Location Based Routing For

Distance Nodes

Submitted By-

TARANG GUPTA (13BCE0352)

VIVEK SHARMA (13BCE0)

Submitted To-

PROF. ABDUL GAFFAR

1. **INTRODUCTION**

A location based routing is a collection of mobile nodes that are dynamically and arbitrarily located in such a manner that the interconnections between nodes are capable of changing on a continual basis. The primary goal of this routing is correct and efficient route establishment between a pair of nodes so that messages may be delivered in a timely manner. LAR is an on-demand protocol who is based on the DSR (Dynamic Source Routing). The Location Aided Routing protocol uses location information to reduce routing overhead of the ad-hoc network! Normally the LAR protocol uses the GPS (Global Positioning System) to get these location information. With the availability of GPS, the mobile hosts knows there physical location.

The networks are a new paradigm of wireless communication for mobile hosts (which we call nodes). In this routing, there is no fixed infrastructure such as base stations or mobile switching centers. Mobile nodes that are within each other’s radio range communicate directly via wireless links, while those that are far apart rely on other nodes to relay messages as routers. Node mobility in an ad hoc network causes frequent changes of the network topology. Figure 1 shows such an example: initially, nodes A and D have a direct link between them. When D moves out of A’s radio range, the link is broken. However, the network is still connected, because A can reach D through C, E, and F. Military tactical operations are still the main application of ad hoc networks today. For example, military units (e.g., soldiers, tanks, or planes), equipped with wireless communication devices, could form an ad hoc network when they roam in a battlefield. The networks can also be used for emergency, law enforcement, and rescue missions. Since an ad hoc network can be deployed rapidly with relatively low cost, it becomes an attractive option for commercial uses such as sensor networks or virtual classrooms.

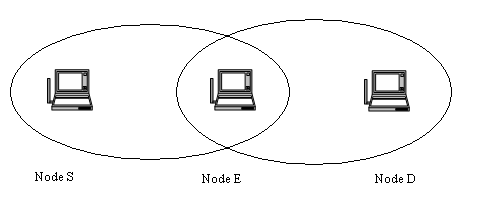
1. **PROPOSED SYSTEM**

**Location Based Routing**

The concept behind these infra-structureless networks is the collaboration between its participating members, i.e, instead of making data transit through a fixed base station, nodes consequentially forward data packets from one to another until a destination node is finally reached. Typically, a packet may travel through a number of network points before arriving at its destination.

Location Based Routing introduces a completely new flavor of network formation. The routers and hosts are free to move randomly and organize themselves in an arbitrary fashion, thus the network topology changes rapidly and unpredictably. Absence of a supporting structure in mobile ad-hoc networks, to a certain extent, invalidates almost all of the existing techniques developed for routine network controls in the existing wireless networks.

A MANET consists of mobile platforms (e.g., a router with multiple hosts and wireless communications devices)--herein simply referred to as "nodes"--which are free to move about arbitrarily. The nodes may be located in or on airplanes, ships, trucks, cars, perhaps even on people or very small devices, and there may be multiple hosts per router. A MANET is an autonomous system of mobile nodes. The system may operate in isolation, or may have gateways to and interface with a fixed network.



**Infra-structure less Network**

Here the node S wants to communicate to node D. The oval indicates the communication range of the node. The communication range of S does not exceed to include D. In this case routing is necessary, node E is in the range of S which has D in its range. So S in order to communicate to D, first sends the message to E which inturn forwards it to D. Thus the node E acts as a router and a node.

1. **System Requirements**
   1. **HARDWARE SPECIFICATION**

**Processor** : Any Processor above 500 Mhz.

**Ram** : 128Mb.

**Hard Disk** : 10 Gb.

**Compact Disk** : 650 Mb.

**Input device** : Standard Keyboard and Mouse.

**Output device** : VGA and High Resolution Monitor.

**3.2 SOFTWARE SPECIFICATION**

**Language** : Java

**Front End Tool** : Swing

**Operating System :** Windows 98.

1. **SOFTWARE ENVIRONMENT**

WHAT IS JAVA?

Java has two things: a programming language and a platform.

Java is a high-level programming language that is all of the following:

Simple Architecture-neutral

Object-oriented Portable

Distributed High-performance

Interpreted multithreaded

Robust Dynamic

Secure

Java is also unusual in that each Java program is both compiled and interpreted. With a compile you translate a Java program into an intermediate language called Java byte codes the platform-independent code instruction is passed and run on the computer.

Compilation happens just once; interpretation occurs each time the program is executed. The figure illustrates how this works.

**Java Program**

**Compilers**

**Interpreter**

**My Program**

You can think of Java byte codes as the machine code instructions for the Java Virtual Machine (Java VM). Every Java interpreter, whether it’s a Java development tool or a Web browser that can run Java applets, is an implementation of the Java VM. The Java VM can also be implemented in hardware.

Java byte codes help make “write once, run anywhere” possible. You can compile your Java program into byte codes on my platform that has a Java compiler. The byte codes can then be run any implementation of the Java VM. For example, the same Java program can run Windows NT, Solaris, and Macintosh.

JAVA PLATFORM

A platform is the hardware of software environment in which a program runs. The Java platform differs from most other platforms in that it’s a software only platform that runs on the top of other, hardware-based platform. Most other platforms are described as a combination of hardware and operating system.

The Java platform has two components:

The Java Virtual Machine (Java VM)

The Java Application Programming Interface (Java API)

You’ve already been introduced to the Java VM. It’s the base for the Java platform and is ported onto various hardware-based platforms.

The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets.

The Java API is grouped into libraries (package) of related components. The next sections, what can Java do? Highlights each area of functionally provided by the package in the Java API.

The following figure depicts a Java program, such as an application or applet, that’s running on the Java platform. A special kind of application known as a server serves and supports clients on a network. Examples of the servers include Web Servers, proxy servers, mail servers, print servers, and boot servers. Another specialized program is a Servlet. Servlets are similar to applets in that they are runtime extensions of the application. Instead of working in browsers, though, servlets run with in Java Web Servers, configuring of tailoring the server.

How does the Java API support all of these kinds of programs? With packages of software components that provide a wide range of functionality. The API is the API included in every full implementation of the platform.

The core API gives you the following features:

The Essentials: Objects, Strings, threads, numbers, input and output, datastructures, system properties, date and time, and so on.

Applets: The set of conventions used by Java applets.

Networking: URL’s TCP and UDP sockets and IP addresses.

Internationalization: Help for writing programs that can be localized for users

Worldwide programs can automatically adapt to specific locates and be displayed in the appropriate language.

JAVA PROGRAM

* Java API
* Java Virtual Machine
* Java Program
* Hard Ware

API and Virtual Machine insulates the Java program from hardware dependencies. As a platform-independent environment, Java can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and Just-in-time-byte-code compilers can bring Java’s performance close to the native code without threatening portability.

WHAT CAN JAVA DO?

However, Java is not just for writing cut, entertaining applets for the World Wide Web (WWW). Java is a general purpose, high-level programming language and a powerful software platform. Using the fineries Java API,you can write many types of programs.

The most common types of program are probably applets and application, where a Java application is a standalone program that runs directly on the Java platform.

Security:

Both low-level and high-level, including electronic signatures, public/private key management, accesses control, and certificate.

## Networking

## Introduction

This article is about a client/server multi-threaded socket class. The thread is optional since the developer is still responsible to decide if needs it. There are other Socket classes here and other places over the Internet but none of them can provide feedback (event detection) to your application like this one does. It provides you with the following events detection: connection established, connection dropped, connection failed and data reception (including 0 byte packet).

## Description

This article presents a new socket class which supports both TCP and UDP communication. But it provides some advantages compared to other classes that you may find here or on some other Socket Programming articles. First of all, this class doesn't have any limitation like the need to provide a window handle to be used. This limitation is bad if all you want is a simple console application. So this library doesn't have such a limitation. It also provides threading support automatically for you, which handles the socket connection and disconnection to a peer. It also features some options not yet found in any socket classes that I have seen so far. It supports both client and server sockets. A server socket can be referred as to a socket that can accept many connections. And a client socket is a socket that is connected to server socket. You may still use this class to communicate between two applications without establishing a connection. In the latter case, you will want to create two UDP server sockets (one for each application). This class also helps reduce coding need to create chat-like applications and IPC (Inter-Process Communication) between two or more applications (processes). Reliable communication between two peers is also supported with TCP/IP with error handling. You may want to use the smart addressing operation to control the destination of the data being transmitted (UDP only). TCP operation of this class deals only with communication between two peers.

**Analysis of Network Client Server**

### TCP/IP stack

The TCP/IP stack is shorter than the OSI one:



TCP is a connection-oriented protocol; UDP (User Datagram Protocol) is a connectionless protocol.

### IP datagram’s

The IP layer provides a connectionless and unreliable delivery system. It considers each datagram independently of the others. Any association between datagram must be supplied by the higher layers. The IP layer supplies a checksum that includes its own header. The header includes the source and destination addresses. The IP layer handles routing through an Internet. It is also responsible for breaking up large datagram into smaller ones for transmission and reassembling them at the other end.

### UDP

UDP is also connectionless and unreliable. What it adds to IP is a checksum for the contents of the datagram and port numbers. These are used to give a client/server model - see later.

### TCP

TCP supplies logic to give a reliable connection-oriented protocol above IP. It provides a virtual circuit that two processes can use to communicate.

### Internet addresses

In order to use a service, you must be able to find it. The Internet uses an address scheme for machines so that they can be located. The address is a 32 bit integer which gives the IP address. This encodes a network ID and more addressing. The network ID falls into various classes according to the size of the network address.

### Network address

Class A uses 8 bits for the network address with 24 bits left over for other addressing. Class B uses 16 bit network addressing. Class C uses 24 bit network addressing and class D uses all 32.

### Subnet address

Internally, the UNIX network is divided into sub networks. Building 11 is currently on one sub network and uses 10-bit addressing, allowing 1024 different hosts.

### Host address

8 bits are finally used for host addresses within our subnet. This places a limit of 256 machines that can be on the subnet.

### Total address



The 32 bit address is usually written as 4 integers separated by dots.

### Port addresses

A service exists on a host, and is identified by its port. This is a 16 bit number. To send a message to a server, you send it to the port for that service of the host that it is running on. This is not location transparency! Certain of these ports are "well known".

### Sockets

A socket is a data structure maintained by the system to handle network connections. A socket is created using the call socket. It returns an integer that is like a file descriptor. In fact, under Windows, this handle can be used with ReadFile and WriteFile functions.

#include <sys/types.h>

#include <sys/socket.h>

int socket(int family, int type, int protocol);

Here "family" will be AF\_INET for IP communications, protocol will be zero, and type will depend on whether TCP or UDP is used. Two processes wishing to communicate over a network create a socket each. These are similar to two ends of a pipe - but the actual pipe does not yet exist.

**Create a server socket that listens for a client to connect**

socket(int af, int type, int protocol)

This method creates the socket

bind(SOCKET s, const struct sockaddr FAR \* name, int namelen)

Associates a local address with a socket This routine is used on an unconnected datagram or stream socket, before subsequent connects or listens. When a socket is created with socket, it exists in a name space (address family), but it has no name assigned. bind establishes the local association (host address/port number) of the socket by assigning a local name to an unnamed socket. In the Internet address family, a name consists of several components. For SOCK\_DGRAM and SOCK\_STREAM, the name consists of three parts: a host address, the protocol number (set implicitly to UDP or TCP, respectively), and a port number which identifies the application. If an application does not care what address is assigned to it, it may specify an Internet address equal to INADDR\_ANY, a port equal to 0, or both. If the Internet address is equal to INADDR\_ANY, any appropriate network interface will be used; this simplifies application programming in the presence of multi- homed hosts. If the port is specified as 0, the Windows Sockets implementation will assign a unique port to the application with a value between 1024 and 5000. The application may use getsockname after bind to learn the address that has been assigned to it, but note that getsockname will not necessarily fill in the Internet address until the socket is connected, since several Internet addresses may be valid if the host is multi-homed. If no error occurs, bind returns 0. Otherwise, it returns SOCKET\_ERROR, and a specific error code may be retrieved by calling WSAGetLastError.

listen(SOCKET s, int backlog )

Establishes a socket to listen to a incoming connection To accept connections, a socket is first created with socket, a backlog for incoming connections is specified with listen, and then the connections are accepted with accept. listen applies only to sockets that support connections, i.e. those of type SOCK\_STREAM. The socket s is put into "passive'' mode where incoming connections are acknowledged and queued pending acceptance by the process. This function is typically used by servers that could have more than one connection request at a time: if a connection request arrives with

the queue full, the client will receive an error with an indication of WSAECONNREFUSED. listen

attempts to continue to function rationally when there are no available descriptors. It will accept connections until the queue is emptied. If descriptors become available, a later call to listen or accept will re-fill the queue to the current or most recent "backlog'', if possible, and resume listening for incoming connections.

accept(SOCKET s, struct sockaddr FAR \* addr, int FAR \* addrlen)

This routine extracts the first connection on the queue of pending connections on s, creates a new socket with the same properties as s and returns a handle to the new socket. If no pending connections are present on the queue, and the socket is not marked as non- blocking, accept blocks the caller until a connection is present. If the socket is marked non-blocking and no pending connections are present on the queue, accept returns an error as described below. The accepted socket may not be used to accept more connections. The original socket remains open. The argument addr is a result parameter that is filled in with the address of the connecting entity, as known to the communications layer. The exact format of the addr parameter is determined by the address family in which the communication is occurring. The addrlen is a value-result parameter; it should initially contain the amount of space pointed to by addr; on return it will contain the actual length (in bytes) of the address returned. This call is used with connection-based socket types such as SOCK\_STREAM. If addr and/or addrlen are equal to NULL, then no information about the remote address of the accepted socket is returned.

closesocket(SOCKET s)

closes a socket

## Making client connection with server

In order to create a socket that connects to an other socket uses most of the functions from the previous code with the exception of a struct called HOSTENT

HOSTENT:

This struct is used to tell the socket to which computer and port to connect to. These struct can appear as LPHOSTENT, but it actually means that they are pointer to HOSTENT.

### Client key function

Most of the functions that have been used for the client to connect to the server are the same as the server with the exception of a few. I will just go through the different functions that have been used for the client.

gethostbyname(const char\* FAR name)

gethostbyname returns a pointer to a hostent structure as described under gethostbyaddr. The contents of this structure correspond to the hostname name. The pointer which is returned points to a structure which is allocated by the Windows Sockets implementation. The application must never attempt to modify this structure or to free any of its components. Furthermore, only one copy of this structure is allocated per thread, and so the application should copy any information which it needs before issuing any other Windows Sockets API calls. A gethostbyname implementation must not resolve IP address strings passed to it. Such a request should be treated exactly as if an unknown host name were passed. An application with an IP address string to resolve should use inet\_addr to convert the string to an IP address, then gethostbyaddr to obtain the hostent structure.

### Part 2 - Send / recieve

Up to this point we have managed to connect with our client to the server. Clearly this is not going to be enough in a real-life application. In this section we are going to look into more details how to use the send/recv functions in order to get some communication going between the two applications.

Factually this is not going to be difficult because most of the hard work has been done setting up the server and the client app. before going into the code we are going to look into more details the two functions

send(SOCKET s, const char FAR \* buf, int len, int flags)

send is used on connected datagram or stream sockets and is used to write outgoing data on a socket. For datagram sockets, care must be taken not to exceed the maximum IP packet size of the underlying subnets, which is given by the iMaxUdpDg element in the WSAData structure returned by WSAStartup. If the data is too long to pass atomically through the underlying protocol the error WSAEMSGSIZE is returned, and no data is transmitted.

recv(SOCKET s, const char FAR \* buf, int len, int flags)

For sockets of type SOCK\_STREAM, as much information as is currently available up to the size of the buffer supplied is returned. If the socket has been configured for in- line reception of out-of-band data (socket option SO\_OOBINLINE) and out-of-band data is unread, only out-of-band data will be returned. The application may use the ioctlsocket SIOCATMARK to determine whether any more out-of-band data remains to be read.

### part 3 - Read unknow size of data from client

Us mentioned earlier in part 2, we are noe going to expand on the way that we receive data. The problem we had before is that if we did not know the size of data that we where expecting, then the would end up with problems.

In order to fix this here we create a new function that receive a pointer to the client socket, and then read a char at the time, placing each char into a vector until we find the '\n' character that signifies the end of the message.

This solution is clearly not a robust or industrial way the read data from one socket to an other, because but its a way to start reading unknown length strings. the function will be called after the accept method.

**5. SYSTEM DESIGN**

**5.1 DATA FLOW DIAGRAM**

The data flow diagrams are very helpful in determining the flow of data in an application.

User

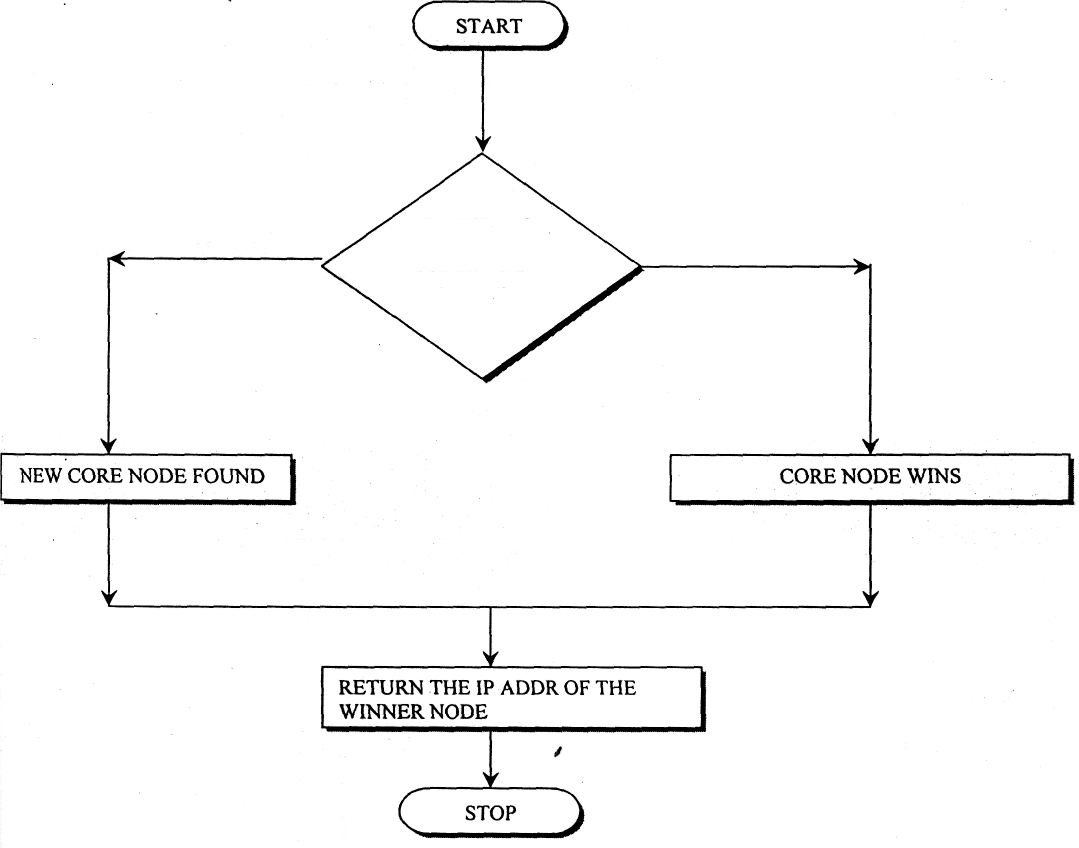
Route Discover

Check node

Core Node

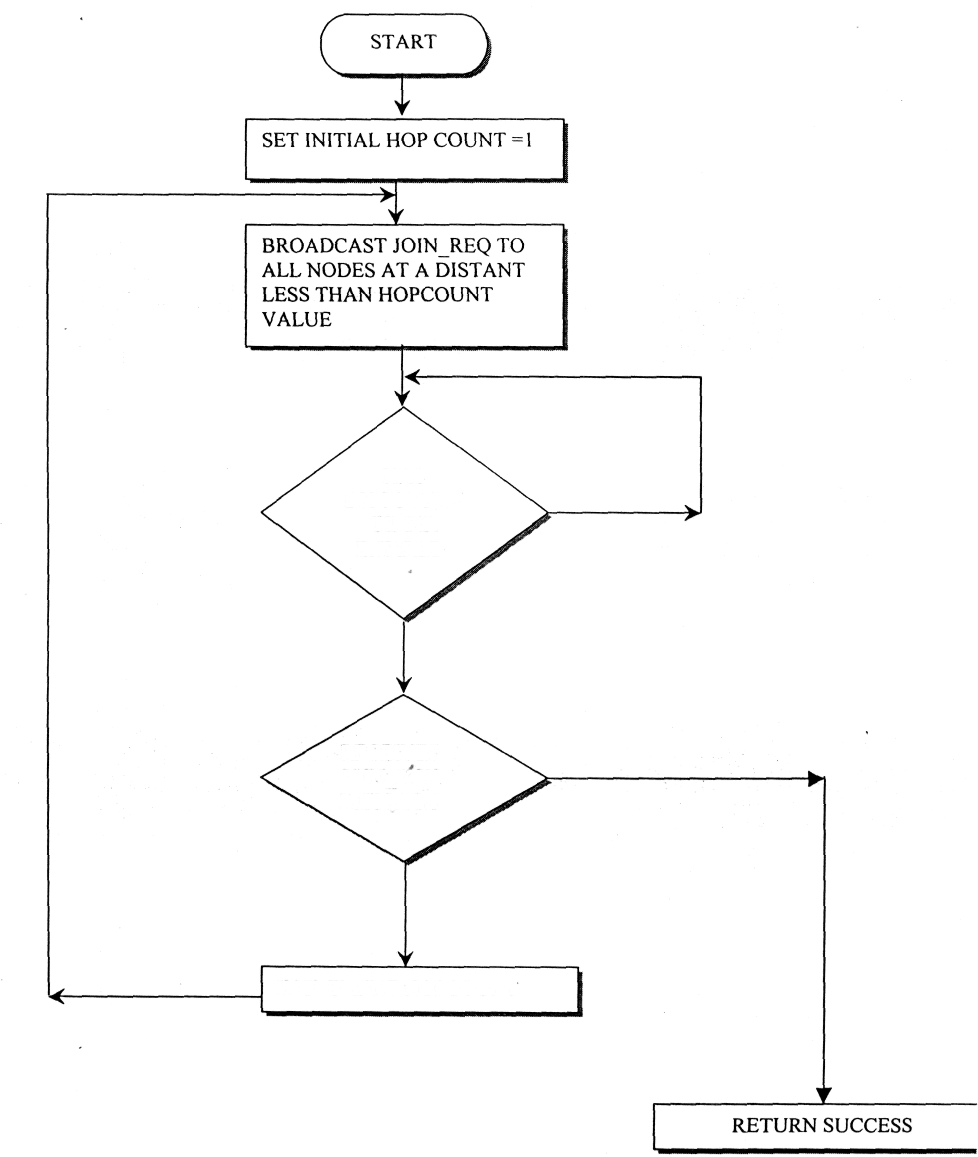
Recore

Transmission



IS MY.IPADDR

> COREIPADDR



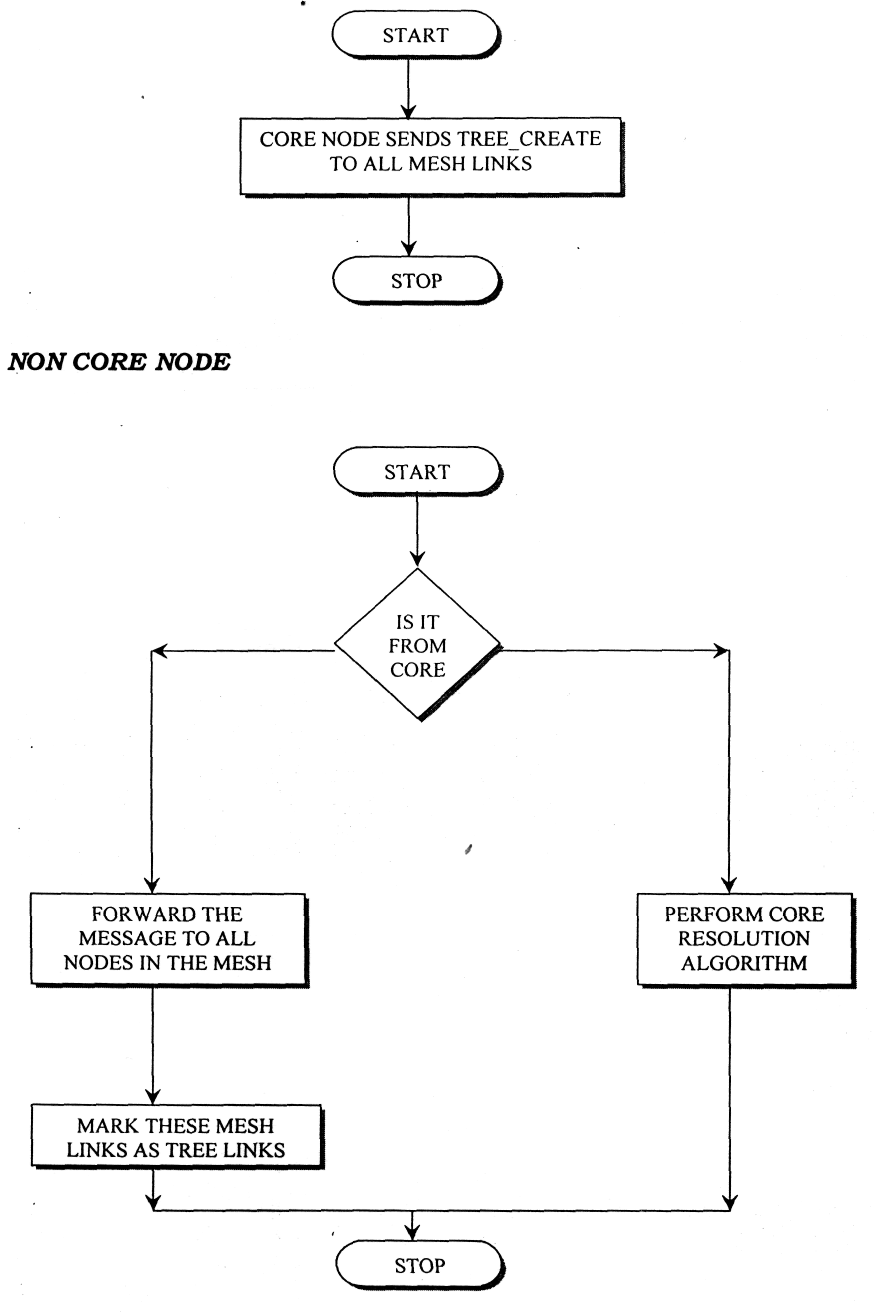
HAS DISCOVERY

TIMER EXPIRED

**Expand ring search**

RECEIVED JOIN\_ACK MESSAGE

INCREMENT HOPCOUNT

******

**Proposed System**

- LAR protocol --> distributed aproach

- each and every node act as server as well as a node

- hop-by-hop comm/n

- one by one node comm/n and finally reach the destination

- node damage will affect the whole network

**Algorithm**

1. Ring Search Algorithm

2. Core Resolution Algorithm

3. Tree Creation Algorithm

Ring Search Algorithm

- route discovery

**Core Resolution Algorithm**

- Core

- Recore

**Core**

- while one node wants to comm/t to another node

- it send request to the nearest node

- it check whether it is our dest node or not

- if it is our dest, it show the msg

- else recore the process

- find the destination

**Recore**

- choose another node

**Tree Creation Algorithm**

- apply when the route node failure

1. **Module Description**

**6.1 Expand Ring Search Algorithm**

To initiate the Route Discovery, node transmits a "Route Request" as a single local broadcast packet, which is received by (approximately) all nodes currently on the transmission range of, including node. Each Route Request identifies the initiator and target of the Route Discovery, and also contains a unique request identification determined by the initiator of the Request. Each Route Request also contains a record listing the address of each intermediate node through which this particular copy of the Route Request has been forwarded.

A

B

C

D

E

**6.2 Core Resolution Algorithm**

**Core Finding**

* If the core node and user defined destination node are same then it transmit the file to the corresponding destination node.
* After route discovery, to find the core node of the system then compares core node and user defined node in core class

**Re-Core Selection**

* To select the re-core node address for finding the destination address.
* Re-Core selection is used when the core process is not able to find the destination address.

**6.3 Tree Creation Algorithm**

When the intermediate node is fail, this process will execute and choose the another intermediate node for transmission. The failure node is identified by route request of corresponding end host, then the user to choose the another node as intermediate node so that the node failure will not affect the whole network

**6.4 Threshold Cryptography**

Threshold cryptography is the process of encrypting the plain text into cipher text and decrypting the cipher text into original plain text using the fixed key and math function. Converted data code can be securely transmitted over a network. Here the actual content is converted into Hex-code for transmission

**7. CONCLUSION**

Terminode routing aims to support location-based routing on irregular topologies with mobile nodes. It achieves its goal by combining a location-based routing method with a link state-based mechanism. Further, it introduces the concept of anchors, which are geographical points imagined by sources for routing to specific destinations, and proposes low overhead methods for computing anchors. Last, a special form of restricted search mode (Restricted Local Flooding, RLF), solves problems due to the inaccuracy of location information, in particular for control packets. The performance analysis shows that, in large mobile ad hoc networks, terminode routing performs better than MANET-like, or existing location-based routing protocols. It does so by maintaining its routing overhead low and by

efficiently solving location inaccuracies.

**8. REFERENCES**

* A Location-Based Routing Method for Mobile Ad Hoc Networks

By Ljubica Blazevic, Member, IEEE, Jean-Yves Le Boudec, Fellow, IEEE, and Silvia Giordano

<http://infoscience.epfl.ch/record/510/files/tnRouting.pdf>

* Tomas Krag and Sebastian Büettrich (2004-01-24).[*"Wireless Mesh Networking"*](http://www.oreillynet.com/pub/a/wireless/2004/01/22/wirelessmesh.html).O'Reilly Wireless Dev Center*. Retrieved 2009-01-20*.
* Location-Aided Routing (LAR) in mobile ad hoc networks

Young-Bae Ko and Nitin H. Vaidya Department of Computer Science, Texas A&M University, College Station, TX 77843-3112, USA