CHAPTER 7
DEVELOPMENT OF MTA AND MANET IMPLEMENTATION

This chapter presents the modules used in developing routing systems in Modified Termite Algorithm (MTA). Route Discovery, Route maintenance and Route failure handling modules are the three main modules which help in efficient routing. All three modules with basic functionality are explained in this chapter. However, detailed design which includes efficient pheromone decay technique in the form of controlled decay for efficient route maintenance and efficient local route repair for route failure handling are explained in chapter 5 and chapter 6.

The basic elements of a structure chart are modules. In the case of the software architecture, modules refer to individual subprograms. The representation differentiates between modules, predefined modules, data modules, macros, and multiple entry point modules.

By means of structure charts, the calling structure between modules (functions, subprograms) can be represented. These representations include sequence, selection, and iteration in connection with module calls. With each call, data and control flows can be listed separately. If required, the call parameters may be better specified in table-like footnotes. The structure charts also permit comments to the modules.

Structure charts show module structure and calling relationships. In a multi-threaded system, each task (thread of execution) is represented as a structure chart. Large structure charts are leveled into a stack of connected diagrams.

7.1 Structured chart of Modified Termite Algorithm (MTA)

This Structure Chart shown in Fig 7.1 depicts the structure and calling relationships in the main module. Each component of this structured chart is broken down into lower levels components.
7.1.1 Structured Chart for Route Discovery

This structure chart shown in Fig 7.2 depicts the structure and calling relationships in the Route Discovery. When a source wants to send data to destination and if the path to destination is not known, then this module is responsible for finding a path to destination. This module will also make sure that it will find those paths to destinations which can satisfy the required QoS (eq. bandwidth).
The Route discovery is responsible for generating a route between a given source and destination.

Working of this module is as follows:

- At the source node a RREQ packet is created and sent through the neighbors in the increasing order of cost value
- Any node, when it receives the RREQ, does the following
  - If current_addr = dest_addr
    - Construct the RREPLY packet and copies the trail content of RREQ packet, sends RREPLY packet to node prevHop from which it has received RREQ and updates the Pheromone table according to the delay
  - else
    - Add the current node address into the trail field & forward the RREQ packet to the neighbor nodes
- Any node, when it receives the RREPLY does the following
  - If current_addr = dest_addr
    - Update the pheromone table according to the delay and retrieve the packet from the data queue and insert into normal queue.
  - else
    - Update the pheromone table according to the delay and forward the packet to neighbor following the trail content

7.1.2 Structured Chart for Route Maintenance

This structure chart given in Fig 7.3 depicts the structure and calling relationships in the Route Maintenance. The route maintenance is done by processing data packets and periodic route decay.
The Route Maintenance module is responsible for the maintenance of the path that is generated during the discovery phase.

- Each node, when it receives a data packet, does the following
  
  if current addr = dest_addr
  
  { Extract data and set type = ack in data packet, remove the data content and send
    Acknowledge packet to prev_id }
  
  else
  
  { get pheromone values of all links using neighbor table, compute probability for all nodes in neighbor table and send packets to that link which has highest probability }

- Decay the table entries using exponential decay technique. If the pheromone value = 0.1 for any destination then delete the destination entry from the routing table.
7.1.3 **Structured Chart for Route Failure**

This Structure Chart given in Fig 7.4 depicts the structure and calling relationships in the Route Failure. In this structured chart, it is clearly seen that the calling relationships and the data that flows between each of the modules present.

![Structured Chart for Route Failure](image)

**Fig 7.4 Structure Chart for Route Failure**

Route Failure Handling module is responsible for handling the situation where a link fails in the network. The time out period given to receive the ACK packet is 10 seconds. The time ‘t’ for the ACK packet to arrive may depend on several cases like congestion of buffering of packets due to lack of transmitting capacity of intermediate nodes. If ‘t’ is great than 10 seconds , it generally means that the path which the ACK packet is compelling is unstable

There are two modules running for it. They are:

- Detects Route Failure - which detects if any links broken by receiving route_error. Once a node finds that a link has failed it transfers that message to the previous node.
• Find Alternative Path, which when receives the route_error message updates the routing table and then tries to find a different route to the given destination.

7.2 Component Design

The main components of the design are described in Figure 7.5. Major components are User Interface, data transfer, pheromone update etc. These components are explained in detail.

![Diagram of Component Design]

**Fig 7.5 Main components of the system**

7.2.1 Module 1: User Interface

This module provides a user Interface from where the user can initiate the topology and send the data packet through the network.
Introduction

This module acts as an interface to the user. Using this interface the user can give the name to the node to initiate the router. User can send hello packet to initialize the topology and data to other specified destination node.

Purpose

The purpose of the module is to provide a flexible interface to the user by which one can initiate the topology and send the data packets.

Functionality

The module does the check for the option selected and calls the appropriate module. Its main functionality is to provide a flexible interface to the user by which he can initiate the network topology by sending hello packet and data packet to the required destination. The functionality can be described with the following pseudo code.

Step 1: Display the interface screen
Step 2: Wait for the user input
Step 3: Accept user input
Step 4: if (choice = “Hello packet”)
   Ask for destination node
   Read the destination address
   Call the Hello packet generator function
   Send the Hello packet
else (choice=”Data packet”)
   Ask for destination node
   Read the destination address
   Ask for data to be sent
   Call the Data packet generator function
   Send the Data packet
Step 5: Display Routing table, Log information and packet received
Step 6: return

7.2.2 Module 2: Receive packet

This module is a thread that waits for the packet as long as the application runs.

Introduction

This is a thread that waits continuously for the packets as long as the application runs and puts the packets in the buffer when received.

Purpose

The purpose of this module is to store the incoming all packets in the buffer until buffer becomes full. This module listens to the specified port for the packet arrival. If a packet is received it puts it into the rear end of the buffer for the first in first out (FIFO) processing. The control packets like RREQ, REPLY, Data Acknowledgement are given higher priority over data packet (by placing them in multilevel priority queues).

Functionality

This module listens in the specified port for the packet arrival. If the packet is received it puts it into the rear end of the buffer for the first in first out (FIFO) processing. The control packets like RREQ, REPLY, data Acknowledgement are given higher priority over the data packet.

7.2.3 Module 3: Send packet

This module used to sends the packet to the given destination.
**Introduction**

This module is used to send the packet to the neighbouring node with given ip address and the port number of the node as the input.

**Purpose**

The purpose of this module is to communicate between the neighbouring nodes by means of socket programming.

**Functionality**

This module sends the packet to the specified port and ip address of the node by making use of UDP packets. It notifies the whether the transmission to the neighbor node fails or successful. If fails neighbor node is deleted from the routing table and packet is forwarded to the other available link.

**7.2.4 Module 4: Process Packet**

Reads the packet from the buffer and categorizes it into the different processing module.

**Introduction**

This module takes the packet from the buffer and reads the header content of the packet. Depending upon the header content it calls the different processing module

**Purpose**

Different types of the packet need different types of processing. Hence the packet needs to be identified by the type field of the header.

**Functionality**

This module does the categorization of the packets by identifying the type field in the header. The functionality can be described by the following pseudo code.
Step 1: Read the packet from the front end of the buffer.

Step 2: Extract the type field of the packet

Step 3: switch (type)

   {
      Case 0: Call the Process Hello Request
            break
      Case 1: Call the Process Hello Reply
            break
      Case 2: Call the Process Route Request
            break
      Case 3: Call the Process Route Reply
            break
      Case 4: Call the Process Data Packet
            break
      Case 5: Call the Process Data Acknowledgement
            break
      Case 6: Call the Process Route Error
            break
      default: print “invalid packet type”
   }

Step 4: Go to step 3 until buffer is empty

Step 5: return

7.2.5 Module 5: Process Route Request packet

Introduction

The module Process the Route Request Packet when received, by searching the requested destination in the pheromone table.
Purpose

When the node does not have the requested destination field in the routing table it constructs the Route Request Packet for the required destination. When these packets received by the other nodes they call this Process Route Request module for searching requested destination field in their routing table.

Functionality

This module extracts fields from the RREQ packet and then uses them to check in the processing of the route. It extracts the dest_id from the ant to check if the RREQ is destined to that node.

The functionality can be described by the following pseudo code.

Step 1: Read The Packet
Step 2: Extract the Source and Destination field
Step 3: Search Entry for the destination in pheromone table
Step 4: If entry not found
    Send the Route Request packet to the outgoing links i.e to its neighbors which are in the transmission range. This packet will propagate till either the destination is found or its address is found in intermediate nodes
Else
    Construct Route Reply Packet Update the Routing table entry forward the packet to previous node
Step 5: return

7.2.6 Module 6: Process Route Reply Packet

Introduction

The module Process the Route Reply Packet when received, by updating the entries of the pheromone table.
Purpose

The purpose of this module is to propagate information of the source node to the all intermediate node by updating their routing table.

Functionality

This module extracts fields from the RREPLY packet and then uses them to updating of the routing table. The functionality can be described by the following pseudo code.

Step 1: Read the Packet
Step 2: Extract the Source and Destination field
Step 3: Update the pheromone concentration for destination node
Step 4: If (destination_id=current node)
   Retrieve the packet from Data buffer
   Send the data packet to the neighbour which has higher pheromone concentration for the destination
   else
   Forward the RREPLY packet to neighbouring link
Step 5: return

As shown in Figure 7.7, data packets after finding the paths to destination (through route discovery phase) have to travel through the intermediate nodes before reaching the destination node.
7.2.7  Module 7: Process Data packet

Introduction

The module Process the Data Packet when received, by extracting the different fields of the packet.

Purpose

Data packets have to be travelled through the intermediate nodes before reaching the destination node. Hence the data packets have to be processed for forwarding to the neighbour nodes or displaying the content if the current node is the destination.

Fig 7.6 Data transfer flow representation
**Functionality**

This module extracts fields from the Data packet and then uses them in the routing. It extracts the dest_id from the data to check if the Data packet is destined to the current node.

The functionality can be described by the following pseudo code.

**Step 1:** Read the Data Packet

**Step 2:** Extract the Source and Destination field

**Step 3:** If ( Destination_id = Current_node )
   
   Display the content of the packet
   
   Construct and forward Data Acknowledgement
   
   Update the Routing table
   
   Go to step 6

   else

   Search for the Destination field in Routing table
   
   if(Destination exists)
       
       Find the neighbour which has higher probability for destination
       
       Forward the data packet to that neighbour
       
       Go to step 6

   else

   Construct the Route Request packet
   
   Store the packet in the Data buffer

**Step 4:** Receive Route Reply packet

**Step 5:** Remove the packet from the data buffer and forward.

**Step 6:** return
7.3 Pheromone updation

The pheromone updation is an important module in this research and is described in Figure 7.8. Pheromone updation not only takes place at the destination, it updates on the intermediate nodes on which the packet has traveled from source towards the destination. This updation will keep the route maintenance as simple as possible and will help in saving network resources efficiently. This updation will avoid route discovery process which could have been used at regular intervals.

![Diagram of Pheromone updation](image)

Fig 7.7 Pheromone updation

\[
\text{Decay:} \quad \text{Pheromone (edge)} = \text{Pheromone (edge)} \times (e^{-\tau_\theta})
\]
The flow chart shown in Fig 7.8 gives out the pictorial flow information regarding the pheromone decay technique and pheromone update techniques employed to handle congestion in MANETS. The trigger from the pheromone decrement to update the tables of nodes existing in the path under consideration and also depending on the time delay between the packet transmissions on the link under consideration are represented.

**Routing table effective decay**

Current application needs the routing table to be decayed periodically as it is derived analogy from the termite pheromone trail that will be evaporated in the environment as time passes. The method of pheromone decay employed is exponential. Effective decay is applied to our application as a substitution to periodic decay. That is, decay is done only when it is accessed - effective decay. This has reduced an extra thread to our application and overhead of synchronization of this table between these threads. Effective decay is implemented using timestamps. The difference between two access times is used to find the period to which it has to be decayed.

### 7.4 Implementation

Different pheromone decay techniques are implemented. Pre-built functions are used wherever possible. Customized programs are written in C/C++ to understand and fine tune the mathematical operations. However, using pre-built functions, as opposed to writing, debugging, and validating our own functions, will save a great deal of programming time (As shown in Appendix A). Routing in MANETs is a NP-hard problem; the development of Modified Termite Algorithm (MTA) has been implemented using Visual C# (As shown in Appendix B). It comes with various built-in tools which perform various functions. Different tools are used according to the need of the software. Visual C# provides support for socket programming. These together provide all the system calls necessary for creation of sockets and communication among them. Threads are also required for the implementation of this research. In the research there are several modules which run simultaneously. These modules will be running in the form of threads. Windows Pipes are used as a packet buffer queue which is used to store packets received. It also provides built-in
synchronization between threads. The platform chosen for the research is Windows (as based on Ad hoc networks there is a requirement of a wireless card on each of the nodes).

7.4.1 Environment and Parameters

The terrain area of 100 X 100 m with varying nodes 30, 40 and 50 are tested. The nodes are distributed in a random fashion over the given area. Each individual’s transmission range is 10 meters. The simulation time is 200 sec. The propagation model is a two ray ground model. The standard mobility model is the community based mobility model. As the proposed research gives a solution to uninterrupted data flow, testing is performed by considering more than one pair of sources and destinations trying to communicate by varying the packet sizes i.e. from 1-6 Kb. The speed of the nodes is tested for 5 m/s varying the pause time from 1-5 sec. Packets is only sent when a path exists between the source and destination. All packets have a time-to-live (TTL) of 15 hops. The performance of the algorithm is tested for both cases where nodes can communicate over congested and contention free media.

A number of different scenarios are simulated in order to compare the performance of the various pheromone accounting techniques over a variety of system parameters. Simulations are designed to test the effect of node mobility, pheromone sensitivity (F), and pheromone decay rate (τ) on the global performance metrics (Time difference between the current time and last successful transaction (Rank of Freshness of the link)).

7.4.2 Data structure of Packets

Packets are used for sending the data to the other nodes. These packets are also used for the maintenance of the route by increasing the pheromone concentration of the link. No special packets are needed for route maintenance. The different packet formats and their description are given below in Table 7.1, Table 7.2 and Table 7.3.
Table 7.1 Data Packet

<table>
<thead>
<tr>
<th>Type</th>
<th>Src Addr</th>
<th>Dst Addr</th>
<th>Prev Hop</th>
<th>Nxt Hop</th>
<th>Msg Id</th>
<th>Data Length</th>
<th>Data</th>
<th>Trail</th>
</tr>
</thead>
</table>

Table 7.2 Route Request / Reply / DataAck

<table>
<thead>
<tr>
<th>Type</th>
<th>Src Addr</th>
<th>Dst Addr</th>
<th>Prev Hop</th>
<th>Nxt Hop</th>
<th>Trail</th>
</tr>
</thead>
</table>

** Extra field Bandwidth for route request (4 Bytes) when QoS is provided to the end user

Table 7.3 Hello Sent / Hello Reply

<table>
<thead>
<tr>
<th>Type</th>
<th>Src Addr</th>
<th>Dst Addr</th>
<th>Timestamp</th>
</tr>
</thead>
</table>

- **Type (1 Byte):** This field is of type int. It is used for defining the type of the packet i.e. whether it is a data packet, an acknowledgement or an error message.

- **SrcAddr (4 Bytes):** This field is of type character array. It is used to store the IP address of the source node. Once the packet reaches the destination the acknowledgement created will store the srcAddr same as the destAddr of the packet.

- **DstAddr (4 Bytes):** This field is of type character array. It is used to store the address of the destination node. Once the packet reaches the destination the acknowledgement created will store the destAddr same as the srcAddr of the packet.

- **PrevHop (4 Bytes):** This field stores previous hop addresses.
- **NxtHop (4 Bytes)**: This field is used where the packet has to be transferred.

- **MsgId (4 Bytes)**: This field is used to eliminate the duplicate packets.

- **DataLen (5 Bytes)**: This field is of type int, used to store the length of the data packets.

- **Data (user defined)**: This is of type character array. This field stores the data content.

- **Trail (User defined)**: This field is of type character array; used to store the path that packet has traversed.

### 7.4.3 Evaluation Metrics

**End to End delay (ms)**: Includes buffering delay/queuing delay, propagation delay/transfer delay, retransmission delay etc.

**Throughput**: The average rate of successful data delivery over a communication channel.

**Routing overhead**: Routing overhead is the ratio of number of control packets required for delivery of data.