Mardus92] does not make efficient use of the network bandwidth. It transmits all of the information and lets the receivers filter out the information that belongs to them. Geographic routing does a better job of efficiently using the available bandwidth by constructing a shortest-path routing tree from each sender to all of the receivers. In this manner, the information travels only by the most direct route and only to the intended receivers.

6 References.

[Mobicom'97] Julio C. Navas and Tomasz Imielinski, "Geographic Addressing and Routing", *Proc. of the Third ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'97)*, Budapest, Hungary, September 26-30 1997.

[Ernst97] Darrell E. Ernst, Donald R. McMillan, Thomas L. Folk, Steven R. Rogers, *Location based selective distribution of generally broadcast information*, U.S. Patent 5636245, assignees: The Mitre Corporation, McLean, VA, issued: June 3 , 1997

[Mardus92] Claus Mardus, *Method and apparatus for route-selective reproduction of broadcast traffic announcements*, U.S. Patent 5095532, assignees: Robert Bosch GmbH, Stuttgart, Federal Republic of Germany, issued: Mar. 10, 1992

[Teare93] Melvin J. Teare, Stephen S. Walker, *Location-sensitive remote database access control*, U.S. Patent 5243652, assignees: GTE Laboratories Incorporated, Waltham, MA, issued: Sep. 7 , 1993.

[Finn87] G. Finn, *Routing and Addressing Problems in Large Metropolitan-scale Internetworks*, ISI Research Report ISI/RR-87-180, University of Southern California, March 1987.

[RFC 1876] C. Davis,P. Vixie,T. Goodwin,I. Dickinson, *A Means for Expressing Location Information in the Domain Name System*, RFC 1876, University of Warwick, January 1996.

[uunet85] ---, *UUCP Mapping Project*, Software available via anonymous FTP from ftp.uu.net., 1985.

[RFC 1712] C. Farrell, M. Schulze, S. Pleitner, D. Baldoni, *DNS Encoding of Geographical Location*, RFC 1712, Curtin University of Technology, November 1994.

[RFC-1884] Deering, S., and Hinden, R., Editors, *IP Version 6 Addressing Architecture*, RFC 1884, Ipsilon Networks, Xerox PARC, December 1995.